Høgskolen i Østfold

MASTEROPPGAVE

En kvalitativ innholdsanalyse av programmeringsoppgaver i lys av algoritmisk tenking

Lisbet Johanne Nordmo & Henriette Lund Myhrvold

15. mai 2022

Grunnskolelærerutdanning 1-7

Fakultet for lærerutdanninger og språk

Sammendrag

I denne masteroppgaven har vi rettet søkelyset mot programmeringsoppgaver i lærebøker. Formålet med studien var å se hvordan programmeringsoppgavene kan legge til rette for algoritmisk tenkning. Dette temaet ønsket vi å skrive om på bakgrunn av at programmering i skolen har blitt mer synlig som et resultat av fagfornyelsen LK20. Programmeringens nye tilstedeværelse i læreplanen gjør dette prosjektet også nyttig for oss selv som fremtidige lærere. For å kunne se på programmeringsoppgaver var det naturlig å bruke lærebøker i matematikk som studiens datamateriale. Ytterligere falt valget på 6. trinn da vi skulle velge ut lærebøker og trinn vi skulle forske på.

For å gjennomføre datainnsamlingen ble det benyttet kvalitativ innholdsanalyse som forskningsmetode. Til utdypning av hvordan innholdet i lærebøkene ble analysert, ble det gjennomført en horisontal og en vertikal analyse. Rammeverket vi har brukt i den vertikale analysen for å finne svar på hvordan programmeringsoppgaver kan legge til rette for algoritmisk tenking er Barefoot Computing's (u.å.-a) modell om den algoritmiske tenkeren. Denne modellen inneholder seks konsepter og fem fremgangsmåter som omhandler algoritmisk tenking. Dermed kan denne modellen brukes for å se på sammenhengen mellom programmering og algoritmisk tenking. Dette gjøres ved å se på hvilke av konseptene og fremgangsmåtene som arbeides med i hver av programmeringsoppgavene.

I arbeidet med denne masteroppgaven ble resultatet at samtlige av konseptene og fremgangsmåtene blir arbeidet med i en eller annen av programmeringsoppgavene i lærebøkene. Vi så at programmeringsoppgavene la til rette for arbeid med komponentene og fremgangsmåtene innen algoritmisk tenking ved at alle disse oppgavene innebar at elevene skulle arbeide med minst et konsept og en fremgangsmåte. Samtidig har vi sett en del fellesnevnere som gjentar seg i flere av programmeringsoppgavene. Videre funn omhandlet hvordan blant annet selve læreboka, de ulike komponentene av algoritmisk tenking og ulike programmeringstyper kan ha innvirkning for tilretteleggingen for algoritmisk tenking.

Abstract

In this master's thesis, we have directed the spotlight on programming tasks in textbooks. The purpose of the study was to see how the aforementioned tasks can facilitate algorithmic thinking. We wanted to write about this topic on the basis that programming in schools has become more visible with the help of the subject renewal LK20. The presence of programming in the new curriculum makes this project useful for ourselves as future teachers. In order to be able to look at programming tasks, it was natural to use textbooks in mathematics as the study's data material. Furthermore, the choice fell on 6th grade textbooks as the basis of our data collection and research.

To carry out the data collection, qualitative content analysis was used as a research method. To elaborate on how we decided to analyse the content of the textbooks through a horizontal and a vertical analysis was carried out. The framework we have used in the vertical analysis to find answers to how programming tasks can facilitate algorithmic thinking is Barefoot Computing's (u.å.-a) model of the algorithmic thinker. This model contains six concepts and five methods that are about algorithmic thinking. Thus, this model can be used to look at the connection between programming and algorithmic thinking. This is done by looking at which of the concepts and methods are worked on in each of the programming tasks.

The result while working on this master's thesis, was that all of the concepts and approaches is being worked on in one or another of the programming tasks in the textbooks. We saw that the programming tasks facilitated work with the components and methods within algorithmic thinking in that all these tasks meant that the students had to work with at least one concept and one method. At the same time, we have seen a number of common denominators that repeat themselves in several of the programming tasks. Further findings dealt with how, among other things, the textbook itself, the various components of algorithmic thinking and different types of programming can have an impact on the preparation for algorithmic thinking.

Forord

Dette masterprosjektet har til hensikt å forbedre vår kunnskap om programmering og algoritmisk tenking i skolen. Denne nye kunnskapen ønsker vi på bakgrunn av at programmering er forholdsvis nytt i skolen og vi som nyutdannede lærere er nødt til å tilegne elevene denne kunnskapen. Under arbeidet med denne studien har vi utviklet vår forståelse for emnet, og vi ser frem til å ha nytte av det i hverdagen som fremtidige lærere. Vi håper at vår studie kan bidra til at andre kan få innblikk i hvordan programmering kan brukes til å arbeide med problemløsningsmetoden algoritmisk tenking.

Vi vil rette en spesiell takk til vår veileder, for gode diskusjoner og konstruktive tilbakemeldinger som kom godt til nytte underveis i prosjektet. I tillegg ønsker vi å takke alle medstudenter som også har bidratt til gode tilbakemeldinger og støtte både på masterseminarer og ellers under skriving på skolen. Videre vil vi takke familie og samboere for tålmodighet og god motivasjon. Hjertelig takk!

Avslutningsvis ønsker vi å rette en takk til hverandre for et godt, lærerikt og hyggelig samarbeid i denne krevende, men spennende perioden.

Halden, 15. mai 2023
Lisbet Johanne Nordmo og Henriette Lund Myhrvold

Innhold

Oversikt over tabeller og figurer

1. Innledning

I dagens samfunn står «21st century skills» sterkt. Ananiadou og Claro (2009, s. 8) definerer «21st century skills» som de ferdighetene som kreves for å fungere som effektive arbeidere og medborgere i kunnskapssamfunnet i det 21. århundre. De gitte ferdighetene befinner seg i ulike områder. Eksempler på områder som Ananiadou og Claro (2009, s. 9) trekker frem er å kritisk hente frem og organisere informasjon fra digitale miljøer. De legger til at det også er nyttig å kunne bruke og omformulere denne informasjonen for å skape nye ideer. En av disse ferdighetene helt konkret er kommunikasjon, og enda mer spesifikt, samarbeid. Det å kunne kommunisere spiller en viktig rolle i det som handler om å være en del av samfunnet og den digitale kulturen (Ananiadou og Claro, 2009, s. 10). I hvilken grad hver enkelt klarer dette avhenger av evnen til å være en del av grupper over nettet og kommunisere digitalt, samt delta i samarbeid og arbeid i team (Ananiadou og Claro, 2009, s. 10).

En annen spesifikk ferdighet de trekker frem for å oppnå «21st century skills» er problemløsningsferdigheter (Ananiadou & Claro, 2009, s. 9). En type problemløsningsmetode er algoritmisk tenking (Utdanningsdirektoratet, 2019, s. 1). Algoritmisk tenking er noe som trer frem i LK20, som en ferdighet elevene burde inneha (Kunnskapsdepartementet, 2019b, s. 2). De har valgt å knytte det til matematikkfaget som en del av kjerneelementet «Utforsking og problemløsning.» Dette kjerneelementet hører blant annet til kompetansemålet som lyder: «bruke variablar, lykkjer, vilkår og funksjonar i programmering til å utforske geometriske figurar og mønster» (Kunnskapsdepartementet, 2019b, s. 10). Dette er nok ikke tilfeldig om det ses i lys av Nouri et al. (2020, s. 3) som trekker frem at programmering er en av aktivitetene som vanligvis fremkaller deler av algoritmisk tenking.

For å kunne arbeide med utvikling av algoritmisk tenkning i skolen, kan det være en fordel å se på innholdet i dagsaktuelle lærebøker. Dette er fordi Brehmer et al. (2015, s. 578) trekker frem at lærebøker står sterkt i skolens undervisning. Alle elever har et forhold til lærebøker. I boka til Skjelbred et al. (2017, s. 9) om norsk lærebokhistorie legger de til at elever har hatt et forhold til tekster i lærebøker i mer enn 250 år. Dette mener vi viser til relevansen til å forske på lærebøker. Videre skriver Skjelbred et al. (2017, s. 9) at det har skjedd en stor endring i løpet av disse årene. Blant annet at læreboken tidligere var en økonomisk belastning for foreldre, mens nå er ressursene skolens ansvar. Disse ressursene gjelder både når det kommer

til variasjonen av lærebøker og læremidler, men også tilgjengeligheten (Skjelbred et al., 2017, s. 9). Utviklingen av lærebøker skjer i takt med samfunnet. I dagens samfunn endrer teknologien seg raskt, og har en viktig rolle i alles liv (Nouri et al., 2020, s. 1). Dermed kan det se ut til at teknologien også vil påvirke lærebøkene både ved at lærebøkene blir mer teknologiske, men også ved at innholdet endrer seg mot teknologien slik at elevene lærer mer om det.

Med utgangspunkt i det som er nevnt ovenfor er hovedmålet med dette prosjektet å finne ut av hvordan programmeringsoppgaver i lærebøkene i matematikk på 6. trinn kan legge til rette for algoritmisk tenking. Videre i innledningen skriver vi om bakgrunnen for valg av tema. Deretter blir problemstilling og avgrensinger beskrevet. Avslutningsvis kommer det en begrepsavklaring, og til slutt i kapittelet en disposisjon over resten av masteroppgaven.

1.1 Bakgrunn for valg av tema

Da vi skulle velge overordnet tema for masteroppgaven vår, kom vi raskt frem til at vi ville skrive om programmering i skolen, spesielt på 6. trinn hvor det kompetansemålet som ble nevnt over befinner seg (Kunnskapsdepartementet, 2019b, s. 10). Ønsket om å skrive om programmering utviklet seg ved at programmering kommer mer inn i skolen ved den nye læreplanen LK20 (Kunnskapsdepartementet, 2018). Samtidig har vi selv opplevd at det ikke er vektlagt i utdanningen. Dette er dog ikke rart på bakgrunn av at vårt utdanningsløp startet før LK20 ble ferdigstilt og det var vanskelig for høgskolen å vite at programmering skulle inn i læreplanen i matematikk. Programmering i skolen er også noe som har vekket egen interesse hos oss begge, og vi syntes det virket interessant å se på om programmering kan være med på å legge til rette for algoritmisk tenking. Det å få et innsyn i hva som befinner seg av programmeringsoppgaver i lærebøkene virket også spennende for å se hva slags ressurs vi som lærere har i lærebøkene når det gjelder programmering og hvordan disse oppgavene legger til rette for programmering. Dette sett i lys av at Stigberg og Stigberg (2020, s. 494) påpeker at det er viktig at læreren har kompetanse innen programmering når det skal undervises i dette.

Vinnervik og Bungum (2022, s. 384) trekker frem at en fremtreden praksis innen algoritmisk tenkning i både den norske og svenske læreplanen er at det blir representert innen programmering. Programmering er heller ikke et eget fag, men blir brukt som et verktøy og

metode for å tilegne seg faginnhold i ulike fag (Vinnervik & Bungum, 2022, s. 384). Noe av argumentasjonen for at programmering kom inn i den nye læreplanen var at den legger til rette for at elevene får øve på algoritmisk tenking (Berg, 2021, s. 42). Lærebøker ble raskt et sentralt datamateriale for denne masteroppgaven ved for eksempel at flere forskere nevner at mange lærere støtter seg til sitt læreverk i sin undervisning (Bellens et al., 2020, 192). Dermed har læreverket en tendens til å påvirke elevenes utdanning (Bellens et al., 2020, 192). I tillegg vil læreboken mest sannsynlig reflektere det som står i læreplanen (Charalambous et al., 2010, s. 119). Dette er på bakgrunn av at det er læreplanen som er et styringsdokument som inneholder noe om elevenes ferdighetsutvikling i løpet av 13-årig skoleløp (NOU 2014:7, s. 72). Dermed vil læreboken for 6. trinn også mest sannsynlig inneholde hva elevene på 6. trinn skal kunne. Det støttes opp ved det Tønnessen (2013, s 149) påpeker om at læreboken er en pedagogisk fortolkning av læreplanen i ulike fag. Dette ga oss implikasjoner på at læreboken er en relevant kilde for å hente ut virkelighetsnær informasjon om hvordan programmeringsoppgaver kan legge til rette for algoritmisk tenking.

Å undersøke hvordan programmeringsoppgaver i ulike trykte lærebøker for 6. trinn kan legge til rette for algoritmisk tenking, kan være et bidrag til andre som skal integrere algoritmisk tenking i sin undervisning. De kan da få et innblikk i hvilken måte, eventuelt ikke, de kan bruke programmering til å legge til rette for algoritmisk tenking. Andre aspekter som kan være behjelpelig for fagfeltet er for de som skal være med på å velge hvilke læreverk som skal tas i bruk på de ulike skolene. De kan få et innblikk i hva som finnes av programmeringsoppgaver i de ulike lærebøkene og om de trenger flere ressurser enn bare læreboken. Samtidig er det hensiktsmessig å nevne at programmering er bare en del av helheten i matematikkfaget, og i et utvalg av lærebøker vil man se på helheten av innholdet i læreboken. Likevel vil dette være et bidrag i form av at det kan gi et innblikk i hvordan lærebøkene fremstiller selve programmeringsdelen slik at de kan ta det med i betraktning sammen med helheten når de skal ta valget. Den vil også være nyttig for oss selv, ved at når vi skal undervise om programmering i matematikk og algoritmisk tenking har vi mer oversikt over hva som finnes av ulike oppgaver og arbeidsmåter. Dermed kan vi få noe innsikt i antallet og hva slags oppgaver læreboken inneholder. Eventuelt ser vi antallet oppgaver som kan finnes andre steder eller lages for å arbeide med kompetansemålet om programmering i matematikk på 6. trinn og få integrert algoritmisk tenking.

1.2 Problemstilling og avgrensninger

Ut ifra det som tidligere ble nevnt i delkapittelet over har vi utformet en problemstilling som lyder følgene:

> *Hvordan kan programmeringsoppgaver i lærebøker i matematikk på 6. trinn legge til rette for algoritmisk tenking?*

I denne studien er det foretatt noen avgrensninger. Grunnet masterens omfang kan vi ikke velge for mange lærebøker for å finne svar på problemstillingen. Vi kom frem til at tre lærebøker ville være tilstrekkelig for å svare på problemstillingen. Likevel inneholdt to av verkene vi så på to trykte lærebøker. Derfor hadde vi noen kriterier for utvalget av bøkene. Ved et raskt overblikk, utforsket vi hvilke temaer de ulike bøkene i læreverkene tok for seg. Et av kriteriene var dermed at læreboken måtte inneholde temaer som omhandlet programmering, slik at vi var mer sikre på at vi kom til å finne programmeringsoppgaver. A-bøkene i to av læreverkene vi så på inneholdt ikke temaet programmering, og det ble derfor naturlig at vi valgte B-bøkene i settet.

Videre var det nødvendig å ha kriterier for hva slags oppgaver som skulle undersøkes, slik at studien ikke ble for omfattende. De gjeldene kriteriene omhandler at oppgaveteksten var nødt til å inneholde ord eller begreper som er knyttet til programmering. Eksempler på dette kan være program, kode, algoritme osv. Kriteriene er også satt slik at det er større sannsynlighet at både elev og lærer er bevisst på at det er programmering og matematikk de holder på med. Denne avgrensningen ble gjort på bakgrunn av at flere forskningsartikler viser til at elever ikke alltid klarer å koble sammen matematikk og programmering. Et eksempel på noen som har funnet ut av dette i deres forskningsstudium er Stigberg og Stigberg (2020, s. 491-492). Andre studier viser til at også lærere ikke ser koblingen mellom matematikk og programmering (Kilhamn, 2021, s. 286). Dermed ønsket vi å begrense oppgavene som skal analyseres til de oppgavene hvor det er tydelig at det arbeides med programmering i matematikkboka. Samtidig sikrer denne avgrensningen også at forfatternes formål med oppgavene blir ivaretatt. Med dette mener vi at ved bruk av disse begrepene vil det være tydelig at det er programmering oppgavene er ment til å handle om.

1.3 Begrepsavklaringer

Dette delkapittelet inneholder redegjørelser for begreper som blir brukt hyppig igjennom hele studien. Her vil vi ta for oss disse begrepene: Programmering, algoritmisk tenking, lærebok og kode.

Programmering:

Gjennomgående i teksten vil det være naturlig å bruke begrepet programmering mye. Det finnes ulike definisjoner på programmering og en av de er satt sammen av Sevik (2016, s. 9). Hun beskriver programmering som mer enn bare å skrive inn koden i et dataprogram som kan kjøres. Det handler også om hvordan selve koden blir skapt (Sevik et al., 2016, s. 9), "Det vil si prosessen fra å identifisere et problem og tenke ut mulige løsninger på problemet, til å skrive kode som kan forstås av en datamaskin, og å feilsøke og kontinuerlig forbedre denne koden" (Sevik et al., 2016, s. 9). Forsström og Kaufmann (2018, s. 19) bruker også ordet prosess i sin definisjon. De definerer programmering som prosessen som kobles til hvordan utvikle og skape ulike koder som en data kan oppfatte og bruke til å utvikle nye egenskaper, som kan løse problemer og utføre spesifikke oppgaver (Forsström & Kaufmann, 2018, s. 19). Med bakgrunn i at dette gjelder programmering på 6. trinn, har vi ut ifra disse eksemplene på definisjoner valgt å definere programmering slik. Å programmere innebærer å analysere et problem, designe en løsning, som kan forstås av en datamaskin eller menneskelig atferd, og deretter ta i bruk løsningen.

Algoritmisk tenking:

Algoritmisk tenking innebærer i henhold til Wing (2006, s. 33) å løse ulike problemer, skape systemer og forstå menneskelig atferd ut ifra de grunnleggende konseptene innen IT-forskning. Utdanningsdirektoratet (2019) definerer i korte trekk algoritmisk tenking som en problemløsningsmetode. For å utdype beskriver de det som en måte å tilnærme seg problemer som er systematisk både i problemformulering og under forslag av løsninger. Med bakgrunn i disse definisjonene har vi valgt å definere algoritmisk tenking som en problemløsningsmetode som består av å løse ulike problemer på en systematisk måte ved hjelp av ulike tilnærminger innenfor IT-forskningen.

Lærebok:

For å kunne forklare begrepet lærebok er det nyttig å begynne med hva et læreverk er. Et læreverk er ifølge Utdanningsdirektoratet (2021, s. 2) en pakke med ulike læremidler som

læremiddelprodusenter kan tilby for å dekke en bredere del av læreplanen. Den delen av pakken, eller gruppen med læremidler som er trykte omtales tradisjonelt sett som læreverk. Digitale ressurser har ofte andre begreper som for eksempel læringsunivers (Utdanningsdirektoratet, 2021, s. 2) Et læreverk kan bestå av mange deler. Et eksempel på en av disse delene er læreboken. Videre i denne masteren når begrepet lærebok blir brukt er det trykte lærebøker det refereres til. En måte å forklare lærebøker på er, som navnet tilsier, en bok som er skapt for læring (Nylenna, 2017, s. 89). Videre om læreboken påpeker Nylenna (2017, s. 89) at den skal formidle, organisere og systematisere kunnskap. Han føyer til at læreboken er med på å avgrense et fagfelt og definere pensum og læringsmål for utdanning. Med andre ord er lærebokens oppgave å formidle en tolkning av det som står i landets læreplan (Valverde et al., 2002, s. 2). Til slutt er det verd å nevne at læreboken brukes som hjelpemiddel i undervisningen (Nylenna, 2017, s. 89).

Koding:
Det å kode noe handler om selve prosessen der du skriver inn instruksjonen til dataprogrammet, altså oversetter planen for å løse oppgaven til et språk som programmet forstår (Mannila, 2017, s. 177). Når begrepet «koding,» eller variasjoner av det, blir brukt videre i teksten er det denne betydningen vi viser til.

1.4 Disposisjon

Vi har valgt å dele masteroppgaven inn i 6 hovedkapitler. Under vil vi forklare kort hva hvert hovedkapittel vil inneholde.

I Kapittel 1 ble bakgrunn for valg av tema presentert, samt en presentasjon av problemstilling. Kapittelet inneholder også en presentasjon av avgrensninger som ble gjort under utviklingen av oppgaven. Til slutt i dette kapittelet er det en disposisjon som vil fungere som en oversikt over hva som kommer videre i oppgaven.

Videre inneholder Kapittel 2 en gjennomgang av teorien og litteraturen som har blitt brukt for å belyse problemstillingen til masteroppgaven. Først presenteres tidligere forskning om programmering i matematikkundervisning. Deretter vil tidligere forskning på tre ulike typer programmering, tekst-, blokk- og analog programmering, bli redegjort for. Påfølgende

kommer et delkapittel om tidligere forskning på algoritmisk tenking i skolen. Videre i masteroppgaven vil tidligere forskning på undervisning som omhandler både programmering og algoritmisk tenking bli presentert. Avslutningsvis vil det komme et delkapittel om tidligere forskning på lærebøker.

I Kapittel 3 presenteres metoden til masteroppgaven. Metoden vi har valgt er kvalitativ innholdsanalyse, mer utdypet en horisontal og vertikal analyse. Før analysestrategiene blir forklart, blir det redegjort for hvordan vi har valgt ut læreverk, samt noe informasjon om dem. Etter lærebøkene er presentert vil den horisontale analysen og hvordan vi har tenkt til å gjennomføre denne redegjøres for. Deretter vil en forklaring på vertikal analyse og hvordan vi vil gjennomføre denne komme. Avslutningsvis kommer to delkapitler til. Det ene er reliabilitet og validitet, hvor vi vil si noe om hva vi har gjort for å ivareta oppgavens gyldighet og pålitelighet. Det andre omhandler forskningsetiske betraktninger som er relevant for denne masteroppgaven.

Analyse er det overordnede temaet for Kapittel 4, hvor funn som har blitt gjort i undersøkelsen vil bli presentert og gjennomgått. Først vil vi ta for oss funn i den horisontale analysen, hvor vi hentet frem bakgrunnsinformasjon om bøkene samt plukket ut programmeringsoppgavene som skal analyseres i den vertikale analysen. Dette kapittelet er delt opp i henhold til bøkene slik at hver bok står for seg. Alle programmeringsoppgavene vi fant i den horisontale analysen ble analysert i den vertikale analysen ved hjelp av modellen til Barefoot Computing's (u.å.-a). I delkapittelet om den vertikale analysen vil vi ikke trekke frem alle oppgavene vi har analysert, men et utvalg for å vise bredde i hvilke konsepter og fremgangsmåter som blir arbeidet med.

Kapittel 5 inneholder en drøfting. Funnene presentert i Kapittel 4 Analyse, drøftes opp mot teorien som ble presentert i Kapittel 2 Teorigrunnlaget. Her er det konseptene og fremgangsmåtene innen algoritmisk tenking, og hvordan de trer frem i oppgavene som er fokuset. Kapittelet er delt inn i temaer for å skape sammenheng. Disse temaene er programmeringsoppgaver i lærebøkene, forekomsten av de ulike komponentene, programmeringstyper i lærebøkene i lys av algoritmisk tenking og til slutt, tverrfaglighet.

Til slutt tar kapittel 6 for seg en samling av oppsummerende kommentarer. Kapittelet deles inn i to delkapitler. Det første delkapittel inneholder en konklusjon og et svar på

problemstillingen, og heter derfor konklusjon. Det siste kapittelet i denne studien heter veien videre, hvor vi beskriver hvordan vi ville videreført denne forskningen.

2. Teorigrunnlaget

I dette kapittelet presentere teori som er relevant for å svare på studiens problemstilling. Problemstillingen er som nevnt tidligere i kapittel 1.2 Problemstilling og avgrensing: hvordan kan programmeringsoppgaver i lærebøker i matematikk på 6. trinn legge til rette for algoritmisk tenking? Derfor vil vi først starte med å presentere programmering, og hva det innebærer når det kommer til matematikk. Videre legger vi frem ulike programmeringstyper, samt algoritmisk tenking og hvordan algoritmisk tenking kan ses på i forhold til programmering. Deretter beskrives forskning innenfor lærebøker og avslutningsvis redegjøres det for studiens rammeverk.

2.1 Programmering i matematikk

Vi kan se at det har vært fokus på programmering i mange tiår. Allerede på 1980-tallet kom Papert ut med boken, *Mindstorms: childrens, computers and powerful ideas* (Papert, 1980). I boka skriver han litt om hvordan elever kan lære geometri i matematikk ved å anvende "Turtle" (Papert, 1980, s. 55). "Turtle" eller «skilpadde» er en datastyrt figur, som eksisterer i «LOGO-samfunnet», hvor LOGO er dataspråket der eleven kommuniserer med Skilpadden (Papert,1980, s. 11). Ifølge Bocconi et al. (2016, s. 39) var Papert en pioner når det kommer til å gjøre programmering tilgengelig for barn. Videre skriver de at han på 60-tallet startet med å introdusere Logo-programmet. Andre programmer i dag anvender enda deler av hans nøkkeldesign (Bocconi et al., 2016, s.39). I delkapittel 2.2 Ulike programmeringstyper, vil vi utdype om ulike programmeringstyper, og Logo vil bli videre forklart.

2.1.1 Hva er programmering?

Programmering er som nevnt i innledningen et tema som har kommet for fullt inn i skolen med LK20. Vi har tidligere i kapittel 1.3 definert hva programmering er i henhold til vår oppgave. For å gjenta det innebærer programmering det å analysere et problem, designe en løsning, som kan forstås av en datamaskin eller menneskelig atferd, og deretter ta i bruk løsningen. For å utdype hva programmering generelt omhandler, krever det ulike ferdigheter av programmereren (Forsström & Kaufmann, 2018, s. 19). Disse ferdighetene innebærer kompetanse innen ulike programmeringsspråk, logikk og kunnskap om spesialiserte algoritmer. Andre egenskaper som er fordelaktige å ha er for eksempel å kunne gjenkjenne algoritmer, vurdere troverdigheten og effektiviteten av dem. Dette krever at

programmereren har kompetansen til å forstå et problem, analysere, det for så å finne en løsning på problemet (Forsström & Kaufmann, 2018, s. 19).

Ifølge Grover og Pea (2013, s. 39) er det stor enighet om at programmering er relatert til utviklingen av algoritmisk tenking. Derfor mener myndighetene i flere land i Europa at det er en viktig egenskap å ha i det digitale samfunnet (Forsström & Kaufmann, 2018, s. 19). Overalt i samfunnet rundt elevene finnes det kode-baserte systemer, som for eksempel telefonene deres og robotstøvsugere (Lee, 2020, s. 266). Elevene anvender og styrer disse elektroniske systemene uten å vite at det egentlig er koding de driver med (Lee, 2020, s. 266). Koding involverer flere forskjellige ferdigheter og prosesser innenfor både forskning og matematikk, som for eksempel romforståelse, tallforståelse, problemløsningsferdigheter, utforskingsferdigheter og logisk tenking (Lee, 2020, s. 266).

2.1.2 Programmering i undervisning

Et annet forskningsfelt innenfor programmering omhandler bruk av programmering i undervisning. Kilhamn et al. (2021, s. 287) viser til at det er forskjell på undervisning om programmering i dag i forhold til tidligere. For en lærer som ønsket å gjennomføre undervisning om programmering før, var det flere betingelser som måtte være på plass. Eksempler på områder som måtte tas i betraktning var økonomi, tid til forberedelser, kunne håndtere maskinvarens kompleksitet, i tillegg til programmeringsspråket og hvor lang tid det tok å koble opp maskinen. Kilhamn et al. (2021, s. 287) viser videre til at det er bedre forutsetninger for denne typen undervisning i dag ved at datamaskinene er raske og at programmeringsspråkene er enklere å forstå og lære seg. En elev i dag kan relativt raskt skape, teste og endre koden sin takket være visuell tilkobling til bildeskjerm (Kilhamn et al., 2021, s. 287). Videre i sin forskning forklarer de at lærerne sliter med å se en tydelig sammenheng mellom matematikk og programmering. Likevel er de positive til at programmeringen skal være i matematikkfaget i læreplanen (Kilhamn et al., 2021, s. 286). I en annen studie gjennomført av Stigberg og Stigberg (2020, s. 483) så de på hvordan lærere introduserte programmering i matematikkundervisning. Der fant de ut at elevene i studien hadde vanskeligheter med å se matematikken i programmeringen. Derfor konkluderte de med at en viktig faktor når elevene skal programmere er at både læreren og eleven forstår hva programmering er i sammenheng med matematikk og hvordan det kan bli brukt i en matematisk kontekst (Stigberg & Stigberg, 2020, s. 494). En av lærerne i denne studien nevnte også at en fellesnevner mellom matematikk og programmering er algoritme og struktur

(Stigberg & Stigberg, 2020, s. 491). Dette er fordi det krever en spesifikk rekke av instruksjoner og struktur elevene skal følge når de programmerer. Hvis elevene øver på programmering, kan en faktor av dette være at eleven lettere vil forstå matematiske strukturer (Stigberg & Stigberg, 2020, s. 491). De stiller spørsmål ved hvor mye programmering som er ønsket i skolen og om man trenger matematisk kunnskap for å programmere. Videre stiller de også spørsmål ved om programmering kan brukes for å tilegne seg ferdigheter innen problemløsning (Stigberg & Stigberg, 2020, s. 491).

Videre i studien til Kilhamn et al. (2021, s. 303) som omhandler programmering i matematikkundervisning, gjorde de funn som tydet på at programmeringen ikke alltid er en ressurs for å lære matematikk. De kom frem til at 68% av alle undervisningstimene omhandlet enten kun programmering eller programmering i samspill med matematikk som elevene hadde kunnskap om fra før. Dette er eksempler på at matematikken utgjør et rammeverk rundt programmeringen for å anerkjenne dens eksistens i læreplanen, uten å bruke programmeringen som en ressurs i matematikken (Kilhamn et al., 2021, s. 303). Kilhamn et al. (2021, s. 304) legger til at en forklaring på dette kan være at når programmering er nytt selv for lærerne vil det av nødvendighet handle mest om hva det er og hvordan det gjøres. De skriver videre at den dagen lærerne har utviklet egen kompetanse og selvsikkerhet innen emnet vil det legges mer vekt på programmering som verktøy for problemløsning, utregning og utforsking av matematikk. De legger til at det også vil være annerledes om noen år når programmering har vært i læreplanen lenger slik at alle elevene møter på programmering hvert år. Kilhamn et al. (2021, s. 304) legger frem at for elevene som begynner med programmering samme år som de begynner på skolen vil ha bedre utbytte (Kilhamn et al., 2021, s. 304)

En annen studie har sett på hvilken virkning det har å begynne å arbeide med programmering i Scratch på ulike klassetrinn og ulike fag (Moreno-León et al., 2016, s. 283). De ulike klassene var 2. og 6. klasse og fagene var matematikk og samfunnsfag. Resultatene var at læringskurven på 6. trinn økte, men i 2. klassene økte ikke kurven ved å programmere i Scratch (Moreno-León et al., 2016, s. 286). Når det kom til programmering i ulike fag, ble læringskurven dobbel så stor i samfunnsfag som i matematikk. Da skriver de at en grunn til dette kan være at det er mer kognitive krav til å jobbe med programmering i matematikken (Moreno-León et al., 2016, s. 287).

Flere forskere sikter til at programmering kan brukes til flere formål. For eksempel mener Husain et al. (2017, s. 1) at lærere imidlertid bør gjøres oppmerksomme på at undervisning i programmering ikke bare er beregnet til å kun tilegne seg ferdigheter, men det kan også øke forståelsen i andre grunnleggende fag som matematikk, språkfag og naturfag.

2.2 Ulike programmeringstyper

Det finnes ulike programmeringstyper, alle med ulike egenskaper, fordeler og ulemper i en undervisningssituasjon. Tre programmeringstyper er analog programmering, blokkprogrammering (for eksempel Scratch) og tekstprogrammering (for eksempel Python) (Mannila, 2017, s. 71-72). Tekstprogrammering er lite relevant for denne studien, da det finnes minimalt med oppgaver om tekstprogrammering i de tre analyserte læreverkene. Derfor vil det kun kort bli redegjort for før vi beveger oss over til blokkprogrammering og analog programmering. Eksempler på hvordan de kan brukes i matematikkoppgaver eller kobles til matematikk vil også bli presentert. I delkapittel 2.1 skrev vi om Papert's (1980) program Logo. Logo var et av de første programmene som ble laget for barn. I Logo-samfunnet er det en Skilpadde som elevene ved å gi kommandoer ved å programmere, ser resultatet av programmeringen i bevegelsene Skilpadden utfører (Papert, 1980, s. 11). Elevene kontrollerer da skilpadden ved å taste inn kommandoer til datamaskinen. Om det tastes inn «fram 100» vil skilpadden gå i en rett linje framover. Skriver eleven inn at den skal snu seg 90 grader, snur den seg 90 grader (Papert, 1980, s. 12). Etter Logo har programmeringsfeltet utvidet seg, og det finnes flere ulike programmer å arbeide i.

2.2.1 Tekstprogrammering

Tekstprogrammering er ifølge Mannila (2017, s. 71) et programmeringsspråk som baserer seg på å gi kommandoer ved hjelp av tekst. Hun legger til at enkelte ord og symboler har en forutbestemt betydning som kan settes sammen til å skape en funksjon. Eksempler på programmer som benytter seg av tekst som programmeringsspråk er for eksempel Python, Java og C++. De nevnte programmene er bare noen av mange, men det kan tyde på at majoriteten av forskningsprosjekter på høyrere trinn omhandler Python. For eksempel Stigberg og Stigbergs (2020, s. 490) forskning på 9. trinn viser at de bruker Python, men på 6. trinn fikk elevene jobbe med Scratch. Tekstprogrammering krever presisjon og at den som programmerer er nøye da det kan oppstå feil i koden ved skrivefeil, som kan være vanskelig å

oppdage blant all teksten (Mannila, 2017, s. 71). Ved dette blir en av fordelene ved å bruke blokkprogrammering synlig, nemlig at det ikke går an å sette blokkene sammen på feil måte.

2.2.2 Blokkprogrammering

Blokkprogrammering er det mest kjente verktøyet som brukes innen programmering (Lv et al., 2022, s. 14). Et blokkbasert program består av ulike biter som kan settes sammen for å skape noe (Mannila, 2017, s. 71). Bitene har ulike funksjoner og når de settes sammen skaper de en instruksjon. De ulike funksjonene har ulike farger og former for å symbolisere og visualisere hvilke blokker som kan settes sammen (Mannila, 2017, s. 71). Elevenes læringsoppnåelser, lærelyst og problemløsningsferdigheter har gjennom visuelle kodemiljøer forbedret seg ifølge en stor del av studier (Lv et al., 2022, s. 14). I samme artikkel fremkommer det som et funn at å arbeide med å innlemme matematikk og algoritmisk tenking i blokkprogrammeringsmiljøer er effektivt (Lv et al., 2022, s. 14).

Et blokkbasert programmeringsspråk vi har observert at blir brukt gjennom tidligere erfaringer i undervisning, er Scratch. Lee (2011, s. 27) trekker frem at dette er et dataprogrammeringsmiljø som retter seg mot yngre barn og utdanning. Videre omtaler han Scratch som et puslespill-lignende program hvor brukeren av programmet enkelt kan sette sammen visuelle blokker for å lage et multimodalt produkt (Lee, 2011, s. 27). Dette viser til at det blir brukt en datamus til å sette sammen en kode istedenfor å taste inn koden med tastaturet på dataen. Videre legger Lee (2011, s. 27) til at det ikke vil bli syntaksfeil i programmet fordi blokkene som passer sammen er visuelt merket og det er umulig å sette sammen blokker som ikke vil gi noe resultat. Sett i sammenheng med påstanden til Kjällander et al. (2021, s. 3) om at det virker som om yngre barn synes det er lettere å lære programmering ved å bruke visuelle eller fysiske ressurser, virker det hensiktsmessig å bruke Scratch i skolen. En annen styrke Scratch har er at det er et visuelt program, som er designet for å utvikle multimodale produkter (Lee, 2011, s. 27). Algoritmisk tenkning tar for seg ulike komponenter, og noen av disse trekker Zhang og Nouri (2019, s. 18) frem som vanskelig å utvikle gjennom Scratch. Hvor vanskelig det er avhenger av nivået av kognitiv utvikling (Zhang og Nouri, 2019, s. 18). Dette stemmer overens med det Lee et al. (2016, s. 187) tekker frem om at elevenes forståelse for programmering øker sammen med alderen til elevene.

2.2.3. Analog programmering

Analog programmering definerer Berg (2021, s 42) som programmering som blir gjennomført uten bruk av elektroniske verktøy. Dette samsvarer med Campbell og Walsh's (2017, sitert i Lee og Junoh's, 2019, s. 710) definisjon, men de vektlegger videre viktigheten av å bruke konkrete representasjoner som barn kan koble til sin hverdag ved innlæringen av programmering. Det innebærer praktiske oppgaver utenom datamaskinen hvor elevene får arbeide med å fysisk flytte på ting uten å måtte forholde seg til abstrakte arbeidsmetoder som ofte kan oppleves som vanskelig å forstå, i motsetning til å ha noe håndfast å se på og holde i (Campbell & Walsh, 2017, sitert i Lee og Junoh's, 2019, s. 710). Videre påpeker Lee & Junoh (2019, s. 710) de at barn koder mye analogt i hverdagen sin, uten at de selv og de voksne rundt dem oppdager det, for eksempel når de knyter/lærer å knyte skolissene sine (Lee & Junoh, 2019, s. 710). I artikkelen til Berg (2021, s. 51) fremkommer det at elevenes motivasjon og forståelse for hva de ulike løkkene de arbeidet med økte ved gjennomføring av analog programmering. Det kan ses i sammenheng med Lee og Junoh's (2019, s. 710) påstand om hvor viktig det er at læreren fanger opp disse hverdagslige aktivitetene/gjøremålene som barn skal lære. Å koble disse hverdagslige aktivitetene til en steg-for-steg-metode, der det er mulig, kan ses på som analog koding (Lee & Junoh's, 2019, s. 710). Analog programmering er dog ikke heldekkende alene. Av artikkelen til Berg (2021, s. 51) fremkommer det at læreren kan treffe en større del av elevgruppen ved en kombinasjon av å anvende analog og digital programmering grunnet elevenes ulike forutsetninger.

2.3 Algoritmisk tenkning

Gadanidis et al. (2017, s.77) omtaler Jeanette Wing som en med sterk påvirkning for betydningen av begrepet algoritmisk tenking i grunnskolen. Algoritmisk tenking innebærer i henhold til Wing (2006, s. 33) å løse ulike problemer, skape systemer og forstå menneskelig atferd ut ifra de grunnleggende konseptene innen IT-forskning. Det handler om å omformulere et komplisert problem til et problem som er mulig å forstå. Å kunne vurdere et verktøy ut ifra både hvor korrekt det er og hvor effektivt eller enkelt det er, er en viktig del av algoritmisk tenking. Kompetanse til å kunne ta valg ut ifra hvilke modeller og design som er hensiktsmessig å bruke til å fremstille deler av, eller hele problemet ses på som viktig å ha ifølge Wing (2006, s. 33). Hun skriver at når man skal løse et problem effektivt, bør man spørre seg selv om den tilnærmede løsningen er god nok, om vi kan anvende tilfeldig

utvelging til vår fordel (Wing, 2006, s 33). Hun skriver også at algoritmisk tenkning ikke bare er det å vurdere om programmet er korrekt og effektivt, men også se på om det er tiltalende (Wing, 2006, s. 33). Videre i denne artikkelen setter Wing (2006) fokus på fire komponenter når det kommer til algoritmisk tekning. To av komponentene hun legger frem er: abstraksjon og dekomponering, som hun skriver blir anvendt når man begynner med en stor kompleks oppgave (Wing, 2006, s. 33). Den tredje komponenten er algoritme, og den fjerde er det å være på jakt etter mønstre i en stor mengde sekvensdata (Wing, 2006, s. 34).

I studien til Lv et al. (2022, s. 1) ser de algoritmisk tenking som et trendy tema innen undervisningen i matematikken. Videre skriver de at forholdet mellom matematikk og algoritmisk tenking ikke er klart nok når det kommer til tilnærminger og resultatene (Lv et al., 2022, s. 1). For å innhente data i denne forskningsartikkelen så de gjennom høyt kvalifiserte skrevne tekster (Lv et al., 2022, s. 3). Ved å gjøre dette endte de opp med å analysere 22 studier, der de hadde noen kriterier, som for eksempel matematisk innhold. Et resultat de fant innebar at det var mest forsket på integreringen av algoritmisk tenkning og matematikk i grunnskolen, deretter kommer ungdomskolen (Lv et al., 2022, s. 6). De så også at det var forsket mer på temaet geometri når det kom til forholdet mellom algoritmisk tenkning og matematikk, men det var også forsket på blant annet tall og algebra (Lv et al., 2022, s. 8).

Da de så på integrering av algoritmisk tenkning i oppgavene som var forsket på, ble abstraksjon og mønstergjenkjenning hyppigest nevnt (Lv et al., 2022, s.11). Etter dette ble dekomponering, feilsøking og til slutt algoritme nevnt. De skriver at problemløsningsferdigheter i matematikk basert på algoritmisk tenking antydes det at hovedsakelig er konsentrert rundt disse fem aspektene (Lv et al., 2022, s. 12). Dette kan samsvare med det som presenteres i kapittel 2.6 Rammeverk, om konseptene som hører til i algoritmisk tenkings modellen til Barefoot Computing (u.å.-a). Videre i artikkelen foreslår de at ruteark kan være et nyttig verktøy i integreringen av algoritmisk tenking i matematikkundervisningen. Rutearket kan til en viss grad være med på å legge til rette for elevens matematiske læring (Lv et al., 2022, s. 14). Det helhetlige resultatet for denne forskningen viser til at resultatene innen elevenes læring er positive. Dette inkluderer like mye av elevenes engasjement og motivasjon, som deres forståelse av matematisk kunnskap, mestring av matematiske ferdigheter og utarbeidelsen av algoritmisk tenking. Likevel viser Lv et al. (2022, s. 18) til at flere forskningsartikler peker på at læringsutbyttet kan bli begrenset av elevens kognitive nivå og tidligere informasjon.

De ovennevnte definisjonene på algoritmisk tekning kan ses i sammenheng med Lodi (2020, s. 128) funn, der han så på sammenhenger mellom forskeres definisjoner på algoritmisk tenking. Han så at det var mye ulikheter, men det var enighet om at definisjonene på algoritmisk tenking var en tankeprosess, når det gjelder en spesifikk problemløsning. Denne spesifikke problemløsningen er avhengig av en utenforstående aktør, som kan automatisere gjennomføringen av oppgaven (Lodi, 2020, s. 128). Ut ifra funnene i denne artikkelen skapte han et forslag på en bred kategorisering av de ulike elementene i algoritmisk tekning, som Lodi og Martini (2021, s. 896) har tolket og utdypet. Det er deres tolkning vi vil benytte oss av videre i teksten. Kategoriseringen er i fire deler, som er deretter delt opp i underkategorier. Det er kun to av disse kategoriene som vi anser som relevant for denne problemstillingen og det er dermed kun de som blir utdypet.

> *Tankeprosess:* en tankestrategi som er hjelpsom under problemløsning (algoritme, logikk, dekomponering, modellering, abstraksjons og mønstergjenkjenning) (Lodi & Martini, 2021, s. 896).

> *Tverrfaglige egenskaper:* generelle perspektiver på å fungere i dagens samfunn som kommer frem ved å tenke som en IT-forsker. Kan også forklares som ferdigheter som er nyttige i livet generelt og for å forbedre egenskapen til å tenke som en IT-forsker. (Designe og skape, kommunisere og samarbeide, reflektere, lære) (Lodi & Martini, 2021, s. 896).

2.4 Programmering og algoritmisk tenkning

For å finne ut av hvordan programmering kan legge til rette for arbeid med algoritmisk tenking er det naturlig å finne tidligere forskning på det området. Programmeringsferdigheter og problemløsning omtaler Bai et al. (2021, s. 3) som tett relatert til algoritmisk tenking. For å utvikle elevenes ferdigheter innenfor algoritmisk tenking skriver Bai et al. (2021, s. 1) at programmering som undervisningsform er hensiktsmessig. Samtidig peker Barcelos et al. (2018, s. 832-833) på at matematikk har en betydelig fordel i forhold til tilgang til, og variasjon i didaktiske oppgaver og aktiviteter som kan kobles til algoritmisk tenking.

Lv et al. (2022, s. 14) viser til resultater som indikerer at å integrere algoritmisk tenking gjennom bruk av blokkprogrammering er effektivt. Malik et al. (2019, s. 87) argumenterer for at programmering kan legge til rette for algoritmisk tenking ved å ha gjennomført et forskningsprosjekt med en ny metode for å undervise i programmering. Denne nye metoden la vekt på at elevene skulle finne ut hvilken fremgangsmåte som er best for å løse problemet, som for eksempel algoritmisk tenking. Å inkludere algoritmisk tenking viste seg å forbedre elevenes tillit og kunnskap til programmeringsdomenet. Malik et al. (2019 s. 91) fant videre ut at å sette søkelys på fremgangsmåte førte til at de forstod programmering i en bredere kontekst som igjen kan hjelpe dem med å utvikle programmeringsferdighetene sine. Å bruke algoritmisk tenking for å utvikle en løsning, hjalp elevene på andre områder også. For eksempel hjalp det med å utvikle deres logiske tenking på en måte som er strukturert. I tillegg hjalp det studentene med å nå sine mål i kurset (Malik et al., 2019, s. 91).

I en annen studie ble det sett på om analoge programmeringsoppgaver kunne få elever til å forbedre sine ferdigheter innen algoritmisk tekning (Sun et al. 2021, s. 13). Denne studien fant ut at elevers algoritmiske tenking ble bedre i form av at elevene ble flinkere til å finne den enkleste måten å løse et problem i hverdagen. Elevenes logiske tenkning ble også drastisk forbedret av å jobbe med programmeringsoppgaver. De fant videre ut at elever som har god programmeringserfaring også har bedre forståelse innen algoritmisk tekning (Sun et al. 2021, 11-12). Dette kan ses opp mot det Bocconi et al. (2016, s. 48) fant i sin studie, der det kan se ut til at det er hensiktsmessig å starte med algoritmisk tekning tidlig i skoleløpet til elevene.

Når elevene skal jobbe med programmeringsoppgaver krever det matematisk tenkning, problemløsning og logisk tenking (Husain et al., 2017, s. 1). Vi nevnte i delkapittel 1.3 at algoritmisk tenking er en problemløsningsmetode. I en forskningsartikkel har Husain et al. (2017, s. 1) undersøkt metodene for å lære grunnleggende matematiske konsepter og effekten av å lære Scratch ved å anvende barneskoleelevers problemløsningsferdigheter. Det de fant ut var at elevene utviklet deres ferdigheter innen problemløsning og logisk tenkning ved å lage et program. Et annet positivt poeng var at de fant ut at elevene synes det var enkelt å lære seg å bruke Scratch, og viste interesse for å jobbe videre for å utvikle ferdighetene deres i programmering (Husain et al., 2017, s. 1).

I en artikkel skrevet av Bråting og Kilhamn (2022, s. 594) ser de på hvordan programmering blir integrert i matematikken. Dette gjorde de ved å kategorisere programmeringsoppgaver ut

ifra hvilke handlinger elevene måtte gjennomføre. I enkelte av oppgavene fant de spor av ferdigheter man bruker innen algoritmisk tenking. Eksempler på deler de fant var feilsøking, lage algoritme og det å skape noe (Bråting & Kilhamn, 2022, s. 599). I denne studien gjorde de også funn innen plassering av programmeringsoppgaver. Det er forskjell på hvor læreverk velger å plassere programmeringsinnholdet sitt. Et eksempel fra Sverige viser til at enkelte læreverk har plassert mesteparten av programmeringsoppgavene i nettressursen, andre har valgt å gjøre motsatt (Bråting & Kilhamn, 2022, s. 598). Studien fant også ut at det var forskjell mellom de ulike klassetrinnene i forhold til hvor innholdet blir plassert. Læreverk for 1. - 3. trinn hadde plassert sine programmeringsoppgaver i de trykte lærebøkene, imens fra 4. - 9. trinn ble flesteparten av oppgavene plassert i den digitale ressursen (Bråting & Kilhamn, 2022, s. 598). Ved å se nærmere på lærebøkene kom de frem til at det er typisk for oppgaver i matematikkbøker å formulere seg på en slik måte som kan være en type «steg for steg» instruksjon i oppgaveteksten (Bråting & Kilhamn, 2022, s. 605).

2.5 Lærebok

En lærebok inneholder det samfunnet mener er viktig her og nå, og burde ikke betraktes som ei bok med diskuterbar informasjon (Skjelbred et al., 2017, s. 9). Mot slutten av 1900-tallet skjedde en endring hvor det ble mer av at læreverkene inneholdt flere komponenter enn kun en bok (Skjelbred et al., 2017, s. 24). Pepin et al (2013, s. 929) skriver at lærebøker før bare var boken, men at det både har endret seg hvordan vi får og bruker lærebøkene nå. I dag ser lærere på læreboken i samhandling med ulike ressurser som hører til, som for eksempel digitale ressurser (Pepin et al. 2013, s. 929). At læreverkene ble bestående av mer enn bare læreboken førte til at arbeidet ble mer tilrettelagt for individuelt arbeid (Skjelbred et al., 2017, s. 24). Lærebøkene er med på å påvirke hva slags oppgaver og eksempler elevene arbeider med, samt hvordan begreper innen matematikk fremstilles (Johansson, 2006, s. 24-25). Dette kan føre til at det oppstår en spenning mellom lærere som følger læreboken fra perm til perm og staten som forventer at undervisningen tar utgangspunkt i målene formulert i læreplanen (Johansson, 2006, s. 26).

I en undersøkelse av lærerens bruk av lærebøker svarte nesten alle lærerne at de bruker læreboken omtrent hver dag i sin undervisning (Van den Ham & Heinze, 2018, s. 136). I samme undersøkelse var det 86 lærere som besvarte spørsmålet om de synes de ble styrt av

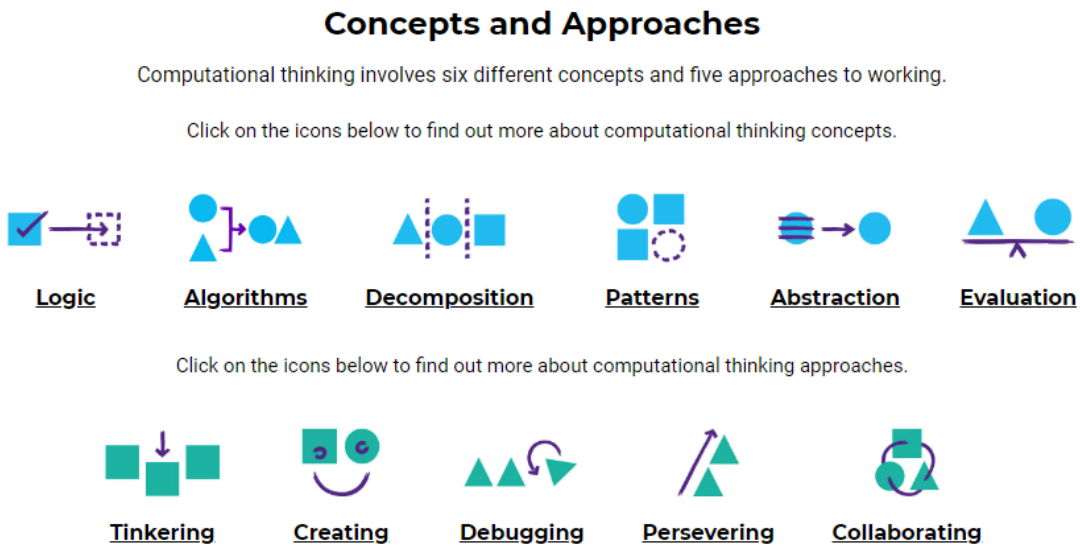
læreboken i forhold til blant annet metode og materiell. 75 av disse 86 lærerne sa seg enige i at undervisningen i stor grad blir styrt av læreboken (Van den Ham & Heinze, 2018, s. 136). Dette kan vise til hvor innflytelsesrike lærebøker kan være.

Ifølge TIMMS-undersøkelsen fra 2011 (Mullis et al., 2012, s. 391) er det matematikkboka som oftest blir brukt som hovedgrunnlag for matematikkundervisningen. Dette kan da ha mye å si for hva slags undervisningsoppgaver eleven møter. IDM (2007, sitert i Gracin & Krišto, 2022, s. 100) peker på at oppgaver i lærebøker kan variere i forhold til hvilket kognitivt nivå eleven blir utfordret til å arbeide med. Videre viser de til at enkelte oppgaver kan kreve at elevene henter frem konkrete grunnleggende ferdigheter og kunnskaper. Andre oppgaver er mer komplekse og krever at elevene blir nødt til å se sammenhenger og ha kunnskap om flere konsepter og regler for å kunne løse oppgaven. I sin studie kommer Gracin og Krišto (2022, s. 108) frem til at i alle klassetrinn er det en tydelig overvekt av oppgavene som er lite komplekse og krever mindre av elevene kognitivt.

I en studie oppdaget Brehmer et al. (2015, s. 589) at enkelte elever sannsynligvis aldri vil få arbeidet med problemløsning, som kan være et eksempel på kognitivt krevende oppgaver. Det at det vil bli forskjeller mellom elevene i forhold til hva slags oppgaver de får arbeide med grunnes at læremidlene ofte bygges opp slik at problemløsningsoppgavene ligger enten i slutten av kapittelet eller på de vanskeligere nivåene. Dette kan igjen føre til at elever som ikke rekker, eller får til å jobbe på de vanskeligere nivåene, ikke får arbeidet med problemløsning (Brehmer et al., 2015, s. 589). De argumenterer videre for at elevene som ikke får arbeidet med problemløsningsoppgaver ikke vil se nytten i sin matematiske kunnskap og sitte igjen med verktøy de i mindre grad vet hvordan de kan anvende for å løse andre problemer enn regnestykker (Brehmer et al., 2015, s. 589). Dette støttes av Malik et al. (2019, s. 85) som argumenterer for at gode valg av øvelser/aktiviteter samt læringsmaterialer spiller en viktig rolle i en suksessfull læringsprosess. Det kan se ut til at dette kan være med på å føre til forskjeller i utbyttet til elevene ut ifra hvilke lærebøker de får og hvilke valg læreren tar. Andre ting som kan være med på å føre til forskjeller er publiseringsvirksomheten. Johansson (2006, s. 29) påpeker at bak en lærebok står det en eller flere forfattere i tillegg til en produsent som mest sannsynlig ønsker å tilby et produkt som er godt laget og nøye utarbeidet pedagogisk versjon av læreplanen. Hun legger til at publiseringsvirksomhet i de fleste land er en bransje. Dette er tilfellet i Norge hvor det kan virke som om det ble åpnet opp for noe frihet for forfattere da godkjennelsesordningen ble fjernet i 2000 (Skjelbred et al., 2017, s.

18). Johansson (2006, s. 29) legger til at denne industrien fører til at utformingen og produksjonen av lærebøker drives både av økonomi og pedagogikk.

2.6 Rammeverk



Figur 1: Modell med OGL license (åpen lisens) fra Barefoot Computing (Barefoot Computing, u.å.-a)
https://www.barefootcomputing.org/concept-approaches/computational-thinking-concepts-and-approaches

I denne masteroppgaven har vi valgt å bruke modellen til Barefoot Computing (u.å.-a), som de kaller «The Computational Thinkers» for å kunne se på programmering i lys av algoritmisk tenking. Dette er en modell som tar for seg ulike komponenter av algoritmisk tenking. Disse komponentene er presentert her for at de senere i oppgaven benyttes for å skape en tabell som er inspirert av Barefoot Computing's (u.å.-a) modell. Dette gjøres for å gjennomføre vår vertikale analyse. Nettsiden hvor Barefoot Computing har publisert denne modellen, er utviklet av lærere og støttet av forskning (Barefoot Computing, u.å.-b). Videre på denne siden beskriver de at deres mål er å gi lærere tilgang til ressurser til undervisning. Utdanningsdirektoratet (2019, s. 2) anvendte Modellen «The Computational Thinker» da de har utarbeidet en plan for hvordan skolen skal jobbe med programmering og algoritmisk tenking. Vi har valgt å gå direkte til kilden, derfor vil vi anvende begrepene konsept og fremgangsmåte for de ulike komponentene i modellen, i stedet for nøkkelbegrep og arbeidsmåter som Utdanningsdirektoratet (2019, s. 2) anvender i sin oversettelse. *Figur 1* viser de ulike konseptene og fremgangsmåtene til Barefoot Computings (u.å.-a) modell for

algoritmisk tenking. Inne på nettsiden fungerer modellen slik at om det trykkes på de ulike bildene dukker definisjon til det gjeldende komponenten eller framgangsmåten opp. Deretter ligger det anvist et forslag til hvordan dette kan jobbes med i undervisningen. Vi vil redegjøre for deres forklaring på disse begrepene nedenfor.

2.6.1 Konsepter

Logikk er ifølge Barefoot-modellen (Barefoot Computing, u.å.-c) det som hjelper oss å beskrive hvorfor noe skjer. De skriver at logikk kan brukes for å finne ut hva et datasystem vil komme frem til. Ved å gi samme instruksjoner til to datamaskiner, vil du mer eller mindre få samme resultat. Dette er grunnet at datamaskiner ikke kan ta egne eksterne valg, og vil dermed ikke avvike fra instruksjonene underveis i prosessen. Derfor kan vi si at logisk resonering er at vi kan forklare hvorfor noe er slik det er (Barefoot Computing, u.å.-c). Videre skriver de at logikk er viktig, fordi det er det som kontrollerer alt en datamaskin gjør. Forskere bruker logikk under utvikling av programmer til datamaskiner. Logikk blir også anvendt når forskeren feilsøker i arbeidet sitt (Barefoot Computing, u.å.-c).

Algoritme er en gitt rekkefølge av instruksjoner som kan ses som mange regler som får noe gjort (Barefoot Computing, u.å.-d). En forklaring på dette kan være din favorittrute hjem fra skole eller jobb. Dette kan vi se på som en algoritme fordi det kan være flere artenative ruter hjem, og hver av disse rutene kan det lages en algoritme til. Det kan sies at en algoritme er forskjellig fra et program på bakgrunn av at den er skrevet slik at mennesker skal forstå, ikke en datamaskin (Barefoot Computing, u.å.-d). Videre påpekes det at algoritme er viktig, fordi man ønsker å finne en algoritme for å løse et problem på den mest hensiktsmessige måten, med et resultat som er presist og samtidig bruke få ressurser. Dette er noe forskere i dag strever med å få til (Barefoot Computing, u.å.-d).

Dekomponering er navnet på prosessen hvor et stort og komplekst problem blir delt opp i mindre og mer forståelige problemer (Barefoot Computing, u.å.-e). Å dekomponere et problem eller en oppgave har flere fordeler for eksempel at det hjelper til slik at det er mulig å håndtere større problemer uten å føle seg overveldet. En annen fordel er at flere kan jobbe med det samme problemet samtidig, men med hver sin del slik at hver enkelt får brukt sin kunnskap på det de er gode på (Barefoot Computing, u.å.-e). Andre grunner til at dekomponering er viktig, er at det er overførbart til hvilket som helst stort problem eller stor oppgave som skal gjennomføres. Det vil gjøre det enklere å løse dersom problemet eller

oppgaven er delt opp i mindre deler. Det har også overførbarhet til yrker i det virkelig liv hvor en og samme vare kan være produsert på flere steder (Barefoot Computing, u.å.-e).

Mønstre er med på å gi muligheten til å løse mer generelle problemer fordi de finnes overalt i samfunnet og ved å kunne identifisere mønstrene kan regler skapes for å løse problemene (Barefoot Computing, u.å.-f). Et eksempel på dette kan være formler i geometri i matematikken. Så fort elevene lærer seg formelen for hvordan regne ut areal vil de kunne løse problemet mye raskere enn da de telte ruter. Videre formidler de at mønstre er også viktig for å kunne finne løsninger som er mest mulig effektivt. Forskere leter etter gjentakende deler av en algoritme, altså et mønster, slik at de kan gjøre den så effektiv som mulig. Ved å finne disse mønstrene i algoritmene kan de også bruke dem til å løse andre problemer som ligner. Det å kunne kjenne igjen mønstre spiller også en viktig rolle i sammenheng med det å skulle lære seg å bruke ulike maskiner. Det siste som nevnes er at mønstre er en spesielt viktig del av IT-forskningen (Barefoot Computing, u.å.-f).

Abstraksjon er å forkorte noe, ved å identifisere det som er viktig og ikke la seg distrahere av unødvendig informasjon (Barefoot Computing, u.å.-g). Et eksempel på abstraksjon kan være familiekalender, der har man gjort abstraksjon med å bare ta med viktig nøkkelinformasjon. Nøkkelinformasjon i en familiekalender kan være en oversikt over fritidsaktiviteter og lignende. Men det som blir nøkkelinformasjon for deg er hvordan fritidsaktivitet du skal på. Da vil planene til de andre i familien bli uviktig informasjon for deg og ved å gjøre dette luker du bort unødvendig informasjon (Barefoot Computing, u.å.-g). I tillegg er abstraksjon viktig for å få oss til å tenke gjennom oppgaver i ulike grader av detaljer. Dette er en viktig egenskap under arbeid med IT- forskning, der det er vanlig å bruke abstraksjon til å håndtere designets kompleksitet (Barefoot Computing, u.å.-g).

Evaluering omhandler det å kunne ta avgjørelser på både en systematisk og en objektiv måte (Barefoot Computing, u.å.-h). Disse små og store avgjørelsene tas hver dag på bakgrunn av et bredt spekter av ulike faktorer. Evaluering er samtidig viktig for å kunne bedømme ulike produkters kvalitet, effektivitet, løsninger og systemer for å nevne noe. Andre egenskaper det er viktig å ha for er å kunne bedømme om for eksempel et program er passende og hensiktsmessig for målet (Barefoot Computing, u.å.-h).

2.6.2 Fremgangsmåter

I dette delkapittelet defineres de fem framgangsmåtene til Barefoot Computing (u.å.-a). *Fikling* handler om å teste ut noe ukjent for å finne ut av hva det er og hvordan det fungerer. Det er tett relatert til logisk resonnement (Barefoot Computing, u.å.-i). Gjennom å fikle bygger elevene seg opp ulike erfaringer i form av «hvis jeg trykker på denne, så skjer dette.» Dette bidrar til selvstendig læring, lek-basert eksperimentering og er en fase full av spørsmål, og overraskelser. Selv ideer elevene tenker kan være feil, får de prøvd ut kun for å se hva som skjer. For eldre elever kan fiklingsfasen handle mer om det å skape noe. Det hjelper til med å kunne se hvordan teknologi brukes og å tilegne seg en forståelse istedenfor å kun søke riktig svar. Ved å fikle med et digitalt verktøy først, vil elevene mest sannsynlig være mer åpne for nyskapende løsninger (Barefoot Computing, u.å.-i). Fikling er også viktig for å gi elevene selvtillit og en la-oss-prøve-innstilling ved at de har frihet til å utforske i et risikofritt miljø. Det åpner også for å kunne se ting fra flere ulike vinkler og fremhever kreativitet. Programmerere pleier vanligvis å utforske ny teknologi for å gjøre seg kjent med programmet slik at de kan få ideer til hvordan de kan utvikle det. IT-forskning utvikler seg fort og det er derfor viktig at elevene lærer å bli åpne for raske endringer og utviklinger. Å være selvsikker under fiklingsprosessen hjelper med å endre synsvinkel slik at ferdighetene kan holdes oppdatert og tilegne seg ny kunnskap om ny teknologi, samt å utvikle utholdenhet (Barefoot Computing, u.å.-i).

Skape omhandler å kunne planlegge og skape (Barefoot Computing, u.å.-j). Det må være rom til å kunne vise sin kreativitet uansett om du bruker ulike verktøy. Kreativitet kommer frem ved at ulike programvarer og digitale verktøy lager et rom for det. Når du mestrer de ulike verktøyene vil du dyrke kompetanse, selvtillit og kunne bruke det fritt, utforskede og lekent for å skape noe nytt. Programmering i seg selv er en kreativ prosess hvor vi har ideer for hva vi har lyst til å skape eller løse, analysere problemet, designe, feilsøke og til slutt evaluere (Barefoot Computing, u.å.-j). Andre grunner til at fremgangsmåten skape er viktig, er på bakgrunn av IT-forskning er ikke bare et akademisk emne, men det er også praktisk, der du skal skape et forslag til hvordan ekte verdensproblemer kan løses. Prosessen med å lage noe nytt er også en bra plass for læring (Barefoot Computing, u.å.-j).

Feilsøking handler om prosessen hvor algoritmen eller programmet blir sett gjennom etter feil (Barefoot Computing, u.å.-k). Disse feilene kan være så lite som en liten del av en algoritme og denne prosessen kan dermed ta enda lengre tid enn det tok å skape algoritmen i seg selv.

Det finnes hovedsakelig to forskjellige typer feil. Den ene er logisk feil og den andre er feil i syntaks. Disse feilene forklarer Barefoot Computing (u.å.-k). som den feilen hvor vendepunktet og slutten i en fortelling ikke gir mening. Syntaksfeil går mer på skrivefeil i fortellingen. Under feilsøking, for å gjøre prosessen enklere, vil det være nyttig å se variablene i algoritmen slik at det blir mulig å se hva som skjer imens programmet kjører. Barefoot Computing (u.å.-k). har foreslått fire steg som kan være hjelpsomt å følge under arbeid med feilsøking.

1. Forutsi hva som skal skje
2. Finn ut hva som skjer
3. Finn ut hvor noe har gått galt
4. Fiks det

Feilsøking er viktig fordi den kompliserte koden programmerere skaper fungerer ikke alltid slik det er tenkt at den skal. Det er rapportert at 50-75% av et prosjekts budsjett går med til aktiviteter knyttet til feilsøking og testing. Dette er dermed en viktig del av selve programmeringsprosessen (Barefoot Computing, u.å.-k).

Å holde ut i denne sammenhengen vil si å være bestemt, målrettet og aldri gi opp (Barefoot Computing, u.å.-l). Programmering er vanskelig og krever dermed mer enn kunnskap og kompetanse. Av programmereren kreves det en vilje til å fortsette å arbeide med problemet uansett hvor frustrerende det er. Dette er noe som kan oppleves som vanskelig i lengden. Likevel er det noe som gjelder for mer enn bare programmering. Ved utvikling av all slags kompetanse, for eksempel fotballferdigheter, sjakk, musikk eller dataspilling er det det å trene og øve om og om igjen som er nøkkelen til suksess (Barefoot Computing, u.å.-l). Under arbeid med programmering eller andre typer komplekse problemer vil det ofte være sannsynlig at det blir nødvendig å prøve flere ganger, prøve flere ulike alternativer og fremgangsmåter for å komme frem til svaret. Det kan bli nødvendig å endre tankemåten fra raskt og effektiv til en langsommere metode hvor logisk tenking trer tydeligere frem. Derfor argumenterer Barefoot Computing (u.å.-l) for at IT-forskere trenger gode egenskaper innen tålmodighet, utholdenhet, altså *holde ut*, innenfor arbeid med samme komplekse problem, eller det å kunne tåle forvirring.

Samarbeide er å jobbe i samhandling med andre for å få best mulig resultat (Barefoot Computing, u.å.-m). Det å jobbe i samhandling med andre kan være med på å motivere seg

selv til å prøve litt til. Oppgaver som en alene vil gi opp, kan en i samhandling med andre oppleve forståelse og mestring. IT- forskere henter mye inspirasjon fra hverandre. Ofte under arbeid i par i en programmeringsoppgave, er den ene den som skriver ned koden, mens den andre skal se helheten. Rollene vil rullere innad i gruppen, slik at en får erfaringer med alle arbeidsoppgavene (Barefoot Computing, u.å.-m). Samarbeide er også viktig fordi ulike programdesignere har ulike kompetanse, dermed vil en få et best mulig produkt ved å arbeide godt sammen (Barefoot Computing, u.å.-m).

Ut ifra det som nå har blitt redegjort for ovenfor har vi laget en tabell som er vedlagt under i kapittel 3.4.2. Den ene inneholder de ulike konseptene og hva spesifikt vi leter etter i en oppgave for hvert konsept. Den andre inneholder fremgangsmåtene og kjennetegn for dem i oppgavetekst. I den vertikale analysen kommer vi til å anvende disse tabellene for å finne ut hvilke komponenter av algoritmisk tenking det arbeides med under arbeid med programmeringsoppgaver. Dette vil bli videre utdypet i kapittel 3.4.2 Vertikal analyse.

3. Metode

I dette kapittelet redegjør vi for hvilken metode denne masteroppgaven baserer seg på for å kunne svare på problemstillingen. Vi vil gjøre dette ved å forklare kort hvorfor vi har valgt kvalitativ metode, samt hvilke datainnsamlingsmetoder som faller innenfor denne metode. Vi vil videre avklare valget av innholdsanalyse, som blir brukt for å innhente data om algoritmisk tenkning innen programmering og matematikk. Deretter presenteres lærebøkene, som er datamaterialet for innholdsanalysen. Videre kommer det en begrunnelse og forklaring på hvordan vi skal anvende rammeverket beskrevet i 2.6 Rammeverk. Og sist for å avrunde blir det skrevet om studiens tiltak for å styrke relabilitet og validitet i oppgaven, og forskningsetiske betraktninger knyttet til vår oppgave.

3.1 Kvalitativ metode

Ved valg av forskningsmetode er det viktig å ha et godt overblikk og være klar over styrker og svakheter i de ulike metodene for å finne den som gir best mulig svar på studiens problemstilling (Postholm & Jacobsen, 2018, s. 89). Dette delkapittelet vil derfor omhandle betraktninger omkring kvalitative forskningsmetoder.

Kvalitativ forskningsmetode er mer relevant for vår problemstilling fordi den åpner opp for muligheten til å gå i dybden av oppgaven. Dette kan man se ved at viktige begreper i en kvalitativ tekst er beskrivelse, forståelse og mening (Postholm & Jacobsen, 2018, s. 95). Det gir oss også muligheten til å beskrive ulike nyanser og fortolkninger, hvilket kommer godt med når målet med denne studien er å få en større forståelse for innholdet i programmeringsoppgaver på 6. trinn. Videre formidler Postholm og Jacobsen (2018, s. 89) at det eksisterer flere måter å fremstille denne typen informasjon, og at en av dem er gjennom forskerens egne observasjoner. På en annen side åpner ord opp for fortolkning og forskerens tidligere erfaringer og eventuelle fordommer kan påvirke resultatet (Postholm & Jacobsen, 2018, s. 92). Likevel vil kvalitativ metode være mest hensiktsmessig i denne oppgaven, på grunn av at det er våre betraktninger om innholdet i oppgaver i lærebøker på 6. trinn som er vektlagt. Derfor blir det viktig å tenke over dette under arbeidet med oppgaven. Innenfor kvalitative metoder finnes flere ulike metoder å samle inn data. Det kan være intervju, observasjon og tekstanalyse for å nevne noen (Postholm & Jacobsen, 2018). Tatt i betraktning at vårt datamateriale er læreboken, vil det være naturlig å bruke tekstanalyse som metode.

3.2 Tekstanalyse

Ifølge Bratberg (2021, s. 11) kan man se på tekst som datamateriale, og denne formen for forskningsmetode kalles for en tekstanalyse. For å innhente det rette datamaterialet til det gjeldene forskningsprosjektet kan man anvende ulike metoder for å analysere tekstene. Dette gjør vi ved å tolke og filtrere ut den informasjonen som en nyttig for å svare på problemstillingen. Gjennom bruk av tekster som datamaterialer vil det kunne drøfte frem for eksempel ulike synpunkter og slutninger som kommer frem i teksten (Bratberg, 2021, s. 13). Det er ulike metoder som er vanlig innenfor metodefamilien tekstanalyse (Bratberg, 2021, s. 12). Disse metodene er retorisk analyse, kvalitativ innholdsanalyse og diskursanalyse (Bratberg, 2021, s. 13-14). Forskningsspørsmålet styrer metoden, men justering og tilspisning av forskningsspørsmålet kan bli styrt av metoden som forskeren ser på som mest hensiktsmessig (Bratberg, 2021, s. 12). For å best mulig kunne besvare hvordan lærebøker i matematikk bruker programmering for å legge til rette for arbeid med algoritmisk tenking, er det naturlig å analysere lærebøkene. På bakgrunn av at vi vil se på kun enkelte oppgaver, altså kun programmeringsoppgavene i lærebøker, har vi valgt å bruke innholdsanalyse som metode.

3.2.1 Kvalitativ innholdsanalyse

Når det gjelder kvalitativ innholdsanalyse omtaler Clark et al. (2021, s. 516) det som mest sannsynlig det er den mest utbredte metoden når det gjelder analyse av dokumenter. Dokumenter i dette tilfellet betyr skrevne tekster som er offentliggjort slik at alle har tilgang til dem (Clark et al., 2021, s. 498). Lærebøker på 6. trinn i matematikk vil dermed falle under kategorien dokument. Metoden involverer å finne underliggende temaer i datamaterialet (Clark et al., 2021, s. 516). Dette kan overføres til at vi i denne studien forsker på hvordan matematikkbøker bruker programmeringsoppgaver for å legge til rette for arbeid med algoritmisk tenking. Dette forutsetter ikke at alle programmeringsoppgaver legger til rette for algoritmisk tenking, men vi ønsker å se etter om oppgaver som kan gjøre det. Å søke etter underliggende temaer medfører at det er naturlig for forskere som gjennomfører en kvalitativ innholdsanalyse å dele opp dokumentet sitt i ulike temaer (Clark et al., 2021, s. 516). For å kunne gjennomføre dette har vi valgt å bruke Charalambous et al. (2010) sin analysestrategi. Denne strategien og hvordan vi vil bruke den vil vi utdype i kapittel 3.4.

Ifølge Krippendorf (2019, s. 22) har kvalitativ innholdsanalyse røtter blant annet innen sosialvitenskapen, hvilket er med på å gjøre denne metoden passende for å besvare problemstillingen til denne oppgaven. Dette er fordi vårt forskningsområde omhandler det som foregår i samfunnet, altså undervisning om programmering på 6. trinn. Clark et al. (2021, s. 516) legger til at det da også er vanlig å vedlegge eksempler fra teksten til hvert av disse temaene.

3.3 Valg av lærebøker

Valget av lærebøker i norsk skole er noe skolene bestemmer selv, påvirket av ulike faktorer. Det er mange ulike matematiske læreverk å velge i. Som nevnt tidligere i kapittel 2.5 Lærebok, kan hvem som helst utgi læreverk i Norge. Dette er på bakgrunn av at myndighetene bestemte i 2000 at godkjenningsordningen opphørte (Skjelbred et al., 2013, s. 18). Dermed, da vi skulle velge lærebøker til vår studie undersøkte vi oss frem til de tre største forlagene i Norge som har gitt ut læreverk. Ut ifra det vi fant var Gyldendal, Aschehoug og Cappelen Damm de største forlagene i Norge som har gitt ut læreverk (Neraal, 2022). Deres læreverk heter Multi fra Gyldendal, Matemagisk fra Aschehoug og Matematikk fra Cappelen Damm. Det vil komme en forklaring på deres læreverk lenger ned i delkapittelet. Først kommer den avklaring for valg av årstrinn.

I denne masterstudien har vi som tidligere nevnt valgt å forske på lærebøker tilhørende 6. trinn. En av grunnene til dette er at elevene på dette tidspunktet i grunnskoleløpet, har tilegnet seg grunnleggende forkunnskaper om programmering. Det arbeides med programmering fra 1. trinn, bortsett fra at begrepene innen temaet ikke anvendes. Ved å se på læreplanen for matematikk for 5. trinn oppdaget vi at det er dette året begrepene innen programmering også begynner å tre inn i undervisningen. Kompetansemålet som lyder: «lage og programmere algoritmer med bruk av variabler, vilkår og løkker» (Kunnskapsdepartementet, 2019, s. 10) henviser til at elevene før 6. trinn skal ha arbeidet med og opparbeidet seg en viss forståelse for begrepene variabler, vilkår og løkker. Samtidig har elevene heller ikke kommet like langt i læringsprosessen som elever på 7. trinn hvor de skal begynne med programmering i tabeller (Kunnskapsdepartementet, 2019, s. 11). Sett i lys av at den nye læreplanen ble innført for tre år siden vil ikke sjetteklassinger i dag ha samme forkunnskaper som sjetteklassinger om noen

år. De som går på 6. trinn i dag begynte opplæring om programmering da de gikk på 3. trinn, om de begynte med en gang. Læreplanen kan ha en tendens til å bli gradvis innført som fører til at elevene på 6. trinn i dag og om noen år vil ha ulikt utgangspunkt. Det kan også antas at flere skoler har avendt læreboken fra LK06 samtidig som de ventet på at kommunen gikk til anskaffelse av lærebøkene tilhørende LK20. Dette kan være med på å gjøre at fokuset på programmering ikke har kommet inn ordentlig før noen år etter læreplanen ble innført, i påvente av oppdaterte læreverk til den nye læreplanen.

Til de fleste læreverk finnes det både trykte bøker, nettressurser og eventuelt ekstramateriale. I oppstartsfasen til arbeid med masteroppgaven, hadde vi en antakelse om at det ville være flere programmeringsoppgaver i de digitale ressursene. Likevel valgte vi å sette søkelys på kun trykte læreverk for å få et bilde av hvilke ressurser innen programmering og algoritmisk tenking som befinner seg i landets skoler, uavhengig av teknologisk utstyr som vil variere fra skole til skole.

3.3.1 Matemagisk

Opprinnelsen til Matemagisk stammer fra Sverige. Den første Matemagisk-boka er oversatt fra svensk og revidert slik at den passer til den norske læreplanen som da var LK06 (Bjerke et al., 2007). Læreboken Matemagisk er skrevet av Kongsnes, Raen og Sørdal (2021). Denne læreboken tilhører forlaget Aschehoug, som er Norges tredje største forlag (Neraal, 2022). På nettsiden trekker de frem egenskaper de selv mener boka har, og hva elevene får ut av å anvende den (Aschehoug, u.å.). Det første de nevner er at elevene får oppgaver der de er nødt til å snakke et matematisk språk. Sammen med den typen oppgave får de også andre typer som problemløsning og utforskende aktiviteter. De presenterer en struktur som de omtaler som tilpasset både klasseromsundervisning, individuelt arbeid og differensieringsmodell som lar elevene lære på eget nivå, men likevel parallelt med de andre. Avslutningsvis fremlegger de at fra 5. til 7. trinn tilbys det adaptiv læring på en digital plattform gjennom ØveMatematikk (Aschehoug, u.å.).

Matemagisk er en nivådelt bok som legger opp til at alle elevene skal gjennom de oppgavene som befinner seg i kategorien *følg stien.* Oppgavene er godt merket ved at det både står *følg stien* i en rød boks på siden, samtidig som at hele siden er rødfarget. Boka forklarer selv disse oppgavene som øvingsoppgaver der elevene får trent mer på det de arbeidet med i fellesskap. Det trenes også på kun en ting om gangen. Neste nivå kalles for *terrengløypa*. Dette er

oppgaver som er noe mer utfordrende og er merket med gul farge på samme måte som *følg stien.* Selv beskriver boka disse oppgavene som at de bygger videre på det de arbeidet med i fellesskap og at det er sammensatte utfordringer fra flere temaer. Det mest utfordrende nivået kaller de for *topptur.* Fargekoden for topptur er blå. Oppgavene er godt markert på samme måte som de andre, med skilt og sidefarge i blått. Dette er de mest utfordrende oppgavene boka inneholder. Det anbefales ikke å gjøre disse oppgavene med mindre eleven mestrer oppgavene i terrengløypa særdeles godt. Boka omtaler oppgavene på nivået *topptur* som svært utfordrende.

Foruten om nivådeling har Matemagisk også ulike typer oppgaver som er markert i boka. Den ene typen er *spill.* Spillene i boka er rammet inn av en striplet rød linje. Øverst på siden kan man se symbolet for at det er et spill som er to røde terninger med hvite øyne. En annen type oppgave boka inneholder er *aktivitet*. Disse oppgavene er markert på samme måte som spillene, med rød striplet linje. Symbolet for *aktivitet* er en rød legokloss. Matemagisk tilbyr også oppgaver som kan gjøres på *kopiark*. Disse arkene er tilgjengelig for læreren på Aunivers.no og kan hentes og printes ut etter ønske. På sidene i boka er kopiark oppgavene markert med et symbol som er en printer tegnet i svart. En fjerde type oppgave som befinner seg i boka er *nøkkeloppgaver*. Dette er oppgaver som ifølge boka selv viser spesielt viktige ideer og tenkemåter. I boka er *nøkkeloppgavene* markert med et nøkkelhullsymbol i gult og hvitt. Den siste typen oppgaver som er beskrevet i boka er *snakke matte*. Dette er oppgaver som er beregnet til at elevene skal samtale med hverandre om matematikk. I boka til elevene er denne typen oppgaver markert ved at de er i et blått felt med *snakke matte* til overskrift.

3.3.2 Multi

Ifølge Neraal (2022) er Gyldendal Norges nest største forlag. De utgir også læreverk for flere fag på flere trinn. Læreverket for matematikk på grunnskolen heter Multi, som er skrevet av Alseth, Røsseland, Arnås og Nordberg (2021). På Gyldendal sine nettsider er det publisert et intervju med forfatterne av Multi (Gyldendal, 2021). Der presenterer de grunner til hvorfor de mener skolene skal velge å bruke Multi som læreverk. Det første de nevner er at Multi er et komplett læreverk i matematikk for hele barnetrinnet som har blitt revidert og oppdatert etter LK20 kom. De tilføyer at læreverket inneholder en kombinasjon av digitale og fysiske læremidler. Andre grunner de trekker frem som grunnlag for at skolene skal velge Multi er at læreverket inneholder ulike arbeidsmåter, for eksempel utforsking, samarbeid og lek. De meddeler også at oppgavene er varierte og legger til rette for dybdelæring gjennom aktiviteter,

mengdetrening og elevtilpasset øving. Til slutt viser de til grundig og innholdsrik lærerveiledning sammen med et bredt utvalg av bøker og digitale ressurser som er med på å skape fleksibilitet og variasjon (Gyldendal, 2021).

I Multi har de utformet de ulike oppgavetypene ulikt, noe de sier er for at elevene og læreren skal vite at det er ulike forventinger på de gitte oppgavetypene (Gyldendal, 2021). I starten av hvert kapittel har de et *samtalebilde.* Dette bildet har vi valgt å ikke ta med i vår analyse av oppgavene. Det er fordi vi har satt kriterier for at en oppgave skal være en oppgave, må den ha en gitt aktivitet i form av tekst der eleven skal gjennomføre noe. Dette har ikke samtalebildene i Multi, men det kan hende det hadde vært annerledes om vi hadde analysert lærerveiledningen i tillegg til læreboken. En annen oppgavetype de har er grønne *aktivitetsruter*, som er merket med en stor A i venstre hjørne. *Aktivitetsrutene* skal ofte gjøre at elevene må jobbe utenfor elevboka (Gyldendal, 2021). Videre har de en oppgavetype som de kaller U. Dette er en brun *utforskningsrute,* som tar opp temaer eleven ikke har gjennomgått før. Etter utforskningsoppgavene kommer en forklaringsrute. Den har vi ikke valgt å se på som en oppgave, men heller informasjon til elevene. Boka har også røde ruter som omhandler *spill*. Nest sist i kapittelet har den en del som de kaller Kan du dette? Der får eleven se hva de kan etter å ha jobbet med kapittelet. Helt til slutt i kapitlet har de noen sider, som de kaller *øvesider*. Disse skal være med på å gi eleven bedre dybdeforståelse i temaene (Gyldendal, 2021).

Vi har også tatt kontakt med alle forlagene der vi har stilt spørsmål angående utviklingen av programmeringsoppgavene i læreverkene deres tilhørende 6. trinn. En representant fra Gyldendal (personlig kommunikasjon, 6. mars 2023) skrev i sin e-post at:

> *I arbeidet med programmering i Multi har vi brukt fagkonsulenter, dvs. både interne og eksterne programmerere og utviklere, som har jobbet med det faglige mht. programmeringen. Det didaktiske har forfatterne og lærere sett på. Så her har det altså vært et tett samarbeid mellom de ulike fagmiljøene, slik at både programmering, didaktikk og nivå skal være ivaretatt. I tillegg er det designere som ser på oppbygging og det visuelle uttrykket, slik at brukeropplevelsen skal bli best mulig.*

3.3.3 Matematikk fra Cappelen Damm

Norges største forlag (Neraal, 2022) er Cappelen Damm. De har produsert læreboken Matematikk fra Cappelen Damm, som er skrevet av Gulbrandsen, Løchsen, Måleng og Olsen (2020). På Forlagets nettside har de skrevet at Matematikk er en oppdatert og revidert versjon av den gamle læreboken Radius (Cappelen Damm, u.å.). Videre skriver de at det som kjennetegner deres læreverk er at den inneholder varierende undervisningopplegg som støtter dybdelæring. Og kravene de legger på elevene er at de skal tenke selv og sammen, snakke matematikk, utfordre seg selv og teste ut det de gjør med mer. Når det kommer til grunnboken i læreverket skriver de at de har en bok tilhørende hvert trinn (Cappelen Damm, u.å.). De starter deres kapitler med å illustrere en fortelling, som presenterer et matematisk problem. Kapitlene deres avsluttes med en oppsummering, som åpner opp for refleksjon og problemløsning (Cappelen Damm, u.å.).

I Matematikk fra Cappelen Damm finnes det ulike oppgavetyper. Den ene typen er *samtale.* Der de skriver at samtalerutene har en problemstilling, som skal diskuteres og finne mulige løsninger (Gulbrandsen et al. 2020, s. 2). Og etter dette kommer boka med et eksempel på en løsning eller to. Vi har her valg å ikke se på dette som en oppgave, grunnet at det blir for mye informasjon om hvordan dette kan gjennomføres på siden. Men vi har tatt med de samtalene som ikke har med mulige løsninger. Boken har også *oppgaver* som en oppgavetype. Der denne oppgavetypen starter likt som det eleven har jobbet med i samtaleoppgavene, men etter dette har oppgavene varierende innhold og vanskelighetsgrad (Gulbrandsen et al. 2020, s. 2). Deretter har de en oppgavetype de kaller *utforsk sammen*, som omhandler at elevene skal jobbe sammen, der de må reflektere og diskutere i en samtale for å løse problemet. Dette skal gjøre at elevene blir bedre på å snakke matematisk, ta imot og dele deres kunnskap. Etter dette har denne læreboken skrevet at de har en oppgavetype som heter *oppgaver med digitale verktøy*. Målene med disse oppgavene er å bli kjent med digitale verktøy som Excel og GeoGebra (Gulbrandsen et al., 2020, s. 3). Derimot kan vi se at flere digitale oppgaver er i eksemplene, disse har vi ikke talt med som oppgaver, grunnet at eksempler ikke blitt talt som en oppgave i de andre lærebøkene heller. Grunnen til at vi ikke har valgt å telle et eksempel er at vi ser på et eksempel som en type forklaring. *Temaoppgaver* har vi ikke talt for seg selv, fordi vi ikke så noen tydelige skiller mellom dem og *oppgaver.* Disse oppgavene gjør at elevene får bruke kunnskapen sin på tvers av temaer. En annen oppgavetype boken nevner er *sant eller usant?* Her skal elevene ta stilling til om en påstand er sannhet eller løgn.

Under arbeid med oppgavetypen *oppsummerende oppgave* får du prøve ut hvordan kunnskap du sitter igjen med etter endt kapittel. Til slutt har du oppgavetypen *Spill*. Der boka sier at elevene får jobbe med matematikken på en kreativ måte (Gulbrandsen et al., 2020, s. 3).

Som med de andre forlagene har vi spurt om hva slags kompetanse de som har laget programmeringsinnholdet i bøkene har. Fra Cappelen Damm fikk vi dette som svar i en e-post av en representant fra forlaget (personlig kommunikasjon, 21. mars 2023):

> *"Vi har hentet inn eksterne til å lage undervisningsopplegg i programmering, og at vi har lagt det i den digitale lærerressursen for at det skal være enkelt å oppdatere innholdet etter hvert som elevene vil lære mer om programmering på småskoletrinnet. De elevene som går i 6. nå, har jo hatt gammel læreplan mesteparten av småskoletrinnet."*

3.4 Bruk av rammeverk

Horisontal og vertikal analyse er en sammensatt analyse der dataen først blir analysert med et perspektiv med overblikk (horisontalt) og deretter et dybdeperspektiv (vertikalt) (Charalambous et al., 2010, s. 122). Det er en tredje analysestrategi de kaller kontekstuell analyse, den tar for seg hvordan det som forskes på gjøres av enten lærere eller elever i undervisningen (Charalambous et al., 2010, s.120). På grunn av oppgavens tidsbegrensing, vil vi ikke kunne observere ute i felt, og denne strategien vil derfor ikke kunne brukes i denne masteroppgaven. Derimot vil studien dra fordeler av å analysere både horisontalt og vertikalt fordi hver av perspektivene har kvaliteter som utfyller hverandre (Charalambous et al., 2010, s. 122). Det var en av grunnene til at vi valgte Charalambous et al. (2010) analytiske strategi for vår studie. Den horisontale analysestrategien tar for seg å kategorisere oppgavene etter gitte kriterier. Da vi ikke fant et rammeverk som inneholdt de spesifikke kriteriene vi har valgt å se på i denne masteroppgaven, har vi utviklet en egen tabell med ord som en programmeringsoppgave må inneholde, for at vi i denne masteroppgaven skal kategorisere det som en programmeringsoppgave. I den vertikale analysen vil vi benytte oss av Barefoot Computing's (u.å.-a) modell på algoritmisk tenking, som vist i kapittel 2.6 Rammeverk, hvor vi i denne masteroppgaven har brukt de samme kategoriene for å utvikle et rammeverk. Charalambous et al. (2010, s. 122) vektlegger også at perspektivene henter frem informasjon om ulike deler av læreboken, som for oss kan være interessante å se nærmere på.

3.4.1. Horisontal analyse

Den horisontale analysen dreier seg om å skape en oversikt over lærebokens struktur (Charalambous et al., 2010, s. 122). Strukturer de trekker frem som eksempler kan være størrelsen på sidene, antall sider, hvilke emner boka inneholder og i hvilken rekkefølge de er fremstilt. Videre forklarer Charalambous et al. (2010, s. 122) at de skapte to kategorier innenfor det horisontale perspektivet på analysen. Den ene kategorien kalte de *bakgrunnsinformasjon* og den andre kategorien fikk navnet *overordnet struktur*. For å utdype vil *bakgrunnsinformasjon* i denne masteren dreie seg om et beskrivende overblikk av boka, og grunner til at den ble laget. Det vil også være naturlig å se på bokas oppbygning, dermed kategorien *overordnet struktur.* Charalambous et al. (2010, s. 122) beskriver innholdet i denne kategorien som emnene i læreboken og hvordan boka generelt er organisert og strukturert. For denne studien vil det innebære ulike emner, typer oppgaver og hvordan de er strukturert og bokas generelle oppbygning. Dette gjør vi fordi det kan være med på å øke forståelsen for hva forfatterne har tenkt angående anvendelse av boka. Informasjonen som hentes ut gjennom denne formen for analyse er likevel lite detaljert og gir lite informasjon om hva som står i boka (Charalambous et al., 2010, s. 122). Tidligere i oppgaven har vi beskrevet overordnet hvilke typer oppgaver bøkene inneholder og litt om deres oppbygning. I analysekapittelet vil det presenteres utdypende informasjon om hvert av kapitlene.

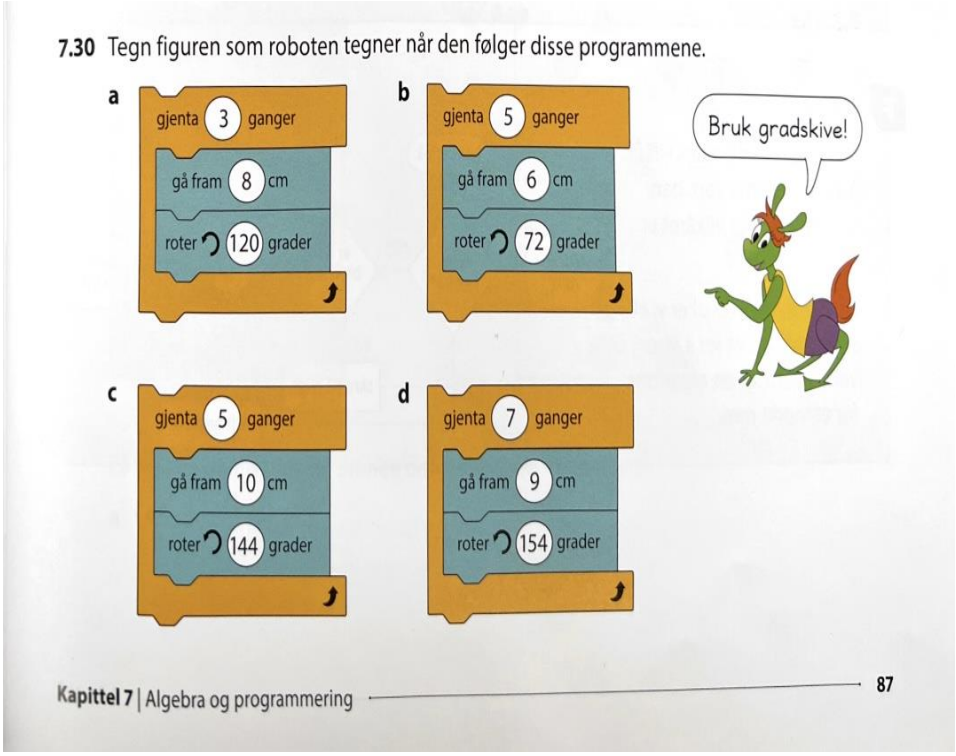
Når det gjelder bakgrunnsinformasjon om bøkene i denne studien er det relevant å se på blant annet sideantall, antall oppgaver totalt, samt antall programmeringsoppgaver. Dette gjøres ved at vi teller opp hvor mange oppgaver som krever programmering av elevene og hvor mange som ikke krever det. For å kunne si om en oppgave er en programmeringsoppgave eller ikke, satt vi som nevnt tidligere i kapittel 1.2 Problemstilling og avgrensninger, et krav til hva oppgaven måtte inneholde. Det andre kravet vi satt til oppgavene er at de i beskrivelsen er nødt til å inneholde begreper som omhandler programmering. Disse begrepene er listet opp i *tabell 1* vist under. Dette valget har vi tatt på bakgrunn av det Stigberg og Stigberg (2020, s. 491-492) påpeker om at elever ikke alltid klarer å se koblingene mellom matematikk og programmering, Ved å sette en begrensning ved hjelp av programmeringsbegrepene kan det føre til at både elev og lærer er klar over at det er programmering de holder på med. Vi ønsket også at det skulle være tydelig slik at det vil være lett å etterprøve og få samme utfall som oss. En av programmeringsoppgavene som er analysert i denne oppgaven inneholder ikke begreper i selve oppgaveteksten. Oppgaven er likevel regnet som en programmeringsoppgave

fordi overskriften på siden er koding, hvilket gjør det tydelig at det er koding de skal arbeide med. I tillegg omhandler oppgaveteksten en robot som også kan assosieres til programmering. Dermed er robot tatt med som et begrep i tabellen med begreper om programmering. Et siste krav til programmeringsoppgavene var at de må kreve en håndfast handling av elevene. Under arbeidet med oppgavene må elevene gjøre noe for å løse den. For eksempel skrive noe for hånd, eller tegne noe.

Begreper	Variasjoner
Kode	koden/koding
Programmering	programmer/programmert
Programmet	programmene
Blokkprogrammet	kodeblokkene/blokker
Algoritme	
Løkke	
Vilkår	
Funksjon	
Variabel	
Robot	

Tabell 1: Ord i programmeringsoppgaver

Under presenteres et eksempel på en oppgave som både inneholder begreper om programmering, og hvor elevene er nødt til å gjennomføre en handling. Ved dette er det synlig hvordan vi har gjennomgått alle oppgavene i lærebøkene, for å finne hvilke oppgaver vi har som kan kategoriseres som programmeringsoppgaver. Vi valgt en oppgave som tydelig viser hva en programmeringsoppgave i denne studien må inneholde.



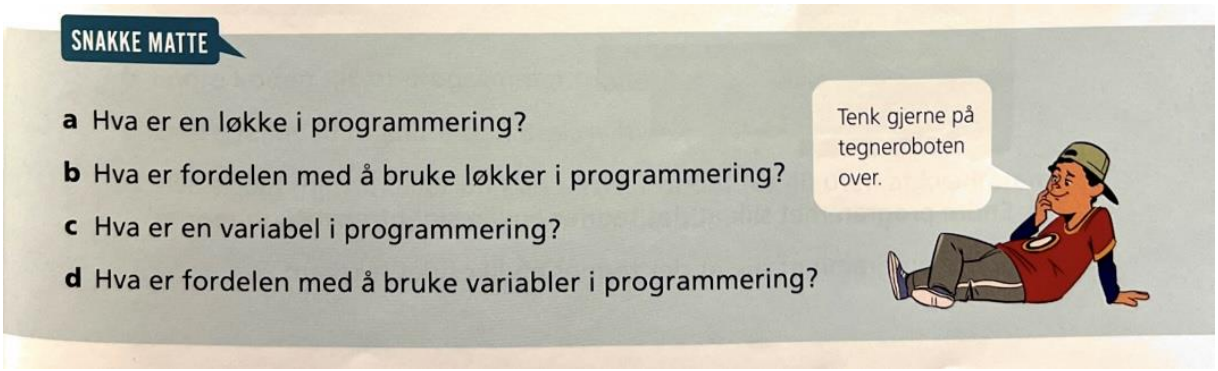
Figur 2: 7.30 (Multi 6B) (Alseth et al., 2021, s. 87)

Figur 2 er en oppgave som går ut på at eleven skal følge koden som allerede er gitt ved å føre blyanten på samme måte som er forklart. For å argumentere for at dette er en analog programmeringsoppgave er det nødvendig å få frem at elevene her arbeider med programmering ved at de følger en algoritme for å løse et problem; altså at de skal tegne figuren. Dette stemmer overens med vår definisjon på programmering, der elevene må analysere et problem og deretter finne en løsning. Denne løsningen skal de gjennomføre ved å bruke teknologi eller ved en håndfast handling. I denne oppgaven må eleven tegne etter koden og derfor blir det menneskelig atferd.

Oppgaver som ikke oppfyller kravene for å være en programmeringsoppgave blir også talt opp for å kunne se en sammenheng. De nylig nevnte kategoriene i *tabell 1* har vi brukt under letingen etter programmeringsoppgaver gjennom alle bøkene. Likevel for å finne svar på problemstillingen kreves det et dypdykk i oppgavene som innebærer programmering. Dette gjøres ved å analysere alle programmeringsoppgavene ut ifra vårt rammeverk og deretter presentere et utvalg av dem.

Under har vi et eksempel på en oppgave som ikke fyller kravene på en programmeringsoppgave selv om noen av begrepene innen programmering er nevnt. Grunnen til at denne oppgaven likevel ikke oppfyller kravene for å være en programmeringsoppgave er

at elevene ikke skal gjennomføre noe håndfast. Det oppgaven ber om er kun at elevene skal samtale om hva de ulike begrepene innen programmering betyr. Dermed skal de ikke gjennomføre eller løse en oppgave. Oppgaven er plassert i starten av kapittelet og kan fungere som en hensiktsmessig innføring av temaet. På grunn av at den ikke oppfyller kravene for å være en programmeringsoppgave i denne masteren, er den hverken regnet med eller analysert.



Figur 3: Snakke matte (Matemagisk 6B) (Kongsnes, 2021, s. 105)

3.4.2 Vertikal analyse

Den vertikale analysen dreier seg som nevnt om å gjøre et dypdykk og gå nærmere inn på oppgaver i detalj (Charalambous et al., 2010, s. 122). I deres studie har de delt dette perspektivet inn i tre kategorier som de har navngitt: *kommunisert til elevene, kreves av elevene* og *sammenhenger (*Charalambous et al., 2010, s. 122). Vi ser at dette ikke vil være hensiktsmessig å benytte i denne studien fordi vi skal se på lærebøker. For å finne svar på vår problemstilling, har vi valgt å bruke de ulike konseptene og fremgangsmåtene i Barefoot Computing's (u.å.-a) modell for algoritmisk tenking som kategorier. Disse er utdypet og redegjort for i kapittel 2.6 Rammeverk. Vi analyserer som nevnt alle programmeringsoppgavene som ble funnet i den horisontale analysen for å se nærmere på dem gjennom komponentene i algoritmisk tenking. I kapittel 4 Analyse, vil åtte programmeringsoppgaver trekkes frem som og forklares på et dypere plan. Alle de andre programmeringsoppgavene kommer kun til syne i oppsummeringstabeller.

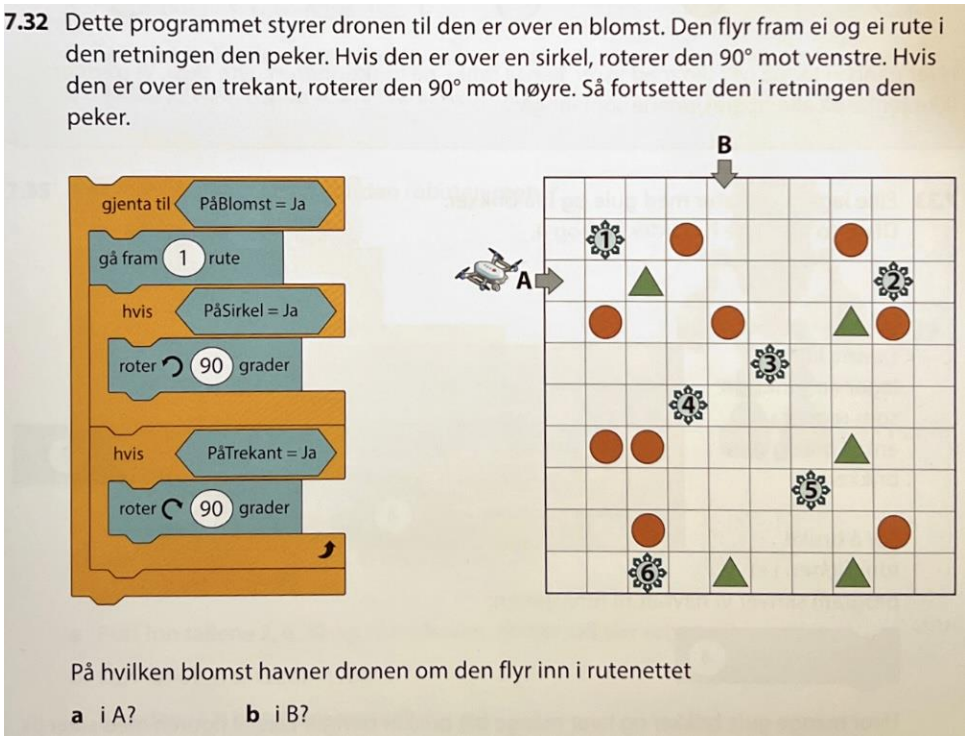
Å se etter komponentene åpner opp for muligheten til å se programmering i samspill med algoritmisk tenking. Dette kan gjøres gjennom å utforske hvilke konsepter og fremgangsmåter som det arbeides med i programmeringsoppgavene. Dermed kan vi gå dypere inn i hvordan programmeringsoppgavene legger til rette for algoritmisk tenking. Dette gjøres ved å se på hvilke konsepter og fremgangsmåter innen algoritmisk tenking elevene bruker i hver av de

utvalgte oppgavene. Det som er annerledes med *tabell 2* i forhold til de tabellene vi tar i bruk i kapittel 4 Analyse, er at den inneholder en forklaring for hver komponent i forhold til hva en oppgave må inneholde for at vi skal sette kyss i de respektive rutene. Forklaringene i rutene er laget med utgangspunkt i Barefoot Computing's (u.å.-a) modell om algoritmisk tenking. For å kunne bruke den til å analysere oppgavene har vi vinklet forklaringene til å omhandle hva en oppgave må inneholde for å oppfylle kravet. Under selve analysen vil ruten med forklaring enten være blank eller inneholde et kryss, avhengig av om oppgaven oppfyller kravet til begrepet. For hver oppgave fyller vi inn navnet på oppgaven på øverste rad. Deretter vil vi krysse av i de rutene som tilhører konseptene og fremgangsmåtene oppgaven inneholder. Under tabellen med forklaringer, kommer et eksempel på hvordan vi har tenkt til å gjennomføre analysen.

Konsepter					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
Oppgaver som krever at elevene må hente frem forkunnskaper for å kunne lage en løsning eller beskrive hvorfor noe skjer.	Oppgaver som krever at elevene arbeider med en rekke med regler som får noe gjort.	Oppgaver som er så store at elevene blir nødt til å dele de opp i mindre forståelige deler.	Oppgaver hvor eleven enten ser, ser etter eller skal skape noe som gjentar seg.	Oppgaver med mye unyttig informasjon som elevene må luke bort.	Oppgaver som ber elevene ta valg, bedømme det de selv har gjort.

Fremgangsmåter				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
Oppgaver hvor elevene arbeider med noe ukjent de må gjøre seg kjent med.	Oppgaver hvor elevene blir bedt om å lage noe.	Oppgaver som krever at elevene må sjekke algoritmen sin for feil.	Oppgaver som sier noe om at elevene må fortsette å prøve helt til de får det til.	Oppgaver som sier noe om at elevene ikke skal jobbe individuelt.

Tabell 2: Analyse kriterier for den vertikal analyse (egen oversettelse) (Barefoot Computing, u.å)



Figur 4: 7.32 (Multi 6B) (Alseth et al., 2021, s. 89)

Konsepter i oppgave 7.32 side 89 i Multi 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
x	x	x		x	

Fremgangsmåter i oppgave 7.32 side 89 i Multi 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
		x		

Tabell 3: Tabell 3: Vertikal analyse av oppgave 7.32 s.89 (Multi 6B)

For å utdype hva *tabell 2* betyr og hvordan vi har tenkt, forklarer vi med en eksempeloppgave. Ut ifra oppgaveteksten tilhørende *figur 4,* får vi informasjon om hva slags fremgangsmåter og konsepter elevene skal arbeide med. Først vil vi tydeliggjøre at dette er en programmeringsoppgave av type analog programmering, fordi eleven blir bedt om å flytte en drone ved å «kjøre et program» (følge algoritmen) uten å bruke digitale verktøy. Dette ser vi gjennom at problemet til eleven er at hen har fått steg-for-steg-instruksjoner som må følges for å finne ut av hvor dronen havner om den flyr inn i A først og deretter B. Problemet kan deles opp i ulike håndterbare problemer, altså *dekomponere* problemet. Først må eleven lese, forstå og bruke *logikk* til den *algoritmen* som er avbildet ved siden av kartet og dronen. Deretter kan eleven dele opp slik at hen flytter dronen ett og ett steg helt til den er fremme på

blomsten. Det neste konseptet det er krysset av for er abstraksjon. Grunnen til at eleven i denne oppgaven blir nødt til å arbeide med abstraksjon er at det står mye tekst over som inneholder samme informasjon som i algoritmen. Dette kan oppleves som mye å forholde seg til og derfor blir det nødvendig å plukke ut den nødvendige informasjonen for å løse oppgaven.

Avkrysningen for *feilsøking* kommer av at oppgaven krever at dronen skal havne på en blomst til slutt. Dersom dronen ikke havner på en blomst blir eleven nødt til å finne ut av hvor feilen ligger for så å forsøke igjen. To eksempler på komponenter av algoritmisk tenking som ikke har blitt krysset av for er *fikling* og *holde ut*. Elevene hadde ikke noe å utforske eller fikle med, i henhold til definisjonen på å fikle i denne masteren, fordi oppgaven ikke legger opp til at elevene skal jobbe enten på datamaskin eller i ukjente programmer. Når det gjelder *holde ut* er det en kategori som er vanskelig å se ut ifra oppgaveteksten. Dersom vi skulle hatt muligheten til å krysse av *holde ut*, måtte vi ha observert elever utføre oppgaven for å kunne få et troverdig gyldig svar, eller oppgaven måtte ha oppfordret til at elevene skulle holde ut. På grunnlag av de ovennevnte begrunnelsene ser *tabell 3* ut som den gjør.

3.5 Reliabilitet og validitet

Dette delkapittelet omhandler hvordan reliabilitet og validitet blir ivaretatt i vår oppgave, det blir også definert hva disse begrepene betyr.

Reliabilitet omhandler det at analysen burde gjøres slik at den er repliserbar (Bratberg, 2021, s. 138). Ifølge Clark et al. (2021, s. 271) er det to nøkkelfaktorer som må til for å kunne øke reliabiliteten for innholdsanalysene. Dette er at forskerne må være objektive og systematiske (Clark et al., 2021, s. 271). Det er vanskelig å ikke legge egne meninger, forkunnskaper, fordommer og tanker inn i masteroppgaven (Krippendorf, 2019, s. 6). Det er mennesker som gjennomfører studien, som igjen vil gi en påvirkning på studien. Likevel er dette noe det er viktig å streve etter å få til slik at masteroppgaven blir mest mulig rettmessig. Om studien hadde inneholdt tall i større grad vil det være enklere å øke påliteligheten på oppgaven fordi tall er noe som ikke kan endres på, og åpner dermed ikke opp for tolkning på samme måte som tekst (Postholm & Jacobsen, 2018 s. 223-224).

Det kan se ut til at i en innholdsanalyse sørger selve metoden for at både reliabiliteten og validiteten blir ivaretatt til en viss grad. På grunn av at datamaterialet er konstant legger Krippendorf (2019, s. 24) frem at det vil være enklere å gjenskape og få samme resultat. Å forske på trykt tekst vil også være en fordel i form av at uansett når eller hvor en annen forsker skal forske på det samme, vil også teksten være den samme Krippendorf (2019, s. 24). På en annen side er temaet vi skriver om i rask utvikling. Dermed kan det ikke sies med sikkerhet at en annen forsker ville fått samme svar om ti år. Læreboken og oppgaveteksten vil som nevnt være den samme, men programmering, algoritmisk tenking og konteksten rundt kan ha utviklet seg.

I denne oppgaven vil begrepet inter-rater-reliability være viktig, da vi er to som skriver sammen. Postholm og Jacobsen (2018, s. 154-155) beskriver begrepet ved at om det er flere som er med på å kategorisere data, er det viktig å være konsekvent i beslutningene som tas. Dette har vært viktig gjennom flere faser i arbeidet med dataen. For eksempel da vi skulle definere hva som er en programmeringsoppgave og ikke. Oppgavens reliabilitet var avhengig av at vi hadde klare og tydelige kriterier slik at kategoriseringen ble gjennomført på samme måte gjennom hele boka, og ikke minst i alle de ulike bøkene. Disse tydelige kriteriene var også for å øke sjansen for at en eventuell gjenskapelse av dette prosjektet ville ende opp med samme data. Dette med å være konsekvent i avgjørelser er noe vi arbeidet med gjennom hele oppgaven både for å forenkle overfor oss selv, men også for å gi selve oppgaven et best mulig uttrykk. Våre tolkninger kan også ses på som en slags feilkilde, da tolkninger er personbasert og er mulig å tolke forskjellig.

Andre betraktninger å ta hensyn til er oppgavens validitet. Validitet omhandler i hvilken grad oppgaven har oppnådd å måle det som er intensjonen å måle (Bratberg, 2021, s. 138). Dette er valg som gjøres i gjennomføringen av forskningen som skal skape troverdighet (Krippendorf, 2019, s. 361). Forskerens mål om å minimalisere feilkilder kan være med på å øke validiteten til en oppgave (Clark et al., 2021, s. 192). I denne sammenheng presenterer vi mulige feilkilder i kapittel 3.7. Etter å ha funnet problemstilling var det viktig å finne en metode som kunne måle det vi ønsket å måle. Rammeverket er beskrevet grundig av flere grunner. En av dem var for vår egen del slik at vi begge er enige om hvordan oppgavene skal analyseres. Før vi kunne begynne å analysere oppgavene måtte vi finne oppgavene i læreboken. I denne prosessen hadde vi også laget klare og tydelige kriterier for hva en programmeringsoppgave måtte inneholde for å kunne defineres som dette. Rammeverket var også grundig beskrevet

for å være sikker på at det rammeverket vi har valgt finner svar på det vi ønsker å forske på. Feil valg av rammeverk kan være med på å føre til at oppgaven blir lite troverdig. Et dårlig rammeverk i denne masteroppgaven, kan være et rammeverk hvor analysen av alle programmeringsoppgavene får mange og omtrent samme kryss. Derfor var det viktig å anvende et rammeverk som er spesifikt i hvilke områder som arbeides med.

Krippendorf (2019, s. 361) bruker begrepet face validity om det som oppleves som selvfølgelig eller felles sannhet. Dette er en av grunnene til at vi valgte å analysere lærebøker. Under forskning på hvordan programmeringsoppgaver i matematikk kan legge til rette for algoritmisk tenking, kan det oppfattes selvfølgelig at forskeren retter seg mot de som har laget oppgaver i henhold til den lovpålagte læreplanen. Dette kan være med på å gi det mest virkelighetsnære resultatet ved at de aller fleste skolene har lærebøker og at det er innenfor læreplanens rammer at elevene skal ha kunnskap om både programmering og algoritmisk tenking. Grunnen til at vi mener at dette resultatet vil bli virkelighetsnært er at det er en felles sannhet i samfunnet at det som står i læreplanen er det som elevene skal kunne. Dermed kan det tenkes at lærebøkene har tatt utgangspunkt i læreplanverkets innhold.

3.5.1 Muligheter og begrensninger

Bratberg (2021, s. 141) skriver at uavhengig av hvordan metode som velges for å svare på problemstillingen vil metoden ha konkrete styrker og begrensninger. En av begrensningene som ligger for denne oppgaven er at det kun er oppgavene i de trykte lærebøkene vi ser på. Programmering er ofte noe som foregår digitalt og derfor ville det kanskje ha blitt et annet utfall om studien inkluderte nettressurser og deler av et helhetlig læreverk. En annen begrensning er at vi ikke ser hva som foregår i klasserommet. Derfor vil det ikke være mulig å kunne se hvordan oppgavene fungerer i praksis, eller hvordan læreren velger å bruke dem. Derimot får vi muligheten til å se på oppgavene sånn som forlaget og forfatterne har tenkt at det skal være uten å være innom et ekstra ledd med tolkninger, altså læreren. En annen faktor er at vi ikke ser på lærerveiledninger hvordan forfatterne og forlaget skriver hvordan de anbefaler å arbeide med innholdet i læreboken. Hadde vi gjort dette kunne vi ha tolket litt mer hvordan programmeringsoppgavene kunne vært jobbet med i praksis. Dette ble valgt bort på grunnlag av masteroppgavens størrelse og omfang.

3.6 Forskningsetiske betraktninger

Her beskrives hva forskningsetiske betraktninger er for noe. Deretter presenteres hva vi må ta hensyn til i vår oppgave. Et av de etiske prinsippene innenfor forskning er at forskeren har et ansvar for å legge frem dataen på en objektiv måte uten å legge til for mange egne tanker (Postholm &Jacobsen, 2018, s. 246). Grunnet at dette er en analyse av oppgaver i lærebøker vil en ting vi må ta hensyn til være lærebokforfatterne. Der vi ikke kan legge ord i munnen deres. Vi har som nevnt vært i kontakt med forlagene angående hvilken kompetanse de som har utviklet programmeringsinnholdet i bøkene har. Der vi fikk svar av forlagene, spurte vi om vi fikk lov til å sitere svaret deres i oppgaven vår. Dermed fikk de en mulighet til å si nei om de ikke ville det, eller endre svaret slik at det fikk den ordlyden de ønsket teksten skulle ha. I masteroppgaven når vi refererer til denne samtalen på e-post har vi valgt å anonymisere enkeltpersonene som har svart. En av grunnene til at vi har gjort dette valget er at e-post er et ganske lukket forum, og når det gjelder kommunikasjon som er privat peker Clark et al. (2016, s. 129) på at forskeren har et ansvar for å ivareta den andres anonymitet. Samtidig var det forlaget vi ønsket å stille spørsmål til, og dermed ble personen som besvarte e-posten en representant for forlaget. Det er altså ikke personens personlige mening vi spør etter, men heller forlagets mening som denne representanten formidler. Dermed kan det være hensiktsmessig å anonymisere forlagets representant for å få frem budskapet.

3.7 Mulige feilkilder

I denne masteroppgaven kan det være enkelte deler som kan være mindre korrekt.
Som nevnt er det viktig å prøve å minke disse feilkildene slik at oppgaven fremstår mer gyldig (Clark et al., 2021, s. 129). For å få til dette er det enkelte deler av arbeidsprosessen vi har tenkt nøye gjennom. En eventuell feilkilde kan være at antallet oppgaver kan være feil. Vi som forskere er bare mennesker og kan dermed gjøre menneskelige glipper som å glemme å telle en oppgave eller sette et kryss i en rute når det egentlig var meningen å sette krysset i ruten til venstre. Likevel har vi vært to om å telle og analysere oppgaver, samt sjekket flere ganger at resultatet vårt samsvarer. Dette gjorde vi for å minske denne feilkilden mest mulig. Andre slike feil kan være at vi har tolket innholdet i oppgavene annerledes enn andre ville gjort. Dette kan føre til at andre som gjennomfører forskningen med samme materiale som oss kan få et annet svar. Samtidig har vi satt sammen et rammeverk som vi ønsket skulle være så presist som mulig for å unngå dette i høyeste grad. Tilspisningen av kategoriene gjorde vi

også fordi som nevnt tidligere er det viktig for oppgavens validitet at den er mulig å replisere (Bratberg, 2021, s. 138). En siste mulig feilkilde som kan trekkes frem er at på grunn av at vi har oversatt rammeverket vårt fra engelsk kan det ha skjedd misoppfatninger under oversettelsen. Under arbeidet med oversettelsen har vi samarbeidet om å få det så korrekt som mulig for å unngå denne feilkilden.

4 Analyse

Problemstillingen for denne masteroppgaven tar for seg programmeringsoppgaver og hvordan de kan legge til rette for algoritmisk tenkning. For å finne ut av dette har vi sett igjennom 1309 oppgaver til sammen i alle de tre lærebøkene. Vi vil i dette kapittelet legge frem funn tilknyttet den horisontale analyse, som blant annet viser til forekomsten av antall programmeringsoppgaver. I tillegg vil bøkenes struktur og oppdagelsene vi gjorde bli presentert i en tabell. Deretter vil vi analysere i dybden enkelte oppgaver fra den horisontale analysen. Kravene for disse oppgavene er at de må være innenfor kategorien med programmering og algoritmisk tenkning-oppgaver. Dette vil vi gjøre som nevnt i kapittel 3.4.2 Vertikal analyse, ved å bruke modellen utarbeidet av Barefoot Computing (u.å.-a) om algoritmisk tenking. Under delkapitlene 4.1 Horisontal analyse og 4.2 Vertikal analyse kommer funnene for bøkenes overordnede struktur. For å gjøre det så oversiktlig som mulig har vi valgt å ta for oss en bok om gangen. Dette er også gjort for at vi skal unngå å sammenligne bøkene, da dette ikke er relevant for problemstillingen tilknyttet denne masteroppgaven. Likevel observerer vi objektive forskjeller som kan være interessante å trekke frem for å skape en forståelse av bøkene.

4.1 Horisontal analyse:

I dette delkapittelet presenterer vi hvilke funn vi har kommet frem til i den horisontale analysen. Det er nyttig å nevne at i analysen har vi også angitt hvor mange oppgaver i de ulike kategoriene det befinner seg i hvert kapittel. Disse tallene presenteres ikke i form av statistikk, men de er flettet inn i teksten der de er hensiktsmessige å ha med.

4.1.1Matemagisk 6B

I Grunnboka til Matemagisk 6B har de delt inn boka i fem kapitler. Under er en tabell med sammenfattet overordnet bakgrunnsinformasjon om boka. Ut ifra denne informasjonen blir det synlig at i kapittel syv vil det være større sannsynlighet for å finne programmeringsoppgaver, fordi de nevner begreper innenfor hovedinnhold.

	Tittel	Hovedinnhold	Antall oppgaver
Kapittel 5	Vinkler og parallelle linjer	Ulike typer vinkler, samt begreper som vinkelrett og parallelle linjer	56
Kapittel 6	Trekanter og firkanter	Begreper for ulike typer trekanter og firkanter og begreper som vinkelsum, diagonal og egenskap	71
Kapittel 7	Geometriske mønstre	Ulike begrep for kongruensavbildninger, begreper innen koordinatsystem, samt begreper i relasjon til programmering som løkke, variabel og funksjon.	64
Kapittel 8	Areal og omkrets	Begrepene i dette kapittelet omhandler hvordan måle ulike geometriske figurer samt hva en sammensatt figur er.	86
Kapittel 9	Tredimensjonale figurer	Viktige begreper omhandler ulike tredimensjonale geometriske figurer.	50

Tabell 4: Oversikt over læreboken Matemagisk 6B fra Aschehoug

For å kunne svare på problemstillingen til vår masteroppgave måtte vi videre telle opp antallet oppgaver boka inneholder både med og uten programmering. Ved å gjøre dette fikk vi et bilde av hvor mange oppgaver av hver type det var i hvert kapittel, men også i hele boken. I Matemagisk fant vi ut at det var 20 av 327 oppgaver som inneholdt programmering, dette tilsvarer 6% av alle oppgavene i boka. Vi fant mest programmeringsoppgaver i kapittel 7 med hele 19 av 20 programmeringsoppgaver, men det var også en oppgave i kapittel 5. Temaet innenfor kapittel 7 og 5 er geometri. Vi har også observert at hele boken omhandler dette temaet, selv om de har ulike navn på kapitlene. Det er forholdet mellom antall programmeringsoppgaver og andre oppgaver *figur 5* illustrerer, for å skape et tydelig bilde av forholdet.

Figur 5: Oppgaveoversikt Matemagisk 6B

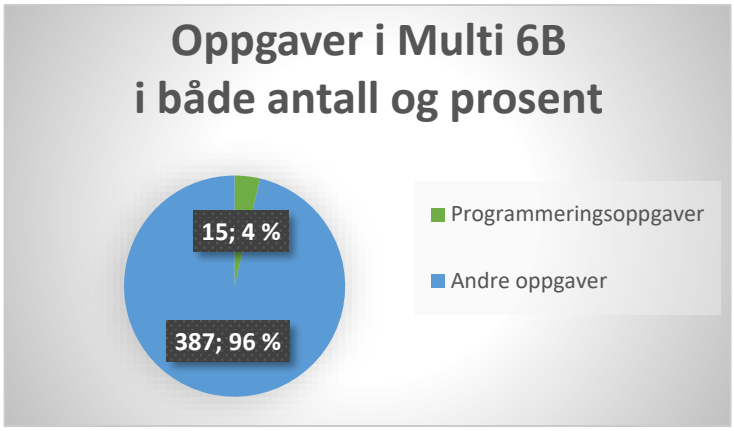
4.1.2 Multi 6B

I Multi 6B har de delt boka inn i fire kapitler. *Tabell 5* viser kortfattet overordnet bakgrunnsinformasjon om boka. Under vil det komme en forklaring på *tabell 5* og *figur 6*.

	Tittel	Hovedinnhold	Antall oppgaver
Kapittel 5	Brøk	Omhandler likeverdige brøker, forholdet mellom brøk, prosent og desimaltall, ulike former for utregning av brøk, samt tekstoppgaver om brøk.	113
Kapittel 6	Mønster og koordinatsystem	Utforske mønstre og symmetri, samt gjennomføre bevegelse og skape mønster ved bruk av speiling, rotasjon og forskyvning. De skal også lære å bruke koordinatsystem.	105
Kapittel 7	Algebra og programmering	Beskrive og fortsette figurtall, arbeide med, og lage regler, formler og algoritmer.	59
Kapittel 8	Tall og regning	Kunne bestemme tallverdi, avrunding, multiplikasjon og divisjon av desimaltall, bruke regneark og bruke matematikk i praktiske situasjoner.	125

Tabell 5: Oversikt over læreboken Multi 6B fra Gyldendal

Videre i analysen undersøkte vi og talte opp antall oppgaver med og uten programmering. Dette kan være med på å gi et bilde av hvor stor del av læreboken som inneholder programmering. I Multi 6B fant vi 15 av 402 programmeringsoppgaver hvilket utgjør 4% av alle oppgavene i boken. Resten av oppgavene var 387 stykker, hvilket tilsvarer 96%. For et oversiktlig bilde på forholdet mellom oppgavene er det vedlagt et sektordiagram i *figur 6* (på neste side) som viser nettopp dette. Programmeringsoppgavene strakk seg over flere kapitler og var på den måten spredt utover i boken, men likevel også samlet ved at de var samlet i de ulike kapitlene. I kapittel 6 befinner det seg tre programmeringsoppgaver. Resten av oppgavene befinner seg i kapittel 7 som omhandler *algebra og programmering*, altså 12 stykker.



Figur 6: Oppgaveroversikt Multi 6B

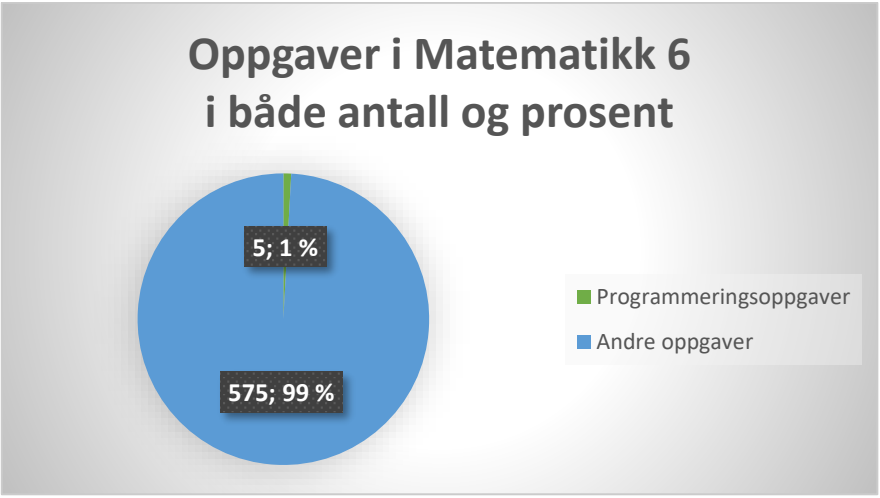
4.1.3. Matematikk 6 fra Cappelen Damm

Matematikk 6 fra Cappelen Damm er delt inn i seks kapitler som er beskrevet under i *tabell 6* med overordnet informasjon om boka.

	Tittel	Hovedinnhold	Antall oppgaver
Kapittel 1	Tall	Kunne utforske, bruke og beskrive skriftlige- og hoderegningsstrategier i addisjon og subtraksjon, plassverdisystemet, samt vurdere og gjøre hensiktsmessige overslag.	115
Kapittel 2	Desimaltall	Omhandler utforsking og regning med desimaltall og hele tall, samt å løse hverdagslige problemer i sammenheng med brøk, desimaltall og prosent. Elevene skal også kunne forklare egne tenkemåter	81
Kapittel 3	Geometri	Utforske og beskrive egenskapene til todimensjonale figurer, strategier for å regne ut areal og omkrets for så å bruke det i praktiske situasjoner. Videre skal de se sammenhenger mellom ulike måleenheter, og radius, diameter og omkrets i sirkler.	101
Kapittel 4	Multiplikasjon og divisjon	Utvikle metoder for multiplikasjon og divisjon, utforske og sammenligne strategier for regning med desimaltall og hele tall, for deretter å skape og løse hverdagsproblemer innen temaet. Elevene skal også kunne gjøre beregninger i regneark ved bruk av formler.	121
Kapittel 5	Algebra	Utforske og beskrive symmetri, speiling, forskyvning, geometriske mønstre og figurer, og lage formler som uttrykker sammenhenger i praktiske situasjoner. Elevene skal også kunne løse enkle ligninger og utforske og lage enkle algoritmer for koding i rutenett.	84
Kapittel 6	Tredimensjonale figurer	Utforske og beskrive egenskaper ved tredimensjonale figurer, ulike strategier for å regne ut volum, samt mål for areal og volum i praktiske situasjoner.	78

Tabell 6: Oversikt over læreboken Matematikk 6 fra Cappelen Damm

I Matematikk 6 fra Cappelen Damm har vi også talt opp programmeringsoppgaver og andre oppgaver. I denne boka befinner det seg 5 av 580 oppgaver som innebærer programmering. Dette tilsvarer 1% som sier noe om at det er tung overvekt av oppgaver som ikke inneholder programmering. *Figur 7* viser frem dette. Alle programmeringsoppgavene vi fant befinner seg i kapittel 5 Algebra. Samtidig som det står i *tabell 6*, arbeider de med geometriske mønstre i algebrakapittelet.



Figur 7: Oppgaveoversikt Matematikk 6 fra Cappelen Damm

4.2 Vertikal analyse

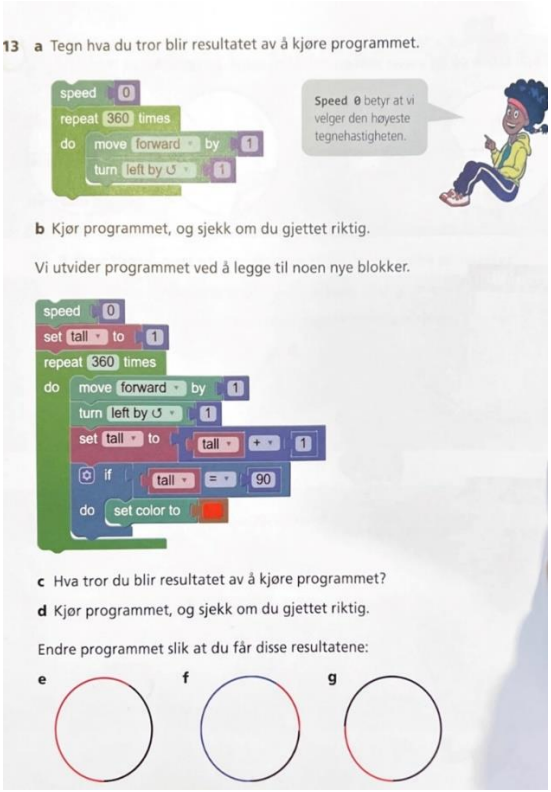
I dette delkapitlet analyseres oppgavene vi har funnet ved å ha gjennomført den horisontale analysen. Analysen av oppgavene vil dreie seg om det som konkret står i oppgaveteksten, ikke det som kan skje om elevene eller læreren gjør sånn eller sånn. Dette betraktes opp imot modellen til Barefoot Computing (u.å.-a) om algoritmisk tenking. Dette vil si at det gjennomføres et dypdykk i oppgavene, for å finne ut om programmeringsoppgaver legger til rette for algoritmisk tenkning.

4.2.1 Utvalgte analyserte oppgaver

Under vil det presenteres syv oppgaver fra lærebøkene. Oppgavene vi har valgt ut er de oppgavene som skiller seg ut i form av hvilke konsepter og fremgangsmåter som arbeides med, dermed vil det ikke være like mange oppgaver fra hver bok. Først vil det være bilde av oppgaven, deretter et bilde av en tabell som viser hvilke konsepter og fremgangsmåter

elevene er gjennom under arbeidet med oppgaven. Under tabellen vil det være en forklaring på hvorfor oppgaven dekker akkurat disse elementene innenfor algoritmisk tenking.

Analyseoppgave 1: Oppgave 13 på side 20 kapittel 5



Figur 8: Oppgave 13 (Matemagisk 6B) (Kongsnes et al., 2021, s.20)

Konsepter i oppgave 13 side 20 i Matemagisk 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
x	x		x		

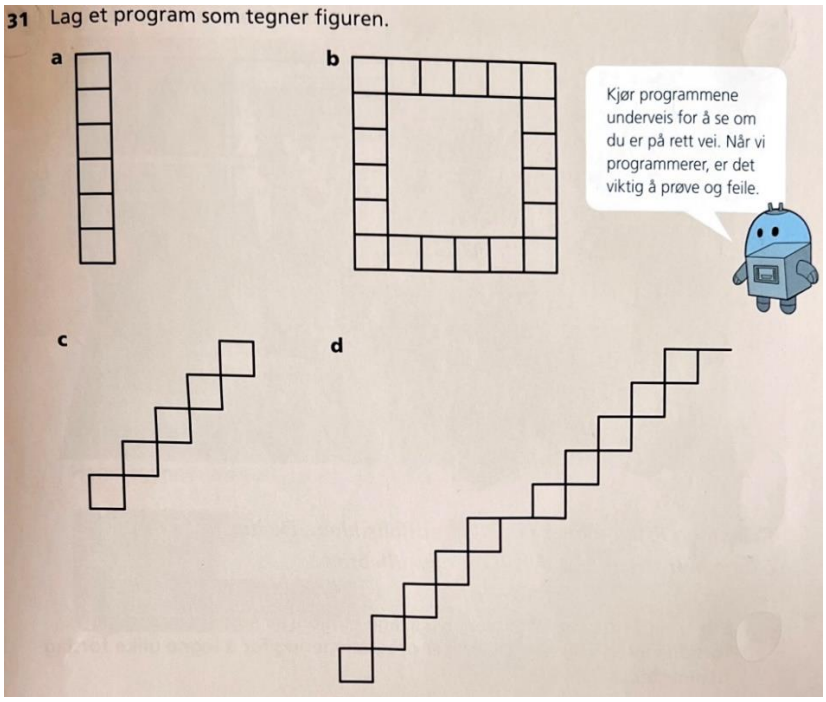
Fremgangsmåter i oppgave 13 side 20 i Matemagisk 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
	x	x		

Tabell 7: Vertikal analyse av oppgave 13 s.20 (Matemagisk 6B fra Aschehoug)

I analyseoppgave 1 får eleven ulike utfordringer knyttet til programmering. Det kan se ut til at eleven skal bruke et blokkprogrammeringsprogram. Først blir elevene bedt om å se på *algoritmen* som er tegnet opp og gjette hva resultatet blir om programmet kjøres. Dette krever at elevene bruker *logisk tenking* til å systematisere informasjonen de får og de må ha

kunnskap om ulike figurer. Denne informasjonen og kunnskapen skal elevene bruke for å kunne gi et svar på hva de tror resultatet blir. I oppgave b) skal de kjøre programmet for å sjekke om de har gjettet riktig. Dette ser vi på som en form for *feilsøking,* fordi elevene må sjekke om deres svar samsvarer med resultatet av å kjøre programmet. De to neste oppgavene gjentar de to første. Når elevene har gjennomført de fire første deloppgavene skal de i gang med å *skape*. Her skal de bruke den *algoritmen* som de arbeidet med i oppgave a)-d) til å *skape* tre ulike sirkler. Denne oppgaven fører til at elevene også må arbeide med *algoritmer* i form av at de er nødt til å forstå den som allerede er laget av boka, men de skal også modifisere den slik at den gir resultatet de skal frem til. For å løse denne oppgaven finnes det *mønster* i form av *algoritmen* som kan ses på som et *mønster*, og at alle tre figurene er sirkler. Dette kan være til hjelp for eleven i arbeid med oppgaven.

Analyseoppgave 2: Oppgave 31 s. 111



Figur 9: oppgave 31 (Matemagisk 6B) (Kongsnes et al., 2021, s.111)

Konsepter i oppgave 31 side 111 i Matemagisk 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
x	x		x		

Fremgangsmåter i oppgave 31 side 111 i Matemagisk 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
	x	x		

Tabell 8: Vertikal analyse av oppgave 31 s.111 (Matemagisk 6B fra Aschehoug)

I analyseoppgave 2 får elevene presenter fire forskjellige figurer som de skal gjenskape gjennom et program. Dermed må de bruke *logikk* for å *skape* en *algoritme* som former figuren i oppgaven når programmet kjøres. For å kunne løse oppgaven vil det være naturlig å *dekomponere* ved å dele opp figuren i ulike deler og lete etter gjentakende deler, altså *mønstre*. Roboten oppe i det høyre hjørnet ber elevene også om å dele opp oppgaven ved at de skal kjøre programmet underveis, og dermed gjøre litt og litt av oppgaven. Ut ifra dette ser det også ut til at elevene skal anvende et blokkprogram. Dette innebærer også at elevene skal *fikle,* ved at de blir bedt om å sjekke at de er på rett vei. Dermed må de sjekke «hva skjer hvis jeg gjør det på denne måten eller denne måten.» Oppgaven presenterer en fasit og krever tydelig at elevene skal lage et program som *skaper* akkurat disse figurene. Derfor kan elevene bli nødt til å *feilsøke* dersom deres figur ikke blir lik som figuren i oppgaven. Grunnen til at vi valgte å trekke frem denne oppgaven er at det er den eneste oppgaven som inneholder fremgangsmåten *holde ut*. Dette argumenterer vi for ved at roboten oppe i hjørnet sier at «Når vi programmerer, er det viktig å prøve og feile,» og ved dette forteller han elevene at de er nødt til å fortsette å prøve selv om de gjør feil. Å ikke gi seg, selv om oppgaven er vanskelig eller eleven gjør mye feil, er ifølge Barefoot Computing's (u.å.-l) modellen om algoritmisk tenking det å *holde ut* innebærer.

Analyseoppgave 3: Oppgave V s. 114

Figur 10: oppgave V (Matemagisk 6B) (Kongsnes et al., 2021, s.114)

Konsepter i oppgave V side 114 i Matemagisk 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
x	x		x		x

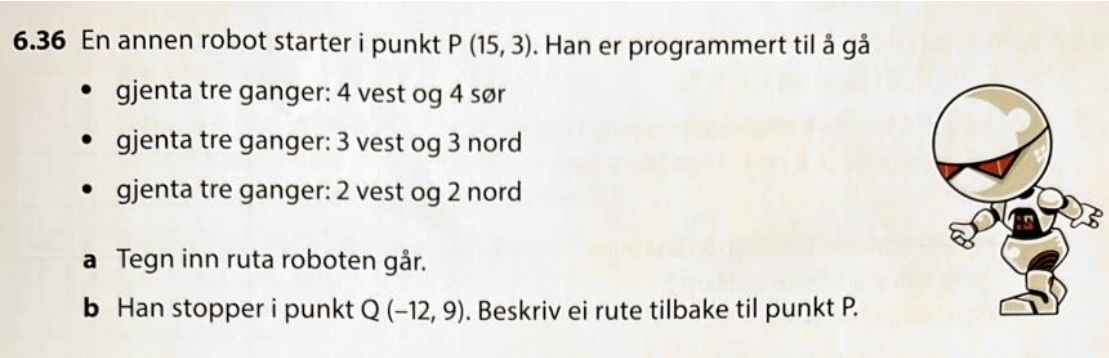
Fremgangsmåter i oppgave V side 114 i Matemagisk 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
	x			

Tabell 9: Vertikal analyse av oppgave V s.114 (Matemagisk 6B fra Aschehoug)

Her presenteres en annen oppgave vi har analysert. Denne oppgaven valgte vi på grunn av at elevene får aktivt beskjed gjennom teksten at de skal jobbe med programmering. Vi mener dette er en programmeringsoppgave ved å vise til vår definisjon på programmering som en aktivitet som innebærer å analysere et problem, designe en løsning, som kan tas i bruk via en datamaskin eller menneskelig atferd, og deretter bruke den. I analyseoppgave 3 ser vi at elevene er nødt til å analysere problemet, som er at de skal lage et *mønster* som skal være på en spillebrikke eller et spillebrett. Valget mellom spillebrikke eller spillebrett gjør at elevene må gjøre en *evaluering,* i forhold til hva de synes er mest hensiktsmessig til det *mønsteret* de ønsker å *skape*. Elevene må også selv bestemme om de ønsker å gjennomføre oppgaven analog eller anvende et digitalt verktøy. Under denne *evalueringen* har eleven bruk for *logisk tenking* i forhold til hva som egner seg best til å løse oppgaven på best mulig måte. Dette vil også si at elevene må *skape* noe, derfor har vi krysset av ruten for *skape* innenfor

fremgangsmåte. Selve oppgaven er å skape et *mønster*, hvilket gjør at det er selvfølgelig at eleven arbeider med å se etter *mønstre*. Samtidig vil det også være naturlig at elevene lager en *algoritme* som skaper *mønsteret* når programmet kjøres. Vi ser også at eleven må velge hvilket programmeringsprogram de skal velge og om det skal gjennomføres ved å bruke en datamaskin eller gjøre det for hånd på papir.

Analyseoppgave 4: Oppgave 6.36 s.53



Figur 11: 6.36 (Multi 6B) (Alseth et al., 2021, s. 53)

Konsepter i oppgave 6.36 side 53 i Multi 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
x	x	x			

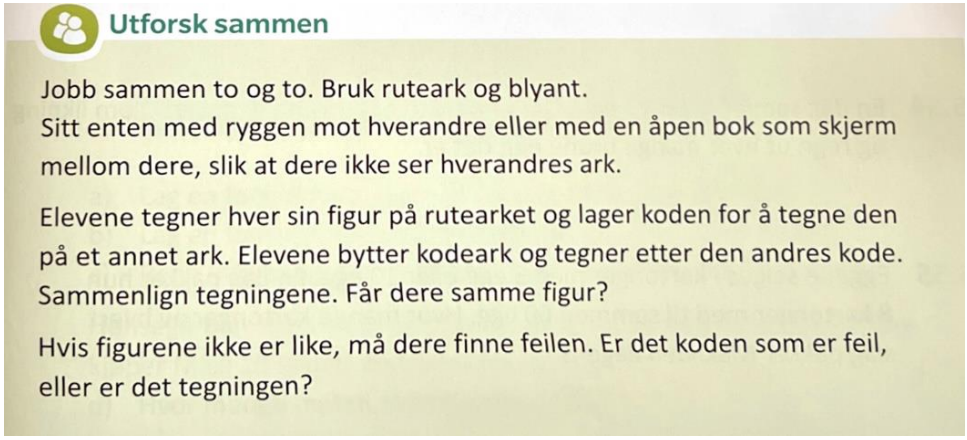
Fremgangsmåter i oppgave 6.36 side 53 i Multi 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
	x			

Tabell 10: Vertikal analyse av oppgave 6.36 s.53 (Multi 6B fra Gyldendal)

Analyseoppgave 4 har vi definert som en programmeringsoppgave på grunn av at den nevner begrepet «programmert.» Dette gjør at elevene forstår at det er snakk om programmering når de arbeider med oppgaven. Når vi vet at dette er en programmeringsoppgave analyserte vi den ved å bruke rammeverket. Elevene skal *skape* den gitte ruten som roboten er programmert til å gå i oppgave a). Etter dette ser vi på i *tabell 10* at vi har krysset av for *logikk*. Dette er fordi at elevene må i b)-oppgaven beskrive ei rute den programmerte roboten kan gå for å komme seg tilbake til punkt P. Ved dette må elevene hente frem sine forkunnskaper for å kunne beskrive hvordan dette skal gjøres. Elevene må vite hva som er nord, vest, sør og øst på deres ark. Samtidig når de skal beskrive denne ruten må elevene lage en rekkefølge som får roboten til punkt P. Når elevene jobber med å få roboten til punkt P, går det under *Algoritmer*. Til slutt

må eleven jobbe med *dekomponering* når de skal gjennomføre denne oppgaven fordi de må dele opp problemet når det kommer til hvilke punkter roboten skal gå for å komme seg til P.

Analyseoppgave 5: Utforsk sammen s. 201



Utforsk sammen

Jobb sammen to og to. Bruk ruteark og blyant.
Sitt enten med ryggen mot hverandre eller med en åpen bok som skjerm mellom dere, slik at dere ikke ser hverandres ark.

Elevene tegner hver sin figur på rutearket og lager koden for å tegne den på et annet ark. Elevene bytter kodeark og tegner etter den andres kode. Sammenlign tegningene. Får dere samme figur?

Hvis figurene ikke er like, må dere finne feilen. Er det koden som er feil, eller er det tegningen?

Figur 12: Utforsk sammen (Matematikk 6 fra Cappelen Damm) (Gulbrandsen et al., 2020, s. 201)

Konsepter i Utforsk sammen side 201 i Matematikk 6					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
x	x				x

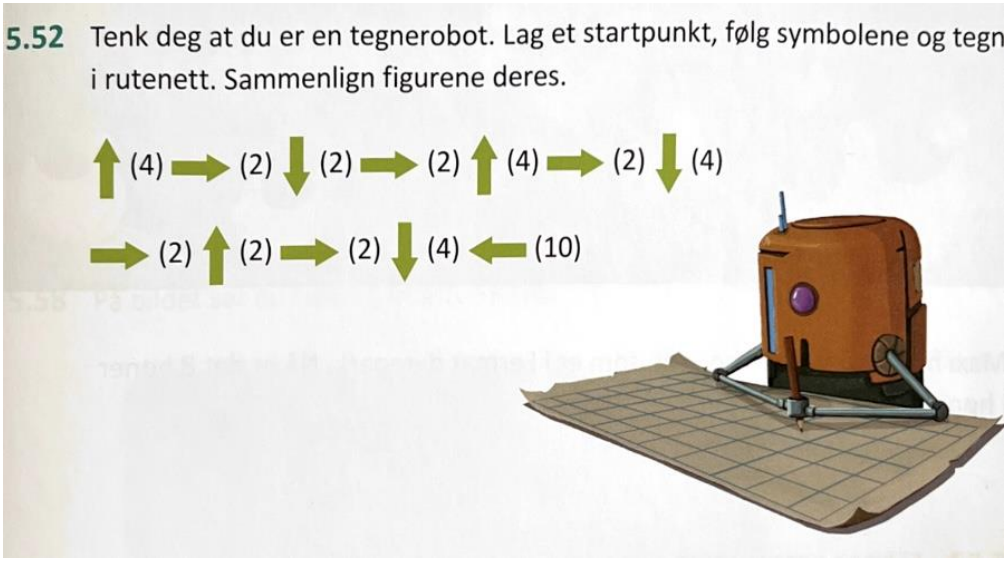
Fremgangsmåter i Utforsk sammen side 201 i Matematikk 6				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
	x	x		x

Tabell 11: Vertikal analyse av utforsk sammen s.201 (Matematikk 6 fra Cappelen Damm)

Ved å begynne helt øverst, med overskriften, blir det tydelig at analyseoppgave 5 er en *samarbeidsoppgave*. Der står det at elevene skal utforske sammen. Like nedenfor kommer første setning som formidler at to elever skal arbeide sammen. Dette er grunnen til at vi valgte å trekke frem denne oppgaven, nettopp fordi den inneholder fremgangsmåten *samarbeid.* Oppgaven de skal utføre er å bruke ruteark og blyant til å hver for seg tegne en figur. Deretter, på et annet ark, skal de skrive en forklaring til hvordan figuren skal tegnes. Videre skal elevene bytte arket med forklaring, slik at de skal gjenskape hverandres figur ved hjelp av den andre elevens kode. Dette gjør denne oppgaven til en analog programmeringsoppgave. Under arbeidet med denne oppgaven får eleven bruk for *logikk* både når de skal *skape* figuren, men også når de skal *skape algoritmen* som den andre eleven skal følge. Denne oppgaven inneholder flere skapelsesprosesser. Først skal de *skape* en figur, deretter en *algoritme* som

produserer figuren. Til slutt skal de *skape* figuren til den andre elevene ved å følge dens *algoritme*. Når dette er gjennomført får elevene en ny oppgave, da skal de sammenlikne figurene og kontrollere om de er like. Dermed skal de *feilsøke*. De blir også direkte bedt om å finne feilen dersom figurene ikke samsvarer. Deretter skal de også gjøre en *evaluering* under *feilsøkingen*. Oppgaven stiller spørsmålet «Er det koden det er noe feil med eller er det tegningen?» Her må elevene ta et valg ut ifra hva de finner ut av, altså en *evaluering* av det de har gjort på bakgrunn av hva de oppdager.

Analyseoppgave 6: Oppgave 5.52 s. 201



Figur 13: 5.52 (Matematikk 6 fra Cappelen Damm) (Gulbrandsen et al., 2020, s. 201)

Konsepter i oppgave 5.52 side 201 i Matematikk 6					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
	x				x

Fremgangsmåter i oppgave 5.52 side 201 i Matematikk 6				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
	x			x

Tabell 12: Vertikal analyse av oppgave 5.52 s.201 (Matematikk 6 fra Cappelen Damm)

Analyseoppgave 6 er en av de få som ikke har *logikk*. Dette er fordi elevene ikke trenger å hente frem forkunnskaper for å løse problemet eller beskrive noe. I oppgaven får eleven all informasjon for å kunne gjennomføre oppgaven og trenger ikke å tenke ut en løsning. De skal

kun følge en allerede gitt algoritme. Elevene må derimot jobbe med en *algoritme.* Koden i denne oppgaven må elevene bruke for å komme frem til hvilken figur som blir tegnet dersom algoritmen følges. Det at de følger denne koden og den *skape*r en figur, gjør at eleven har skapt noe. Eleven skal til slutt i oppgaven *jobbe sammen* med noen andre, der de skal sammenlikne figuren de skapte. Ved å gjøre dette vil elevene *evaluere* oppgaven de har gjennomført, og kan oppleve at andre elever har gjort det helt forskjellig fra dem selv. Dette er en oppgave hvor oppgaveteksten i seg selv ikke nevner begreper innen programmering. Likevel har vi kategorisert den som en programmeringsoppgave grunnet at den er på en side hvor overskriften og temaet er koding. I eksempelet over oppgaven er pilene fra denne oppgaven også avbildet, noe som gjør det enklere for elevene å vite hva de jobber med, selv om begreper innen programmering ikke er nevnt i oppgaveteksten. I tillegg brukes ordet robot som ofte er sterkt assosiert med programmering. Elevene vil i denne oppgaven jobbe med analog programmering på bakgrunn av at de ikke skal anvende noen form for teknologi.

Analyseoppgave 7: Aktivitet s. 88

A Kortspill med vilkår

- Lag et kortspill for to lag, med to personer på hvert lag.
- Bland kortstokken og legg den med tallene ned mellom spillerne.
- I hver runde trekker hvert lag ett eller flere kort.
- På forhånd skal dere lage og skrive ned vilkår som bestemmer om laget får eller mister poeng denne runden.
- Dere må også bestemme et vilkår for når spillet er slutt.

Eksempler på vilkår:

Hvis kortet er rødt
– får laget 3 poeng

Hvis det er et bildekort
– mister laget 1 poeng

Hvis kortet er rødt
– får laget 3 poeng

Hvis ikke
– mister laget 1 poeng

Hvis kortet er kløver
– får laget 1 poeng

Hvis det er spar
– mister laget 1 poeng

Ellers (hvis det er rødt)
– mister laget 1 poeng hvis tallet er over 5

Ellers
– får laget like mange poeng som tallet på kortet

- Spill kortspillet et par ganger. Er det noen vilkår dere vil endre?

Figur 14: Aktivitet (Multi 6B) (Alseth et al., 2021, s. 88)

Konsepter i Aktivitet side 88 i Multi 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
x	x			x	x

Fremgangsmåter i Aktivitet side 88 i Multi 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
	x			x

Tabell 13: Vertikal analyse av Aktivitet s. 88 (Multi 6B fra Gyldendal)

I analyseoppgave 7 skal elevene *samarbeide* om å lage et spill. De skal samtale med hverandre og bli enige om hvilke vilkår som skal gjelde for deres kortspill. Sammen må de hente frem tidligere kunnskap, altså *tenke logisk*, om spill og regler for å kunne *skape* et nytt spill selv. Disse reglene som elevene skaper innen spillet kan også ses på som *algoritmer* ved at det er en rekke med regler som får noe til å skje. Oppgaveformuleringen inneholder mye informasjon som ikke er nødvendig for eleven for å kunne løse oppgaven. Informasjonen vi viser til er alle eksemplene med forslag til vilkår. Dette gjør at elevene blir nødt til å luke ut informasjonen de trenger for å løse oppgaven, slik at det blir enklere for dem å forstå hva de skal gjøre. Altså arbeider de med konseptet *abstraksjon*. Det siste konseptet eleven arbeider med i denne oppgaven er *evaluering*. Dette kan vi se ved at oppgaven ber elevene spille spillet deres to ganger, og deretter finne ut av om det er noen vilkår de vil endre.

4.2.2 Sammendrag Matemagisk 6B

Konsepter i alle oppgavene samlet i Matemagisk 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
20	20	9	16	2	9

Fremgangsmåter i alle oppgavene samlet i Matemagisk 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
13	20	13	2	0

Tabell 14: Vertikal analyse av alle oppgaven i Matemagisk 6B

Ut ifra funnene som ble identifisert i den horisontale analysen var det 20 oppgaver som vi definerte som programmeringsoppgaver. Det som ble observert, var at læreboken til Matemagisk 6B hadde flest programmeringsoppgaver. Etter vi hadde identifisert alle

programmeringsoppgavene, analyserte vi dem etter vårt rammeverk. *Tabell 14* viser hvordan programmeringsoppgavene legger til rette for konsepter og framgangsmåter. Her vil vi forklare tabellen i synkende rekkefølge, der vi starter med det største tallet og ender med det minste. Det vi kan se ved første øyekast er at *logikk, algoritmer* og *skape* gikk igjen i alle oppgavene. Det vil si at i alle oppgaver i Matemagisk 6B må elevene *skape* noe, bruke *logikk* og *algoritmer*. Etter dette er tallene spredt utover i tabellen. Det neste tallet er 16, som vil si at det er 16 oppgaver som inneholder *mønster*, der eleven må se eller *skape* et *mønster*. Noen av tallene gjentar seg i tabellen. Et av disse tallene er 13. Det er 13 oppgaver som anvender *fikle* og *feilsøke*. Deretter har vi ni oppgaver som går igjen to plasser i tabellen. Dette er *evaluering* og *dekomponering*. Det vil si at det er to oppgaver som inneholder *abstraksjon* og *holde ut.* Holde ut er bare krysset av i Matemagisk. Det vi også ser i denne tabellen er at det er ingen *samarbeidsoppgaver* blant programmeringsoppgavene.

4.2.3 Sammendrag Multi 6B

Konsepter i alle oppgavene samlet i Multi 6B					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
13	15	5	5	3	2

Fremgangsmåter i alle oppgavene samlet i Multi 6B				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
3	7	3	0	1

Tabell 15: Vertikal analyse av alle oppgaven i Multi 6B

Tabell 15 viser en oversikt over hvilke konsepter og fremgangsmåter elevene arbeider med i alle programmeringsoppgavene i Multi 6B. Oppgavene som ble trukket frem og utdypet i tidligere delkapitler er også talt med i denne tabellen. Konseptene det arbeides desidert mest med er *logikk* og *algoritmer*. Det er 13 oppgaver hvor elevene arbeider med *logikk* og hele 15 oppgaver hvor elevene arbeider med *algoritmer*. Det vil si at alle programmeringsoppgavene i Multi 6B innebærer at elevene skal arbeide med *algoritmer*. Ellers i tabellen er det noe mer spredning. Det er for eksempel kun én oppgave som innebærer at elevene skal *samarbeide*. I ruta for *holde ut* har det ikke blitt satt noen kryss. Dette er fordi det er vanskelig å se i en oppgavetekst. Omtrent den eneste måten om vi skulle kunne si noe om dette er om vi hadde observert elevene arbeide med oppgavene. Videre i tabellen kan vi se at det er like mange

oppgaver som inneholder *abstraksjon, fikling* og *feilsøking*, altså tre stykker. Antallet oppgaver som inneholder *mønster* og *dekomponering* er fem stykker. Dette tilsier at en tredjedel av oppgavene inneholder arbeid med disse konseptene innenfor algoritmisk tenking. De to siste komponentene er *skape* hvor det er syv oppgaver som ber eleven om å *skape* noe, og *evaluering* hvor det er to oppgaver som ber elevene om dette.

Ved å se på tabellen blir det tydelig at enkelte av konseptene og fremgangsmåtene er mer fremtredende i oppgavene i boka. Likevel har oppgavene i boka vært innom alle konseptene og fremgangsmåtene tilhørende modellen, foruten holde ut.

4.2.4 Sammendrag Matematikk 6 fra Cappelen Damm

Konsepter i alle oppgavene samlet i Matematikk 6					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
2	5	2	1	0	2

Fremgangsmåter i alle oppgavene samlet i Matematikk 6				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
0	4	3	0	2

Tabell 16: Vertikal analyse av alle oppgaven i Matematikk 6 fra Cappelen Damm

I denne læreboka er det fem oppgaver som vi i den horisontale analysen kategoriserte som programmeringsoppgaver. Selv om det ikke er så mange oppgaver kan vi se at oppgavene til sammen likevel har lagt til rette for arbeid med omtrent alle konseptene og fremgangsmåtene innenfor algoritmisk tenking. Alle oppgavene inneholder *algoritmer* og alle utenom en innebærer at elevene skal *skape* noe. Over halvparten av oppgavene inneholdt at elevene skulle *feilsøke*. Det er like mange oppgaver som innebærer at elevene skal arbeide med *logikk, evaluering* og *samarbeid*. Av alle bøkene vi har analysert var dette boka som inneholdt flest oppgaver som involverte at elevene skulle *samarbeide*. Det siste konseptet som arbeides med i denne boka er *mønster*. Av fem oppgaver er det en oppgave som innebærer at elevene skal arbeide med *mønstre.* Det er elleve konsepter og fremgangsmåter totalt, hvor elevene arbeider med åtte av dem. De tre som ikke kommer frem i løpet av arbeidet med de fem programmeringsoppgavene er *abstraksjon, fikle* og *holde ut.*

4.2.5 Sammendrag av alle oppgavene i alle bøkene

Konsepter i alle oppgavene i alle bøkene samlet					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
35	40	16	22	5	13

Fremgangsmåter i alle oppgavene i alle bøkene samlet				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
16	31	19	2	3

Tabell 17: Vertikal analyse av alle oppgavene i alle bøkene

I den siste tabellen har vi samlet alle funnene våre på en plass. Dette gjorde vi fordi vi ønsker å se hvordan alle oppgavene om programmering i alle bøkene la til rette for algoritmisk tenking. Det vi så med å gjøre dette var at alle konseptene og fremgangsmåtene i Barefoot Computing´s (u.å.-a) modell er krysset av i tabellen. Vi kan også se at alle bøkene til sammen har 40 programmeringsoppgaver. En annen ting som legges merke til er at fordelingen av kryss ikke er helt jevn. Fremgangsmåten *holde ut* er den vi har sett minst av når vi har gjort analysen og deretter kommer *samarbeid* med tre kryss. Det konseptet vi så minst av er *abstraksjon* i programmeringsoppgavene med fem kryss. De nevnte konseptene og fremgangsmåtene har blitt krysset av under ti ganger. De som har fått mindre enn 20 kryss er *evaluering* med 13 og deretter er det *fikle* og *dekomponering* med 16 kryss. Til slutt kommer f*eilsøke* med 19 kryss. Dette vil si at de fleste bøkene hadde disse konseptene og fremgangsmåter i programmeringsoppgavene sine.

Etter dette ser vi at det er bare et konsept som har færre enn 30 kryss, som er *mønster* med 22 kryss. Helt til slutt er det fremgangsmåtene og konseptene som har mindre enn 40, eller 40 kryss. Konseptet *algoritme* er det som vi finner i hver programmeringsoppgave. Dette vil si at hver eneste programmeringsoppgave legger til rette for at eleven må jobbe med en rekke regler som gjør at elevene kan gjennomføre oppgavene. Et annet konsept vi ser er brukt i mange oppgaver er *logikk*. Der elevene må hente fram forkunnskap for å finne ut hvordan de skal løse ellers beskrive oppgavene. Vi ser at det er hele 35 av 40 oppgaver som inneholder *logikk*. Til slutt er det fremgangsmåten *skape*, som innebærer at elevene skal lage noe når de jobber med oppgaven. Ved å ha betraktet de ulike programmeringsoppgavene i lærebøkene kommer det til syne at *skape* ofte foregår enten ved at de skal arbeide i et program eller ved å

tegne/skrive for hånd. Det vi videre har gjort funn om er at det er 31 oppgaver som legger opp til at elevene skal *skape* noe når de gjennomfører programmeringsoppgaven.

5 Drøfting

I drøftingen presenteres funnene fra kapittel 4 opp mot teorien fra kapitel 2 Teorigrunnlaget. Vårt overordnede fokus var å se etter hvordan programmeringsoppgavene i de ulike lærebøkene la til rette for algoritmisk tenkning. Noen faktorer i analysen vil ikke bli trukket frem da funnene er beskrivende i seg selv. Vi har delt dette kapittelet inn i ulike delkapittel. Det første delkapittelet vil ta for seg funnene vi fant ved å gjennomføre den horisontale analysen. Der vi diskuterer mengden programmeringsoppgaver og plasseringen av dem opp mot LK20 og tidligere forskning om lærebøker. Neste delkapittel omhandler forekomsten av de ulike komponentene av algoritmisk tenking i programmeringsoppgavene, diskutert opp om tidligere forskning. Delkapittel 5.3 dreier seg om programmeringstyper i lærebøkene i lys av algoritmisk tenking.

5.1 Programmeringsoppgaver i lærebøker

For å kunne se på hvordan programmeringsoppgavene kan brukes til å legge til rette for arbeid med algoritmisk tenking, er det nyttig å først se på enkelte aspekter av hvilken posisjon læreboken og programmering har i skolen. På det grunnlaget starter drøftingskapittelet med å presentere et innblikk i programmeringsoppgaver i lærebøker.

5.1.1 Antall programmeringsoppgaver

Den horisontale analysen gikk ut på å kartlegge antall programmeringsoppgaver hver av de tre bøkene inneholdt. Etter at vi hadde analysert alle programmeringsoppgavene vi fant i de ulike lærebøkene, oppdaget vi at det var en varierende mengde med oppgaver. En av bøkene inneholdt fem programmeringsoppgaver, hvilket kan forklares via en e-post utveksling med forlaget. Da vi hadde bestemt oss for hvilke lærebøker vi ønsket å bruke for å hente inn data, sendte vi forlagene en e-post og stilte de spørsmål. Det vi lurte på var hva slags bakgrunn de som har laget programmeringsinnholdet i lærebøkene har. Av Cappelen Damm fikk vi et utfyllende svar hvor de forteller om bakgrunnen til forfatterne av programmeringsinnholdet. De legger også til at de har lagt deres programmeringsinnhold til den digitale ressursen deres på grunn av at det er et felt som utvikler seg raskt og at det er lettere å endre og forbedre innholdet der istedenfor i bøkene (personlig kommunikasjon, 21. mars 2023). Dette vil også være en begrunnelse for at det fantes få oppgaver som nevnte begreper innenfor programmering i deres bok. Utfallet av dette kan føre til at det er færre oppgaver som kan

legge til rette for algoritmisk tenkning. Dette funnet kan også ses i sammenheng med det Skjelbred et al. (2017, s. 24) legger frem om at det nå har blitt vanlig at et læreverk består av flere komponenter en bare læreboken. Pepin et al. (2013. s. 929) trekker også frem dette. Flere ressurser i et læreverk kan ha både fordeler og ulemper. Det kan åpne opp for variasjon i skolehverdagen fordi læreren får flere ulike ressurser å støtte seg til og bruke. Et eksempel er hvis skolen ikke har kjøpt nettressursen, men bare den trykte læreboken. Da kan de elevene på denne skolen ende opp med å arbeide med færre programmeringsoppgaver enn elever på andre skoler. Dette er da satt på spissen om det ikke blir hentet inn eksterne ressurser utenom læreboken.

Et annet aspekt på det ovennevnte poenget kan være, som Brehmer et al. (2015, s. 589) påpeker, at elevens utbytte kan påvirkes av hvor innholdet er plassert i læreboken. Med dette tolker vi at elevens utbytte av å jobbe med oppgaver vil variere etter hvordan oppgave de jobber med og hva de rekker i løp av en skoletime. Ved å analysere de ulike lærebøkene så vi at de hadde utformet lærebøkene på ulike måte. En av bøkene hadde lagt opp oppgavene etter vanskelighetsgrad. En annen faktor som kan spille inn på elevens utbytte av undervisningen er at det i en TIMMS – undersøkelse fra 2011 viste seg at lærere brukte matematikkboka som hovedgrunnlaget for undervisningen i matematikk (Mullis et al. 2012, s. 391).

Lærebøkene inneholdt mellom en og seks prosent programmeringsoppgaver om disse betraktes i forhold til alle oppgavene i boka. Vektleggingen i lærebøkene kan ha noe å gjøre med læreplanverket. I læreplanen for matematikk på 6. trinn er det to av ti, altså 20%, av kompetansemålene som kan knyttes til programmering ved hjelp av bruken av begreper (Kunnskapsdepartementet, 2019b). De relevante målene er:

- «Bruke variablar og formlar til å uttrykkje samanhengar i praktiske situasjonar.»
- «Bruke variablar, lykkjer, vilkår og funksjonar i programmering til å utforske geometriske figurar og mønster» (Kunnskapsdepartementet, 2019b, s. 10).

Disse to målene inneholder også annet innhold som elevene skal lære, hvilket betyr at målet ikke ene og alene er om programmering. Det kan dermed se ut til at det er lagt opp til at elevene skal få en opplæring i matematikk og programmering sammen. En fordel med at programmering er lagt inn i læreplanen for matematikk, er at det er god tilgang til og variasjon i aktiviteter og oppgaver som er didaktiske og kan kobles opp mot algoritmisk tenking (Barcelos et al., 2018, 832-833).

Mengden oppgaver og vektleggingen av programmering i læreplanen kan virke overensstemmende. Selv om 20% av kompetansemålene inneholder programmering, og 1-6% av oppgavene i lærebøkene omhandler programmering, kan det likevel virke overensstemmende på bakgrunn av at målene inneholder mer enn bare programmering. Dermed kan det ses på som at det ikke er fulle 20% av målene som inneholder programmering. Når det gjelder mengde av programmeringsoppgaver stiller Stigberg og Stigberg (2020, s. 491) spørsmål ved hvor mye programmering som er tilfredsstillende mengde sett i lys av oppnåelse av kompetanse. Altså dette som omhandler hvor mye programmering som er forventet at skal gjennomføres.

5.1.2 Plassering av programmeringsoppgavene

Andre aspekter ved læreplanen det kan være nyttig å se på er hvilke kapitler oppgavene om programmering befinner seg. Tidligere forskning viser til at det som har blitt forsket mest på innen programmering og matematikk er geometri, men et annet felt det også har blitt forsket på er tall og algebra (Lv et al., 2022, s. 17). Majoriteten av oppgavene i to av bøkene vi har analysert befinner seg i kapitler hvor hovedtemaet er algebra. Dette er et interessant valg med tanke på at kompetansemålet retter seg mot geometriske mønstre. Samtidig observerte vi ved hjelp av den horisontale analysen at selv om tittelen på kapittelet om handler algebra, kan innholdet fortsatt omhandle noe geometri.

Det er enighet at lærebøkene påvirker hvilke oppgaver elevene arbeider med (Johansson, 2006, s. 26). At plasseringen av oppgaver og formuleringen av kompetansemål ikke samsvarer kan føre til spenninger som Johansson (2006, s. 26) kalte det. I denne sammenhengen kan det oppstå en spenning mellom en lærer som følger læreboken slavisk og staten som har bestemt hva som skal gjøres gjennom styringsdokumenter. Dette kan påvirke elevenes utbytte av programmering og dermed også arbeidet med algoritmisk tenking. Ved å se nærmere på dette kan det se ut til at det er flere utfordringer. En utfordring kan være, som forsking fra 2018 tilsier, at flesteparten av lærerne i et forskningsprosjekt sa selv, at undervisningen i stor grad ble styrt av læreboken (Van den Ham & Heinze, 2018, 136). At programmeringsoppgavene befinner seg i de ovennevnte kapitlene kan ha noe å gjøre med at deler av programmering kan oppleves som enklere å knytte til algebra, enn geometri og mønstre.

5.1.3 Betydningen av ressurser i forhold til programmeringsoppgaver

Hvilke deler av læreverket som blir kjøpt inn kan avhenge av ulike faktorer. Dermed kan det se ut til at det er stor sannsynlighet for at de ulike skolene i de forskjellige kommunene, kjøper inn ulike ressurser. Enkelte skoler kan ha tilgang på mer ressurser enn andre skoler. Om en skole har kjøpt inn hele læreverket kan det legge til rette for at elevene får arbeide mer med programmering, enn elever på en annen skole som kjøper inn kun læreboken. Dette er et eksempel på at det ikke kun er oppgavene i læreboken som har noe å si for elevenes læringsutbytte. Malik et al. (2019, s. 85) argumenterer for at gode valg av blant annet læringsmateriale spiller en viktig rolle for å skape en suksessfylt læringsprosess.

Det at ulike skoler har ulike ressurser å arbeide ut ifra, har ikke nødvendigvis en stor betydning for elevenes møte med algoritmisk tenking gjennom programmeringsoppgaver. Et funn vi gjorde i analysen var også at mengden programmeringsoppgaver nødvendigvis ikke hadde så mye å si for tilretteleggingen for arbeid med algoritmisk tenking. En forklaring på dette kan være at uavhengig av antallet oppgaver bøkene inneholdt, oppdaget vi at oppgavene la til rette for arbeid med de fleste av konseptene og fremgangsmåtene innen algoritmisk tenking. Et eksempel verdt å trekke frem kan være Matematikk 6 fra Cappelen Damm som inneholder fem programmeringsoppgaver, og likevel har elevene i løpet av de fem oppgavene vært innom alle konsepter og fremgangsmåter, bortsett fra tre. Dermed ser vi at mengden oppgaver ikke nødvendigvis har noe å si for hvor mange komponenter av algoritmisk tenking elevene arbeider med. Det kan se ut til at dette samsvarer med det Bai et al. (2021, s. 3) legger frem om at programmering er tett relatert til og egner seg som undervisningsform for å utvikle ferdigheter innen algoritmisk tenking. Det samme skriver Grover og Pea (2013, s. 39) i deres artikkel. Selv om det var mange konsepter og fremgangsmåter som ble dekket av oppgavene var det enkelte som var mer fremtredende enn andre.

5.2 Forekomsten av de ulike konseptene og fremgangsmåtene

Konsepter i alle oppgavene i alle bøkene samlet					
Logikk	Algoritme	Dekomponering	Mønster	Abstraksjon	Evaluering
35	40	16	22	5	13

Fremgangsmåter i alle oppgavene i alle bøkene samlet				
Fikle	Skape	Feilsøke	Holde ut	Samarbeid
16	31	19	2	3

Tabell 18: Gjengivelse av tabell 17

For å gjøre det mer oversiktlig for leseren, har vi valgt å trekke ned tabellen som viser alle konseptene og fremgangsmåtene som ble funnet i vår analyse.

5.2.1 Logikk og algoritmer

I vår gjennomgang av analysen ser vi at det er en fellesnevner i alle lærebøkene. Denne fellesnevneren var at alle programmeringsoppgavene hadde konseptet *algoritme* i seg. I følge Barefoot Computing (u.å.-a) er konseptet algoritme en rekkefølge av instruksjoner som eleven skal følge, og den skal gjøre at et problem blir løst på best mulig måte. I forskningsartikkelen til Stigberg og Stigberg (2020, s. 491) skrev de at en lærer nevnte at programmering og matematikk har to fellesnevnere. Disse fellesnevnere var algoritme og struktur. Når deres poeng ses opp mot det Barefoot Computing's (u.å.-d) definisjon på hva algoritme er, kan det se ut til at elevene må strukturere seg for å finne den mest hensiktsmessige måten å løse oppgaven på. Dette skriver Stigberg og Stigberg (2020, 491) er viktig under programmering. Dette kan være en faktor for at vi i analysen så at alle programmeringsoppgavene i lærebøkene innholt konseptet algoritme. Videre ble det skrevet at når elevene får kunnskap innenfor programmering kan det gjøre at elevene ser lettere matematiske strukturer i oppgaver (Stigberg og Stigberg, 2020, 491).

5.2.2 Dekomponering og abstraksjon

Ved å se på *Tabell 18* med oversikt over alle oppgavene som inneholder de ulike konseptene og fremgangsmåtene kommer det tydelig frem at enkelte deler av modellen er mer fremtredende enn andre. Konsepter som var relativt lite fremtredende, var abstraksjon og dekomponering. Dekomponering handler som tidligere forklart, om å dele opp et problem i

mindre deler slik at det blir lettere å løse problemet (Barefoot Computing, u.å.-e). Abstraksjon handler om at elevene skal luke bort informasjon som ikke er nødvendig for å skape oversikt over problemet (Barefoot Computing, u.å.-g). Det er ikke sikkert at mengden abstraksjon og dekomponering har relasjon til programmering i seg selv. Med dette mener vi at det kan se ut til at det kan ha noe å gjøre med selve oppbygningen av matematikkbøker. For å utdype hva som menes med dette ser vi på det som står i IDM (2007, sitert i Gracin & Krišto, 2022, s. 100) om at oppgavene i matematikkbøker er skapt med ulikt kognitivt nivå. De kommer frem til at oppgavebøker i matematikk inneholder desidert flest oppgaver som er lite komplekse (Gracin & Krišto, 2022, s. 108). Dette kan ses på som at når oppgavene er laget lite komplekse og på lavt kognitivt nivå, vil de inneholde lite tekst og en kort problemstilling. Ut ifra dette har vi oppdaget at dette stemmer med lærebøkene vi har analysert.

Et eksempel er oppgave 31 i Matematikk 6 fra Cappelen Damm, der de skriver «Tenk deg at du er en tegnerobot. Lag et startpunkt, følg symbolene og tegn i rutenett. Sammenlign figurene deres» (Gulbrandsen et al., 2020, s. 201). Denne programmeringsoppgaven gir ikke elevene mer informasjon en nødvendig, som vil si at det er lite tekst. Wing (2006, s.33) skriver at dekomponering og abstraksjon blir brukt når elevene skal starte på en oppgave som et stort omfang og er vanskelig. Videre fra dette kan det trekkes tråder til at det ikke vil være behov for eleven å dele opp problemet eller luke bort informasjon, derav lite arbeid med konseptene dekomponering og abstraksjon.

Andre aspekter ved oppgaveformulering som er verd å trekke frem baserer seg på Bråting og Kilhamn's (2022, s. 605) ytring om at typiske oppgaver i matematikkbøker har en formulering som er steg-for-steg. Et eksempel på dette kan være at oppgavene i en matematikkbok ofte er delt opp i underoppgaver, som eksempelvis a), b), og c). Når oppgavene er delt inn i deloppgaver på denne måten, kan det ses på som en slags dekomponering av selve problemet. Dette kan vi se i for eksempel oppgave Analyseoppgave 4. Likevel er dette forskjellig fra oppgave til oppgave, og kan ikke generaliseres for alle oppgaver. Enkelte oppgaver kan være komplekse selv om de er delt opp, imens andre ikke. Det at oppgaveteksten ofte er formulert kort og på et kognitivt lavt nivå kan også ha noe med trinnet boka vi har analysert tilhører, eller alderen til elevene. Sett i lys av Zhang og Nouri's (2019, s. 18) funn om at hvor vanskelig det er å implementere komponenter av algoritmisk tenking gjennom programmering avhenger av kognitiv utvikling. Om det er alderstrinnet vi har forsket på som gjør at det er spesielt lite bruk av konseptene abstraksjon og

dekomponering, eller om det er at programmering ikke legger til rette for dette, er noe vi stiller oss undrende til. På en annen side stemmer ikke det funnet vi har gjort overens med Lv et al.'s (2022, s. 11) uttalelser. De peker på at der hvor algoritmisk tenking var integrert i oppgavene de forsket på, var det abstraksjon og mønstergjenkjenning som ble hyppigst arbeidet med (Lv et al., 2022, s. 11).

5.2.3 Samarbeid

Som nevnt tidligere er problemløsning en del av det som kalles for «21st century skills» (Ananiadou & Claro, 2009, s. 9). Ananiadou & Claro (2009, s. 10) beskriver også kommunikasjon som en viktig del av dette settet med ferdigheter. De utdyper at samarbeid er viktig for å kunne fungere som en fullverdig borger i samfunnet, både sosialt og digitalt (Ananiadou & Claro, 2009, s. 10). Ved å se problemløsning, altså for eksempel algoritmisk tenking, og samarbeid i sammenheng, kommer det frem i Barefoot Computing´s (u.å.-m) modell at samarbeid er en av de fem fremgangsmåtene. Det er flere forskere, blant annet Lodi og Martini (2021, s. 896), som peker på at samarbeid er en viktig egenskap når det kommer til problemløsning og algoritmisk tenking. Likevel ved å se på programmeringsoppgavene i lærebøkene er det et minimum av oppgaver som omhandler samarbeid. Kun tre programmeringsoppgaver i alle lærebøkene til sammen innebar at elevene skulle samarbeide, en av dem er beskrevet tidligere i forbindelse med fremgangsmåten fikling innen analog programmering. Hvorfor det er lite av samarbeidsoppgaver kan ha noe med masteroppgavens begrensninger å gjøre. Det kan hende at om vi i tillegg til å analysere lærebøkene hadde analysert lærerveiledningene, kunne vi funnet instruksjoner som hadde åpnet opp for flere samarbeidsoppgaver. På bakgrunn av dette kan oppgavens begrensing ha ført til at vi ser liten forekomst av samarbeidsoppgaver.

5.2.4 Mønster

Konseptet mønster har vært middels fremtredende i oppgavene som har vært dataen for denne masteroppgaven. Av 40 oppgaver var det 22 stykker som innebar at elevene skulle arbeide med mønster. Altså se, lete etter eller skape noe som gjentar seg (Barefoot Computing, u.å.-f). Over halvparten av programmeringsoppgavene handlet om mønster, noe som kan ha noe med at kompetansemålet som omhandler programmering også omhandler geometriske mønstre (Kunnskapsdepartementet, 2019, s. 10). Hvilket peker at det er lagt opp til at når elevene arbeider med programmering, skal de også arbeide med mønstre. Funnene rundt oppgaver som inneholder konseptet mønster stemmer også mer overens med de ovennevnte uttalelsene

til Lv et al. (2022, s. 11) om at mønster er mye brukt når det gjelder algoritmisk tenking i oppgaver. Videre kan det legges til at Wing (2006, s. 34) påpeker at det å være på jakt etter mønstre er en del av det å arbeide med algoritmisk tenking.

5.2.5 Feilsøking

Det var flere konsepter og fremgangsmåter vi la merke til. Et av dem var feilsøking, hvilket var å finne i 19 av programmeringsoppgavene. Feilsøking omhandler det å lete gjennom koden for å finne ut hvorfor resultatet ikke er som forventet, for så å rette opp i feilen (Barefoot Computing, u.å.-k). Ved å se på programmeringsoppgavene i de ulike lærebøkene, ser vi at de aldri nevner at elevene skal anvende programmeringsprogrammet Scratch, men ved å se på det visuelle kan det tyde på at eleven skal bruke Scratch. Hadde vi sett på lærerveiledningene kan det hende at de henviser oss til dette programmeringsprogrammet. I dette forskningsprosjektet har vi ikke undersøkt lærerveiledninger, og kan derfor ikke uttale oss om det. For å få frem poenget om hvordan programmering kan legge til rette for feilsøking er det nødvendig å se tilbake på hvor programmet kommer fra og dets egenskaper i dag. I delkapittel 2.1 Programmering i matematikk, nevner vi at Bocconi et al. (2016, s. 39) peker på at de viktigste konseptene i Papert's design, fortsatt blir brukt i andre programmer i dag. Det kan se ut til at Scratch har tatt noen konsepter fra Papert´s design, der de har en katt som flytter seg når den blir gitt en kommando. Noe som ligner på det Papert (1980, s. 11) skriver i sin bok at «skilpadden» gjør. Dataprogrammet Scratch er også rettet mot elever, men da i hovedsak for yngre elever og deres utdanning (Lee, 2011, s. 27). En av grunnene til at Scratch er knyttet mer mot yngre elever enn Logo – programmet, kan være at de ikke er nødt til å skrive inn en kommando ved hjelp av tastaturet for å få katten til å gjøre det de instruerer den til. Scratch er et puslespill-lignende program hvor de som benytter seg av det setter sammen visuelle blokker for å få katten til å gjøre det de vil (Lee, 2011, s. 27). Ved dette anvender de ikke tastatur, men de bruker heller datamusen for å sette sammen koden.

Uavhengig om det var Scratch eller ikke, var det tydelig at oppgavene i lærebøkene var basert på et blokkbasert program slik som Scratch blir beskrevet som over. En av fordelene ved å programmere i Scratch er at elevene unngår syntaksfeil på grunn av at blokkene som ikke fungerer sammen ikke vil feste seg (Lee, 2011, s. 27). Dette kan føre til at feilsøkingen til elevene kan gå lettere, fordi det er mindre komplisert å lete etter feilen, om det skulle oppstå et resultat som ikke er ønskelig. Samtidig kan det betraktes fra et annet perspektiv hvor

elevene ikke får like god kjennskap til det å feilsøke ved å bruke Scratch på bakgrunn av at blokkene fester seg sammen om de ikke fungerer sammen.

Som nevnt tidligere har Zhang og Nouri (2019, s. 18) uttalt seg om at det kan være enkelte komponenter av algoritmisk tenking som kan være vanskelig å tilegne seg gjennom arbeid med Scratch (les blokkbasert program). Feilsøking kan være en av disse komponentene da programmet kan ses på som behjelpelig med dette, som nevnt i avsnittet over. Samtidig viste oppgaveteksten i mange av programmeringsoppgavene til at de ønsket et spesifikt svar. For eksempel at elevene skulle bruke et program til å skape en figur som er identisk til den som er avbildet i boka. Dermed blir eleven nødt til først å sjekke om resultatet deres tilfredsstiller kravene som oppgaven har stilt. Om de da oppdager en feil, at figuren ikke er helt lik, blir de nødt til å lete i koden de har kommet frem til for å finne ut hva som må endres for å få rett resultat. Her er det en fordel at Scratch er et visuelt program, som gjør dette enklere for elevene å finne feilen (Lee, 2011, s. 27). Et annen positiv faktor er at elevene synes det er enklere å programmere når det de jobber med noe som er visuelt (Kjällander et al., 2021, s.3).

5.2.6 Feilsøking og logikk

Feilsøking er en fremgangsmåte som henger tett sammen med konseptet logikk (Barefoot Computing, u.å.-c). Elevene vil oppdage at for å lete fram til hvor feilen i koden er, vil det være nyttig å bruke logisk tenking. Ved å se på analyseresultatene av programmeringsoppgavene, stemte dette overens med våre resultater også. Vi observerte at omtrent alltid når det var krysset av for feilsøking, var det også krysset av for logikk. Sun et al. (2021, s. 12) viser til resultater hvor elever med programmeringserfaring ble drastisk mye bedre til å tenke logisk. I samme forskning fant de også ut at programmeringserfaringen kan være med på å gi elevene en bedre forståelse for hva algoritmisk tenking er (Sun et al., 2021, s. 13).

Gjennom arbeidet med analysen ble det også tydelig at logikk var et av konseptene som var mest fremtredende i programmeringsoppgavene i lærebøkene. De aller fleste oppgavene innebar at elevene var nødt til å bruke logisk tenking for å finne en løsning. Vi observerte at det var hele 35 av 40 programmeringsoppgaver i alle lærebøkene som omhandlet det at elevene måtte hente fram forkunnskaper for å lage en løsning eller beskrive hvorfor noe skjer (Barefoot Computing, u.å.-c). Dette kan ha noe med at elevene ofte i programmeringsoppgaver blir bedt om å skape noe, og dermed må hente frem tidligere

kunnskap for å løse problemet. Andre aspekter ved programmeringsoppgavene som krever at elevene må bruke logisk tenking er som nevnt over, feilsøking. Begge de ovennevnte eksemplene på hvordan logikk implementeres i en programmeringsoppgave kan det sees på som at det omhandler arbeid med algoritmer. For om elevene blir bedt om å skape noe i en programmeringsoppgave, kan det se ut til at de som regel er nødt til å lage en algoritme. Om de skal feilsøke er det algoritmen de må begynne å lete i. Algoritmen kan også ses på som en form for kode, ifølge hvordan disse begrepene er definert i denne oppgaven. Ifølge Lee (2020, s. 266) krever det blant annet logisk tenking av elevene når de skal kode. Dette kan dermed ses i sammenheng med og støttes opp av studien til Malik et al. (2019, s. 91) hvor de kommer frem til at arbeid med algoritmer fører til at logikken flyter bedre. Deres funn kan dermed være en forklaring på hvorfor så mange av oppgavene inneholdt konseptet logikk innenfor algoritmisk tenkning.

5.2.7 Evaluering

En annen del av algoritmisk tenking er evaluering. Ifølge Barefoot Computing (u.å.-h) kan det omhandle å ta beslutninger på en objektiv og systematisk måte. Videre skriver de at dette er viktig for å vurdere om kvaliteten, effektiviteten og løsningen til produktet er godt nok (Barefoot Computing, u.å.-h). Dersom vi ser dette opp mot oppgavene analysert i den vertikale analysen ser vi at elevene også blir bedt om å evaluere hvilke programmer som er mest optimale å benytte for å gjennomføre en gitt oppgave. Wing (2006, s. 33) trekker frem at algoritmisk tenking ikke bare omhandler om programmet er korrekt og effektivt, men også om det er hensiktsmessig. Ut ifra dette så vi etter oppgaver som ber elevene ta valg, og bedømme det de selv har gjort i oppgaven. Ved å gjøre dette fant vi 13 oppgaver som inneholdt konseptet evaluering. I analyseoppgave 3 fremkommer det at elevene skal velge ut hvilke program de ønsker å anvende, for å kunne lage et mønster til et spillbrett eller en spillebrikke. I denne programmeringsoppgaven vil eleven måtte ta et valg om de ønsker å gjennomføre den ved å bruke analog- eller blokkprogrammering. Da må eleven tenke på hva som gjør denne oppgaven best mulig ifølge oppgavens kriterier. Er det å lage et mønster inne i Scratch eller er det å lage den på papir? En annen faktor er om de skal lage det til et spillbrett eller en spillebrikke. I analyseoppgave 5 fikk elevene instruksjon om at hver av de skulle lage en figur, samt en kode/fremgangsmåten for å lage figuren. Deretter skulle de bytte kodeark og lage den andres figur. Til slutt fikk de beskjed om å «sammenligne figurene deres». Når elevene skal sammenligne må de evaluere først om figurene produsert av samme

kode er like. Deretter, om figurene er ulike, må de gjøre en feilsøking og evaluere om det er koden eller tegningen det er noe feil med.

5.2.8 Holde ut

Avslutningsvis i dette delkapittelet vil vi trekke frem den komponenten av algoritmisk tenking vi så færrest ganger i programmeringsoppgavene, altså holde ut. Tidligere i denne masteroppgaven beskrev vi at i analysen av oppgavene, ble kun det som står i oppgaven tatt i betraktning når vi analyserte etter rammeverket. Fremgangsmåten holde ut handler om at eleven må fortsette å prøve helt til hen har fått riktig svar (Barefoot Computing, u.å.-l). Dermed er dette en komponent av algoritmisk tenking som er vanskelig å se i dette prosjektet. Likevel kom det frem i to programmeringsoppgaver i den ene læreboken. For å argumentere for at vi skulle kunne si at denne oppgaven ba elevene om å holde ut, må vi se nærmere på en liten figur på siden av oppgavene. Den avbildede figuren kommer med en kommentar som lyder: «Kjør programmene underveis for å se om du er på rett vei. Når vi programmerer, er et viktig å prøve og feile» (Matemagisk, 2020, s. 111). Det er også nødvendig å legge til at oppgavene gikk ut på at elevene skulle lage et program som gjenskaper figurer som er avbildet på siden. Figuren formidler altså til elevene at de må fortsette å prøve selv om de feiler. Dette stemmer igjen overens med det å holde ut i arbeidet med en oppgave. Samtidig er dette en fremgangsmåte som det ville vært enklere å si noe om ved å gjennomføre en annen studie som for eksempel innebærer å observere elevene imens de programmerer.

5.3 Programmeringstyper i lærebøkene i lys av algoritmisk tenking

5.3.1 Blokkprogrammering

Blokkprogrammering er ifølge Lv et al. (2022, s. 14) det mest kjente programmeringsverktøyet som brukes. Et program som er blokkbasert har ulike blokker som settes sammen for å lage en visuell fremstilling (Mannila, 2017, s. 71). Denne forklaringen på hva et blokkprogram er samsvarer med det som er nevnt over om Scratch. Hvilket vil si at Scratch kan være et blokkprogram. De ulike blokkene har ulike former og farger etter hvordan funksjon de ulike blokkene har, med det skal også gjøre dette lettere å visualisere hvilke blokker som kan bygges opp hverandre (Mannila, 2017, s. 71). Lv et al. (2022, s.14) skrev at de så gjennom å jobbe med visuelle kodemiljøer vil elevene forbedre blant annet problemløsningsferdigheter. Algoritmisk tenking er ifølge Lodi (2020, s. 128) et eksempel på

en spesifikk problemløsning, og dermed kan det trekkes tråder til at visuelle kodemiljøer kan bidra til å forbedre elevenes algoritmiske tenking. Andre fordeler ved at lærebøkene anvender blokkprogrammering kommer frem i forskningen til Lv et al. (2022, s. 14), hvor de trekker frem at for å kombinere matematikk og algoritmisk tenking er blokkprogrammering et godt miljø.

5.3.2 Analog programmering

Videre når vi ser på de ulike programmeringsoppgavene ser vi at elevene også skal jobbe med programmering uten å bruke et elektronisk verktøy. Dette stemmer igjen med definisjonen til Berg (2021, s. 42) når det kommer til analog programmering. Elevene har i dette tilfellet jobbet med analoge programmeringsoppgaver. De analoge programmeringsoppgavene identifiserer vi som formulert og konstruert noe ulikt i lærebøkene. Noen av oppgavene sier at elevene skal følge en gitt kode ved å tegne det. Etter dette skal de se hva de har skapt. Et eksempel på en oppgave som ber elevene å tegne er en «Utforsk sammen» oppgave fra Matematikk 6 fra Cappelen Damm. Dette er analyseoppgave 5, der elevene får beskjed om å tegne en figur på et ruteark. I følge Lv et al. (2022, s. 14) kan ruteark være et nyttig verktøy i matematikkundervisningen når det kommer til integreringen av algoritmisk tenkning. Etter elevene har tegnet figuren skal de skrive ned koden til figuren, deretter gi kodearket til den andre eleven, som skal følge koden. I denne oppgaven ser vi at elevene må utføre en steg-for-steg-metode, som ifølge Lee og Junoh (2019, s. 710) kan ses på som analog koding. Elevene jobber med analog programmering mer enn vi vet i hverdagen. Dermed er det viktig at lærere tar dette med inn i undervisningen (Lee & Junoh, 2019, s. 710).

I en forskningsartikkel nevnt i kapittel 2.4 Programmering og algoritmisk tenkning, kunne elevenes ferdigheter innen algoritmisk tenking bli bedre ved å jobbe med analoge programmeringsoppgaver (Sun et al., 2021, s. 13). Ved å se på de analoge programmeringsoppgavene som har blitt analysert i denne masteren, tyder det på at det som Sun et al. (2021, s. 13) påpeker stemmer. Om det tas utgangspunkt i de oppgavene som er trukket frem i analysekapittelet, der elevene arbeider analogt, er det kun tre konsepter og fremgangsmåter som ikke blir arbeidet med. En av fremgangsmåtene som ikke kommer frem i disse oppgavene er fikle. Det å fikle handler, som nevnt i delkaptittel 2.6 Rammeverk, om å teste ut noe ukjent for å finne ut av hva det er og hvordan det fungerer (Barefoot Computing, u.å.-i). Dette kan være på grunn av at elevene i de ovennevnte analoge programmeringsoppgavene blir bedt om å bruke penn og papir. Penn og papir er redskaper

elevene allerede har godt kjennskap til, og dermed vil det ikke være noe behov for å bli kjent med ressursen de skal bruke for å løse oppgaven. På bakgrunn av avgrensningene som er gjort for denne masteroppgaven, kan det muligens være flere analoge programmeringsoppgaver i læremidlene, enn det vi har kategorisert som dette. Antakelsen presentert over kan derfor muligens ikke generaliseres til å gjelde alt arbeid med analoge programmeringsoppgaver på generelt grunnlag.

Et annet aspekt å merke seg i alle de analoge programmeringsoppgavene nevnt over, er at de innebærer arbeid med konseptet logikk. Dette samsvarer med det Sun et al. (2021, s. 12) fant ut i deres forskning om at analog programmering førte til at elevene ble bemerkningsverdig bedre på å tenke logisk. Videre kom de frem til at elevene ble bedre på å finne den enkleste løsningen på hverdagslige problemer (Sun et al., 2021, s. 12). Dette kan ha noe å gjøre med at elevene i slike oppgaver arbeider med håndfaste gjenstander som de ser hver dag, i motsetning til typiske programmeringsprogrammer som kanskje ikke er så naturlig å ta i bruk hver dag. På en annen side kan akkurat dette være tilfeldig, da vi som nevnt kun har sett på et lite utvalg av analoge programmeringsoppgaver i forhold til mengden av oppgaver som finnes.

Det vi har observert er at elevene jobber med både blokkprogrammering og analog programmering i programmeringsoppgavene i de ulike lærebøkene. Over er det vist til fordeler ved å undervise i analog programmering. Likevel viser Berg (2021, s. 51) til at analog programmering alene ikke gir elevene god nok kompetanse innen programmering. Hun fant ut at det beste for å kunne treffe større deler av elevgruppa var å bruke en kombinasjon av både analog programmering og blokkprogrammering (Berg, 2021, s. 51). Samtidig viser Forsström og Kaufmann (2018, s. 19) til at kunnskap om ulike programmeringsspråk er nyttig innen algoritmisk tenking. I de tre lærebøkene vi analyserte bestod programmeringsoppgavene kun av blokk- og analog programmering. Ved å da se på *tabell 18*, som viser oversikten over konsepter og fremgangsmåter innen algoritmisk tenking, kan det være mulig å se en sammenheng mellom denne typen programmeringsoppgaver og algoritmisk tenking. Det vi kan se ut ifra *tabell 18* er at elevene arbeider med omtrent alle konseptene og fremgangsmåtene i arbeidet med disse oppgavene. Dette stemmer overens med det som Berg (2021, s. 51) viser til om at en kombinasjon av både analog og blokkprogrammering er det mest hensiktsmessige for å treffe flest mulig av elevene.

5.3.3 Tekstprogrammering

Derimot finner vi ingen tekstprogrammeringsoppgaver i lærebøkene. Dette kan for eksempel ha noe å gjøre med elevenes alder. Lee et al. (2016, s. 187) påpeker at elevenes forståelse for programmering øker i takt med elevenes alder. Vi har tidligere omtalt blokkprogrammering og analog programmering som enklere å forstå for yngre elever. Mannila (2017, s. 71) peker også på at tekstprogrammering krever presisjon og nøyaktighet av den som programmerer. I en videre forklaring påpekes det at grunnen til at det er vanskeligere enn blokkprogrammering og analog programmering er at eventuelle skrivefeil i koden kan være vanskelige å oppdage (Mannila, 2017, s. 71). I og med at denne forskningen er gjennomført på matematikkbøker for elever på 6. trinn, kan det ha innvirkning på at vi ikke har funnet noen tekstprogrammeringsoppgaver, da disse er mer kognitivt krevende ifølge (Mannila, 2017, s. 71). Dette kan ses i sammenheng med at Gracin og Krišto (2022, s. 108) gjorde funn om at det er en tydelig overvekt av oppgaver i lærebøker som er lite komplekse og krever mindre av elevene kognitivt.

5.4 Tverrfaglighet

Husain et al. (2017, s. 1) peker på at programmering ikke bare er egnet til å tilegne seg ferdigheter i matematikk, men også det å kunne øke forståelsen for andre fag (Husain et al., 2017, s. 1). Dette er vesentlig for at programmering skal kunne brukes i den norske skolen, på grunn av at vi ikke har et eget fag som heter programmering (Vinnervik & Bungum, 2022, s. 384). På lik linje som at programmering er tverrfaglig, er også algoritmisk tenking en problemløsningsmetode som er tverrfaglig (Lodi & Martini, 2021, s. 896). Lodi og Martini (2021, s. 896) peker på at det å blant annet kunne designe, skape, kommunisere og samarbeide er viktig for å kunne fungere i dagens samfunn. Dette er også temaer som strekker seg over flere fag og kan brukes der etter. Et eksempel på dette kan være kunst og håndverk hvor både det å designe og skape kan være viktig. Skape er en av fremgangsmåtene i Barefoot Computings (u.å.-a). modell for algoritmisk tenking. Et kompetansemål etter 7. trinn for kunst og håndverk lyder «Bruke programmering til å skape interaktivitet og visuelle utrrykk.» (Kunnskapsdepartementet, 2019a, s. 7). I dette kompetansemålet er både «programmering» og «skape» brukt, som kan vise til at kunst og håndverk er et av fagene som også kan bidra til å legge til rette for algoritmisk tenking.

Læreplanen legger opp til at elevene skal lære programmering i ulike fag for å kunne bruke det som et verktøy og en metode for å tilegne seg kunnskap (Vinnervik & Bungum, 2022, s. 384). Moreno-León et al. (2016, s. 283) viser til at programmering i Scratch kunne brukes både i samfunnsfag og matematikk. Dette kan antyde at også andre læreverk enn de innen matematikk kan bidra til undervisning i programmering. Samtidig er programmering fortsatt forholdsvis nytt i skolen, da det kom inn i læreplanen i 2020 (Kunnskapsdepartementet, 2018). Kilhamn et al. (2021, 304) viser til at elevene vil kunne bruke programmering som et verktøy i større grad om noen år, hvor alle elevene har fått opplæring i programmering fra første klasse. De legger til at dette kan ha sammenheng med at siden det er nytt for lærerne også vil undervisningen om programmering av nødvendighet omhandle hva programmering er og hvordan det brukes (Kilhamn et al., 2021, s. 304). Dermed kan det se ut til at Husain et al.'s (2017, s. 1) poeng om at programmering kan brukes til blant annet problemløsning, først vil tre i kraft om noen år når programmering har vært med elevene fra første skoleår skal vi tro Kilhamn et al. (2021, s. 304).

6 Avslutning

I dette kapitelet presenteres en konklusjon for å svare på masteroppgavens problemstilling og til slutt noen forslag til hvordan veien videre kan være når det kommer til forskningen rundt dette temaet.

6.2 Konklusjon

Hovedmålet med denne masteroppgaven var å se på «Hvordan kan programmeringsoppgaver i lærebøker i matematikk på 6. trinn legge til rette for algoritmisk tenking.» Dette delkapittelet vil inneholde en kort oppsummering av funnene vi gjorde, samt en konklusjon for problemstillingen.

Ved hjelp av den horisontale analysen fikk vi innblikk i at elevenes utbytte av programmeringsoppgavene, dermed også algoritmisk tenking, kan avhenge av hvor i læreboken de er plassert (Brehmer et al., 2015, s. 589). Videre påpeker de at hvor noe er plassert i en lærebok kan ha noe å si for om elevene får arbeidet med innholdet (Brehmer et al., 2015, s. 589). Ved å se dette sammen med uttalelsene til Mullis et al. (2012, s. 391) om at læreboken ofte blir brukt som hovedgrunnlag for undervisningen, kan det se ut til at det kan oppstå ulikheter i læringsutbyttet ut ifra hvilke bøker som blir brukt. Det at læreboken ofte er hovedgrunnlaget for matematikkundervisningen, kan også føre til spenninger mellom det elevene skal lære og det de faktisk lærer. Et eksempel på dette kan være at kompetansemålene for matematikk som omhandler programmering, omhandler også geometriske mønstre (Kunnskapsdepartementet, 2019, s. 10). Samtidig er programmeringsoppgavene i to av lærebøkene vi analyserte i kapitler om algebra.

Videre i den vertikale analysen fant vi flere av konseptene og fremgangsmåtene innen algoritmisk tenking i henhold til Barefoot Computing's modell (Barefoot, u.å.-a). Alle programmeringsoppgavene i bøkene inneholdt konseptet algoritme. Dette kan forklares ved det (Stigberg & Stigberg 2020, s. 491) legger frem angående at algoritme er viktig innen programmering. Andre konsepter vert å trekke frem er abstraksjon og dekomponering, hvilket vi fant lite av. Dette kan være på grunn av formuleringen av oppgaveteksten i matematikkbøker, som ofte ifølge Gracin og Krišto (2022, s. 108) er på et lavt kognitivt nivå. Formuleringen som ofte blir brukt for lærebøker i matematikk er steg-for-steg formuleringer

(Bråting & Kilhamn, 2022, s. 605). Videre kan det ses på som at det henger sammen med det Zhang og Nouri (2019 s 18) sier om at hvor enkelt det er å implementere deler av algoritmisk tenking gjennom programmering avhenger av elevens kognitive nivå. Et annet konsept vi så i halvparten av programmeringsoppgavene er mønster. Dette kan ha med kompetansemålet for programmering å gjøre, fordi det som nevnt omhandler geometriske mønstre (Kunnskapsdepartementet 2019, s. 10). Samtidig kan det stemme overens med det Lv (2022, s. 11) sier om at mønster er mye brukt i oppgaver som innebærer algoritmisk tenking.

Feilsøking ble funnet i 19 oppgaver. Scratch er et program hvor blokkene som ikke passer sammen ikke vil gå sammen (Lee, 2011, s. 27). Dette kan både ses på som at det er mindre komplisert å finne feilen, samtidig som elevene får hjelp med feilsøkingen og dermed kan oppfatte det som enklere. Logikk henger sammen med feilsøking (Barefoot, u.å.). Dette stemte ifølge resultatene til denne masteroppgaven. I nesten alle oppgavene der det var feilsøking var det også logikk. Sun et al. (2021, s. 12) viser til at programmering fører til at elever blir bedre til å tenke logisk. Logikk var også et ganske fremtredende konsept hvor 35 av 40 oppgaver inneholdt dette konseptet. Som nevnt bruker elevene logikk når de feilsøker og ofte foregår feilsøkingen i en algoritme. Malik et al. (2019 s. 91) påpeker at elevene sitt arbeid med algoritmer øker logisk tenking.

Evaluering ble også oppdaget i flere av oppgavene hvor elevene var nødt til å ta beslutninger eller vurdere fremgangsmåte og resultatets kvalitet. En fremgangsmåte det var vanskelig å se var holde ut. Grunnen til dette kan være valget av metode, som gjorde det vanskelig å se om elevene ga opp eller fortsatte til de fikk til oppgaven. Likevel så vi det i to oppgaver ved hjelp av en tekstformulering på siden. For å kunne uttale seg ytterligere om dette er en fremgangsmåte som programmering kan legge til rette for, vil det være naturlig å gjøre en annen studie med en annen metode.

Ved å se på programmeringstyper i programmeringsoppgavene i lærebøkene i lys av algoritmisk tenking, så vi at det ble brukt mye blokkprogrammering og analog programmering. Lv et al. (2022, s. 14) viser til at visuelle kodemiljøer kan forbedre problemløsningsferdigheter – altså for eksempel algoritmisk tenking. Lærebøkene inneholdt også oppgaver som viser til at elevene skal bruke analog programmering som metode. Ifølge Sun et al. (2021, s. 13) ble elevenes ferdigheter innen algoritmisk tenking fremmet gjennom analog programmering. En av disse som Sun et al. (2021, s. 12) trekker frem er logisk

tenking. Det at analog programmering kan fremme algoritmisk tenking stemmer overens med våre funn hvor disse oppgavene var innom mange av de ulike komponentene innen algoritmisk tenking. Et eksempel på en fremgangsmåte som ikke kommer frem i analog programmering i denne oppgaven er fikle. Dette kan begrunnes ved at elevene kjenner til redskapen de skal bruke, altså penn og papir. Vi fant et minimum av oppgaver som omhandlet samarbeid. Til tross for at både Ananiadou og Claro (2009, s. 10) og Lodi og Martini (2021, s. 896) peker på at det er en viktig egenskap å ha innen algoritmisk tenking. Funnene våre kunne vært annerledes om vi hadde sett på lærerveiledningen i tillegg til læreboken. Berg (2021, s. 51) forslår en blanding av analog- og blokkprogrammering for å lære algoritmisk tenking og det samstemmer med tallene fra denne studien. Samtidig finner vi ingen oppgaver med tekstprogrammering i lærebøkene. Dette kan henge sammen med kompleksiteten som medfører disse programmene ifølge Mannila (2017, s. 71), sammen med at flesteparten av oppgavene inneholder lavt kognitivt nivå (Gracin & Krišto, 2022, s. 108).

For å svare på problemstillingen for denne masteroppgaven, vil vi trekke frem at samtlige av programmeringsoppgavene i lærebøkene, la til rette for arbeid med en eller annen fremgangsmåte, samt konsept innenfor algoritmisk tenking. Likevel kan man spørre seg om oppgavene legger til rette for problemløsningsmetoden algoritmisk tenking. Et spørsmål verdt å stille er hvor mange komponenter av algoritmisk tenking som må være inkludert i arbeidet med en oppgave, for at det skal være mulig å si at det har blitt brukt algoritmisk tenking for å løse oppgaven. Om en elev kun hadde hatt bruk for logisk tenking for å løse en oppgave, men ingen av de andre konseptene eller fremgangsmåtene, hadde eleven da brukt algoritmisk tenking for å komme frem til løsningen. Dette er spørsmål som er vanskelig å svare på da vi ikke har funnet noe tidligere forskning som sier noe om akkurat dette. Det det derimot er gjort funn om i tidligere forskning er at programmering kan brukes til flere formål (Husain et al., 2017, s. 1).

6.3 Veien videre

Om temaet i denne masteroppgaven skulle bli videreført til en annen studie, ville det vært interessant å se på andre aspekter. En annen innfallsvinkel er å se hvordan programmeringsoppgaver legger til rette for algoritmisk tenkning når et helt læreverk tas med i betraktning. Et helt læreverk inneholder trykte lærebøker, digitale læremidler samt

lærerveiledning som kan gi lærere forslag til hvordan oppgavene kan gjennomføres i undervisning. Dermed kan dette være med på å gi et mer helhetlig bilde av ressursene ute i skolen. Å gjøre et bredere utvalg av ressurser kan åpne for at det kan bli flere oppgaver å analysere, samt ulike typer oppgaver ut ifra hvor de er plassert (trykt eller digitalt). Dermed kan det også komme tydeligere frem om programmering kan legge til rette for algoritmisk tenking, samt ved at det er mer informasjon å hente om oppgavene i lærerveiledningen.

En annen innfallsvinkel som kan gjøres i en annen studie, kan være å ta med Charalambous et al. (2010, s.120) sin tredje analysestrategi som går på at det blir mulig å se hvordan temaet til denne masteroppgaven blir brukt og gjennomført i klasserommet. Dette kan gjøre at flere av komponentene innenfor algoritmisk tenking kommer frem. For eksempel holde ut, som var veldig vanskelig for oss å si noe om i denne masteroppgaven på bakgrunn av at vi ikke observerte elevene i arbeidet med oppgavene.

Ved å skrive denne masteren har vi fått innblikk i hvordan programmeringsoppgaver kan legge til rette for arbeid med algoritmisk tenking. Dette kan vi ta med inn i vår hverdag som lærere og dermed være mer bevisst på algoritmisk tenking under arbeide med programmering. Dette gir oss en styrke også i form av at vi har noe kunnskap om hva slags programmeringsoppgaver som finnes i de ulike læreverkene vi har analysert. Å vite noe om dette kan gi oss en fordel når det gjelder undervisning om programmering, fordi vi kan ha en anelse om hva som befinner seg i lærebøkene og eventuelt hva vi kan bli nødt til å finne andre steder. Arbeidet med denne masteroppgaven åpner også opp for muligheten for å bruke kunnskapen vi har tilegnet oss om programmering, inn i skolen på en hensiktsmessig måte. Dette er spesielt nyttig sett i lys av at programmering er såpass nytt i læreplanen for grunnskolen som det er.

Litteratur

Alseth, B., Røsseland, M., Arnås, A-C. & Nordberg, G. (2021). *Multi 6B: Elevbok* (3. utg.). Gyldendal

Ananiadou, K., & Claro, M. (2009). 21st century skills and Competences for New Millennium Learners in OECD Countries. *OECD Education Working papers, 41*, 1-33. https://doi.org/10.1787/218525261154

Aschehoug. (u.å.). *Matemagisk 1-7: Når elevene først forstår matematikk, blir magien aldri borte!* Hentet 23. Februar 2023 fra https://skole.aschehoug.no/barneskole/matematikk

Bai, H., Wang, X. & Zhao, L. (2021). Effects of the Problem-Oriented Learning Model on Middle School Students' Computational Thinking Skills in a Python Course. *Frontiers of Psychology, 12*, 1-14 Artikkel 771221. https://doi.org/10.3389/fpsyg.2021.771221

Barcelos, T. S., Munoz, R., Villarroel, R., Merino, E. & Silveira, I. F. (2018). Mathematics Learning through Computational Thinking Activities: A Systematic Literature Review. *Journal of Universal Computer Science, 24*(7), 815-845. https://doi.org/10.3217/jucs-024-07-0815

Barefoot Computing. (u.å.-a). Computational Thinking Consepts and Approaches. Hentet 9. mars 2023 https://www.barefootcomputing.org/concept-approaches/computational-thinking-concepts-and-approaches

Barefoot Computing. (u.å.-b). About barefoot. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/about-barefoot

Barefoot Computing. (u.å.-c) Logic. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/logic

Barefoot Computing. (u.å.- d). Algorithms. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/algorithms

Barefoot Computing. (u.å.-e). Decomposition. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/decomposition

Barefoot Computing. (u.å.-f). Patterns. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/patterns

Barefoot Computing (u.å.-g). Abstraction. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/abstraction

Barefoot Computing (u.å.-h). Evaluation. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/evaluation

Barefoot Computing (u.å.-i). Tinkering. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/tinkering

Barefoot Computing (u.å.-j). Creating. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/creating

Barefoot Computing (u.å.-k). Debugging. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/debugging

Barefoot Computing (u.å.-l). Persevering. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/persevering

Barefoot Computing (u.å.-m). Collaborating. Hentet 9. mars 2023 fra https://www.barefootcomputing.org/concepts-and-approaches/collaborating

Bellens, K., Van den Noortgate, W. & Van Damme, J. (2020). The informed choice: mathematics textbook assessment in light of educational freedom, effectiveness, and improvement in primary education. *School Effectiveness and School Improvement 31*(2), 192-211. https://doi.org/10.1080/09243453.2019.1642215

Berg, T.K. (2021). Analog programmering. *Tangenten – tidsskrift for matematikkundervisning, 32*(3), 42–52. http://tangenten.no/wp-content/uploads/2021/12/tangenten-3-2021-Berg.pdf

Bjerke, A. H., Kroknes, T. E. & Svingen, O. E .L. (2007). *Matemagisk 6A: Lærerveiledning*. Gyldendal.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K. (2016). *Developing computational thinking in compulsory education: Implications for policy and practice* (EUR 28295 EN). Publications office of the European union. https://publications.jrc.ec.europa.eu/repository/handle/JRC104188

Bratberg, Ø. (2021). *Tekstanalyse for samfunnsvitere* (3. utg.). Cappelen Damm Akademisk.

Brehmer, D., Ryve, A. & Van Steenbrugge, H. (2015). Problem solving in Swedish mathematics textbooks for upper secondary school. *Scandinavian Journal of Educational Research, 60*(6), 577–593. https://doi.org/10.1080/00313831.2015.1066427

Bråting, K., & Kilhamn, C. (2022). The Integration of Programming in Swedish School Mathematics: Investigating Elementary Mathematics Textbooks. *Scandinavian Journal of Educational Research*, *66*(4), 594–609. https://doi.org/10.1080/00313831.2021.1897879

Cappelen damm (u.å.). *Matematikk 5-7 fra Cappelen Damm: Om læreverk.* Hentet 23. februar 2023 fra https://www.cappelendammundervisning.no/verk/matematikk-5-7-fra-cappelen-damm-153428#omtale

Charalambous, C. Y., Delaney, S., Hsu, H. Y. & Mesa, V. (2010) A Comparative Analysis of the Addition and Subtraction of Fractions in Textbooks from Three Countries.

Mathematical Thinking and Learning, 12(2), 117-151, https://doi.org/10.1080/10986060903460070

Clark, Foster, L., Sloan, L., & Bryman, A. (2021). *Bryman's social research methods* (6. utg.). Oxford University Press.

Forsström, S. E. & Kaufmann, O. T. (2018). A Literature Review Exploring the use of Programming in Mathematics Education. *International Journal of Learning, Teaching and Educational Research, 17(*12), 18-32. https://doi.org/10.26803/ijlter.17.12.2.

Gadanidis, G., Hughes, J. M., Minniti, L. & White, B. J. G. (2017). Computational Thinking, Grade 1 Students and the Binomial Theorem. *Digital Experiences in Mathematics Education*, *3*(2), 77–96. https://doi.org/10.1007/s40751-016-0019-3

Gracin, D. G. & Krišto, A. (2022). Differences in the Requirements of Digital and Printed Mathematics Textbooks: Focus on Geometry Chapters. *CEPS Journal*, *12*(2), 95–117. DOI: 10.26529/cepsj.1285

Grover, S. & Pea, R. (2013). Computational Thinking in K-12: A Review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Gulbrandsen, J. E., Løchsen, R., Måleng, K. & Olsen, V. S. (2020). *Matematikk 6 fra Cappelen Damm: Grunnbok*. CAPPELEN DAMM

Gyldendal. (2021, 17. mars). *Multi og fagfornyelsen i matematikk: Matematikk 1-7*. https://www.gyldendal.no/artikler/multi-og-fagfornyelsen/?tags=10065

Husain, H., Kamal, N., Ibrahim, M. F., Huddin, A. B., & Alim, A. A. (2017). Engendering problemsolving skills and mathematical knowledge via programming. *Journal of Engineering Science & Technology*, *12*(12), 1–11. https://jestec.taylors.edu.my/Special%20Issue_PEKA_2017/PEKA%202017_01.pdf

Johansson, M. (2006). *Teaching mathematics with textbooks*. [Doktorgradsavhandling, Luleå University of Technology Department of Mathematics] Digitala Vetenskapliga Arkivet. http://ltu.diva-portal.org/smash/get/diva2:998959/FULLTEXT01.pdf

Kilhamn, C., Rolandsson, L & Bråting, K. (2021) Programming in Swedish school mathematics. *LUMAT, 9*(1), 283-312. https://doi.org/10.31129/LUMAT.9.2.1457

Kjällander, S. Mannila, L., Åkerfeldt, A. & Heintz, F. (2021). Elementary Students' First Approach to Computational Thinking and Programming. *Education Sciences*, *11*(80), 1-15. https://doi.org/10.3390/educsci11020080

Kongsnes, A. L., Raen, K. M. & Sørdal, M. (2021) *Matemagisk 6B: Grunnbok* (2. utg.). Aschehoug.

Krippendorf, K. (2019). Content Analysis: An Introduction to Its Methodology (4. utg.). SAGE.

Kunnskapsdepartementet. (2019a). *Læreplan i kunst og håndverk (KHV01-02).* Fastsatt som forskrift. Læreplanverket for kunnskapsløfte 2020. https://data.udir.no/kl06/v201906/laereplaner-lk20/KHV01-02.pdf?lang=nob

Kunnskapsdepartementet. (2019b). *Læreplan i matematikk 1.-10.trinn (MAT01-05).* Fastsatt som forskrift. Læreplanverket for kunnskapsløftet 2020. https://data.udir.no/kl06/v201906/laereplaner-lk20/MAT01-05.pdf?lang=nno

Kunnskapsdepartementet. (2018). Fornyer innholdet i skolen. Regjeringen.no. https://www.regjeringen.no/no/dokumentarkiv/regjeringen-solberg/aktuelt-regjeringen-solberg/kd/pressemeldinger/2018/fornyer-innholdet-i-skolen/id2606028/?expand=factbox2606064

Lee, J. (2020). Coding in early childhood. *Contemporary Issues in Early Childhood*, *21*(3), 266–269. https://doi.org/10.1177/1463949119846541

Lee, J. & Junoh, J. (2019). Implementing Unplugged Coding Activities in Early Childhood Classrooms. *Early Childhood Education Journal, 47(6)*, 709–716. https://doi.org/10.1007/s10643-019-00967-z

Lee, S., Kim, J., & Lee, W. (2016). Analysis of Factors Affecting Achievement in Maker Programming Education in the Age of Wireless Communication. *Wireless Personal Communications*, *93*(1), 187–209. https://doi.org/10.1007/s11277-016-3450-2

Lee, Y.-J. (2011). Scratch: Multimedia Programming Environment for Young Gifted Learners. *Gifted Child Today Magazine*, *34*(2), 26–31. https://doi.org/10.1177/107621751103400208

Lodi, M. (2020). Informatical Thinking. *Olympiads in Informatics, 14,* 113-132. DOI: 10.15388/ioi.2020.09

Lodi, M. & Martini, S. (2021). Computational Thinking, Between Papert and Wing. *Science & education, 30*(4), 883-908. https://doi.org/10.1007/s11191-021-00202-5

Lv, L., Zhong, B. & Liu, X. (2022) A literature review on the empirical studies of the integration of mathematics and computational thinking. *Education and Information Technologies,* (55), 1-23 https://doi.org/10.1007/s10639-022-11518-2

Malik, S. I., Shakir, M., Eldow, A., & Ashfaque, M. W. (2019). Promoting Algorithmic Thinking in an Introductory Programming Course. *International Journal of Emerging Technologies in Learning (iJET)*, *14*(01), pp. 84–94. https://doi.org/10.3991/ijet.v14i01.9061

Mannila, L. (2017). *Att undervisa i programmering i skolan. Varför, vad och hur?* Studentlitteratur.

Moreno-León, J., Robles, G., & Román-González, M. (2016). Code to learn: Where does it belong in the K-12 curriculum? *Journal of Information Technology Education*, *15*, 283–303. https://doi.org/10.28945/3521

Mullis, I. V. S., Martin, M. O., Foy, P. & Arora, A. (2012). *TIMSS 2011 International Results in Mathematics*. TIMSS & PIRLS International Study Center Lynch School of Education Boston College; International Association for the Evalution and Educational Achievement (IEA), Chestnut Hill, MA, Amsterdam. https://timssandpirls.bc.edu/timss2011/downloads/t11_ir_mathematics_fullbook.pdf

Neraal, A. (2022. 9. desember). Norge største forlag 2021. *BOK365*. https://bok365.no/artikkel/norges-50-storste-forlag-2021/

NOU 2014:7. (2014). *Elevenes læring i fremtidens skole: et kunnskapsgrunnlag.* Kunnskapsdepartementet. https://www.regjeringen.no/no/dokumenter/NOU-2014-7/id766593/

Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2020). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry*, *11*(1), 1–17. https://doi.org/10.1080/20004508.2019.1627844

Nylenna, M. (2017). Er den tradisjonelle læreboken avleggs? *Michael, 14*(2), 86-104. https://fhi.brage.unit.no/fhi-xmlui/handle/11250/2488038

Papert, S. (1980). Mindstorms; Children, Computers and Powerful Ideas. Basic Books.

Pepin, B., Gueudet, G. & Trouche, L. (2013). Re-sourcing teachers' work and interactions: a collective perspective on resources, their use and transformation. *ZDM, 45*(7), 929–943. https://doi.org/10.1007/s11858-013-0534-2

Postholm, M. P. & Jacobsen, D. I. (2018). *Forskningsmetode for masterstudenter i lærerutdanning.* Cappelen Damm Akademisk.

Sevik, K. et al. (2016). *Programmering i skolen*. Utdanningsdirektoratet. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/notat-om-programmering-i-skolen/

Skjelbred, D., Askeland, N., Maagerø, E. & Aamotsbakken, B. (2017). *Norsk lærebokhistorie: Allmueskolen–folkeskolen–grunnskolen 1739-1013*. Universitetsforlaget.

Stigberg, H., & Stigberg, S. (2020). Teaching programming and mathematics in practice: A case study from a Swedish primary school. *Policy Futures in Education*, *18*(4), 483–496. https://doi.org/10.1177/1478210319894785

Sun, L., Hu, L., & Zhou, D. (2021). Improving 7th-graders' computational thinking skills through unplugged programming activities: A study on the influence of multiple factors. *Thinking Skills and Creativity*, *42*, 100926. https://doi.org/10.1016/j.tsc.2021.100926

Tønnessen, E. S. (2013). Læreboka som kunnskapsdesign. I N. Askeland, E. Maagerø & B. Aamotsbakken (Red.), *Læreboka: studier i ulike læreboktekster* (s. 147-163). Akademika forlag.

Utdanningsdirektoratet. (2019. 27. mars). *Algoritmisk tenking*. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2021. 24. april). *Læremidler og læringsteknologi i skolen og opplæring*. https://www.udir.no/om-udir/tilskudd-og-prosjektmidler/tilskudd-til-laremidler/begrepsavklaring-skole/

Valverde, G. A., Bianchi, L. J., Wolfe, R. G., Schmidt, W. H., & Houang, R. T. (2002). *According to the Book: Using TIMSS to investigate the translation of policy into practice through the world of textbooks*. Springer Netherlands. https://doi.org/10.1007/978-94-007-0844-0

Van den Ham, A. K & Heinze, A. (2018). Does the textbook matter? Longitudinal effects of textbook choice on primary school students' achievement in mathematics. *Studies in Educational Evaluation,* 59, 133-140. https://doi.org/10.1016/j.stueduc.2018.07.005

Vinnervik, P. & Bungum, B. (2022). Computational thinking as part of compulsory education: How is it represented in Swedish and Norwegian curricula? *Nordina: Nordic studies in science education*, *18*(3), 384-400. https://doi.org/10.5617/nordina.9296

Wing, J. M. (2006). Computational thinking. *Communications of the ACM. 49*(3), 33-35. *https://doi.org/10.1145/1118178.1118215*

Zhang, & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers and Education*, *141*, 103607. https://doi.org/10.1016/j.compedu.2019.103607