# INDSÆT SKOLENS OFFICIELLE FORSIDE

# MIN FORSIDE

# RESUME

# INDHOLDSFORTEGNELSE

# INDLEDNING

# Giv en kort introduktion til forskellige bevismetoder i matematik

Peter vil komme med formel der ikke gælder uendeligt

Strukturel induktion

Fuldstændig induktion. Stærk induktion

Det jeg laver i Binets formle er svag induktion

Direkte bevis, modstridsbevis sqrt2 er irrationel, eksistensbevis, geometriske beviser (anal. plan), entydighedsbevis

Sprogligt overbeviser mange. Godt mat bevis overbeviser alle

# Redegør i detaljer for induktionsbevismetoden herunder induktionsbasen, induktionsantagelsen og induktionstrinnet. Medtag gerne et konkret bevis eksempel (du kan f.eks. eftervise Binets formel)

Induktionsbevismetoden er en metode der kan bruges til at bevise at et udtryk er gyldigt for alle naturlige tal …………

Induktionsbevismetoden består af 2 skridt: Induktionsbasen og induktionstrinnet. I induktionsbasen viser man at udtrykket der skal bevises, er gyldigt for en bestemt talværdi for eksempel 0. Herefter viser man i induktionstrinnet at hvis man antager at udtrykket er gyldigt for tallet n, så er udtrykket også gyldigt for tallet n+1.

## Bevis for Binet’s formel

Et eksempel på et udtryk som kan bevises ved brug af matematisk induktion, er Binet’s formel:

Binet’s formel er en formel der kan bruges til at beregne det n’te tal i fibonacci-talfølgen som er en talfølge der er givet ud fra følgende rekursive talfølge:

Startbetingelserne til talfølgen er:

Indutkionsbasen vil nu blive at vise at Binet’s formel gælder for n=0 og n=1. For at vise at formlen gælder for begyndelsesbetingelsen indsættes 0 på n's plads i formlen:

Formlen gælder altså når n er 0. For at vise at formlen gælder for begyndelsesbetingelsen indsættes 1 på n's plads i formlen.

Formlen gælder derfor også når n er 1.

# Introducér sorteringsproblemet og analysér og sammenlign de to sorteringsalgoritmer InsertionSort og MergeSort. DU kan f.eks. analysere og sammenligne deres køretider. Vis endvidere deres korrekthed ved at opstille passende løkkeinvarianter

## Introduktion til sorteringsproblemet

## InsertionSort

InsertionSort er en algoritme der kan anvendes til at løse sorteringsproblemet. For at vise hvordan algoritmen fungerer, kan man skrive den op i pseudokode:

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

For at forstå hvordan algoritmen fungerer, er det vigtigt at have en grundlæggende viden om hvordan pseudokode opskrives. Derfor vil nogle af de grundlæggende begreber indenfor pseudokode nu blive introduceret.

Når der skrives A[i] betyder det at man tilgår det i’te element i listen A. Når man skriver A[1..j] indikerer man at man snakker om en delliste af A der indeholder elementerne A[1], A[2], A[3] … A[j]

Man tildeler værdi til variabler ved at bruge er lig med tegnet (=), så hvis man for eksempel skriver min\_variabel = 3, så tildeler man værdien 3 til variablen min\_variabel. Ligeledes tildeler man for eksempel værdien A[j] til variablen key på linje 2.

På linje 1 laves der en **for** løkke, her sættes variablen j til at være 2 til at starte med. Hele løkkens krop, det vil sige alt under linje 1, som er indrykket (dvs. linje 2-8), vil nu blive udført én gang hvor j = 2. Herefter vil j blive gjort én større, dvs. j = 3. Nu køres hele koden igennem hvor j = 3. Herefter vil j blive gjort én større igen og løkken vil blive kørt igen. Denne proces fortsætter indtil j er lig med A.length dette bliver udtrykt når der skrives ”**to** A.length”. Det vil sige løkkens krop køres én gang for værdierne j = 2,3,4…. A.length. Når koden har kørt for j = A.length, vil j bliver gjort én større dvs. j = A.length + 1 og for-løkken vil ikke længere blive kørt, i pseudokode vil j dog beholde sin værdi. Når en løkke udføres sin krop, kaldes det er iteration.

En anden form for løkke, som der gøres brug af, er en **while** løkke. Denne bruges på linje 5. En **while** vil køre sin krop så længe udtrykket der står efter **while** er sandt, med andre ord vil løkken køre indtil udtrykket er falskt. Det vil sige at **while** løkken på linje 5 vil køre så længe i er større end 0 og A[i] er større end key. Rent praktisk fungerer **while** løkken ved at den først tjekker om udtrykket er sandt og hvis dette er tilfældet, så udføres kroppen én gang. Herefter tjekkes det igen om udtrykket er sandt og i så fald udføres kroppen igen og dette fortsætter indtil udtrykket bliver falskt. Det vil sige at hvis i på et tidspunkt i løkkens krop bliver sat til 0, så vil løkken stoppe med at køre når udtrykket tjekkes igen.

### Korrekthed af InsertionSort

Metoden til at bevise korrektheden af en algoritme, minder meget om induktionsbevismetoden. Lighederne mellem disse metoder vil blive diskuteret i et senere afsnit. For at vise korrektheden af en algoritme opstiller man en såkaldt løkkeinvariant. En løkkeinvariant er et udtryk der kan bruges til at fortælle hvorfor en algoritme er korrekt. Der skal vises tre ting om denne løkkeinvariant:

**Initialization:** Løkkeinvarianten er sand før den første iteration af løkken.

**Maintenance:** Hvis løkkeinvarianten er sand før en given iteration af løkken, så forbliver den sand før den næste iteration af løkken.

**Termination:** Når løkken er kørt færdig, giver løkkeinvarianten en information eller egenskab der kan bruges til at vise at algoritmen er korrekt (Cormen, et al., 2009, p. 19).

Når man har vist at løkkeinvarianten er sand før den første iteration af løkken og at den er sand før den næste iteration såfremt at den var sand før en given iteration af løkken, kan man slutte at løkkeinvariant må være sand før enhver iteration af løkken.

Hvis man vil vise korrektheden af InsertionSort, kan man opstille følgende løkkeinvariant: For hver iteration af for-løkken i linje 1-8, består dellisten A[1..j-1] af de oprindelige elementer i A[1..j-1] men i en sorteret rækkefølge. (Cormen, et al., 2009, p. 18)

Nu skal det altså vises at denne løkkeinvariant er sand før den første iteration af for-løkken, at hvis løkkeinvarianten er sand før en iteration af løkken, forbliver den også sand inden den næste iteration af løkken og at løkkeinvarianten ved løkkens afslutning kan bruges til at vise at InsertionSort er korrekt.

Først vises det at løkkeinvarianten gælder før den første iteration af for-løkken (Initialization). Før første iteration af j = 2 og det vil sige at A[1..j-1] bliver til A[1..1] og består derfor kun af ét element nemlig A[1], dette element er selvfølgelig også det oprindelige element A[1]. Siden der kun er ét element i dellisten, må dellisten være sorteret. Det er nu vist at alle betingelserne for løkkeinvarianten er sande før den første iteration af løkken og derfor er løkkeinvarianten sand før den første iteration af løkken.

## MergeSort

En anden algoritme der kan bruges til at løse sorteringsproblemet, er MergeSort. InsertionSort løste sorteringsproblemet ved at sortere ét tal ad gangen, ind i den allerede sorterede del af listen. MergeSort løser derimod sorteringsproblemet på en rekursiv måde, det vil sige at den kalder sig selv med mindre sorteringsproblemer. MergeSort gør brug af Divide-and-conquer metoden (del-og-hersk metoden på dansk). I divide-and-conquer metoden gør man brug af tre simple trin, for hvert rekursionsniveau: Divide, conquer og combine.

I divide-trinnet deler man problemet i flere underproblemer, som ligner det oprindelige problemet. I MergeSort gøres dette ved at dele listen af tal midt over, sådan at man har to halve lister, som man i stedet kan sortere.

I combine-trinnet, kombinerer man løsningerne på underproblemerne, sådan at man har løst det oprindelige problem. I MergeSort gøres dette ved at kombinere de to halve (og nu sorterede) lister, til én samlet sorteret liste (Cormen, et al., 2009, p. 30).

Et billede, der indeholder tekst

Automatisk genereret beskrivelse

## Implementering og analyse

Hvis man gerne vil benytte algoritmerne på sin computer, er det dog ikke nok at have pseudokoden. Man bliver nødt til at implementere dem i et reelt programmeringssprog. Der findes mange programmeringssprog, men i denne opgave vil der blive brugt programmeringssproget JavaScript. InsertionSort kan implementeres i JavaScript på følgende måde:



Som det kan ses, minder koden meget om den pseudokode, der før blev vist for InsertionSort. Der er dog nogle små forskelle. Først og fremmest bruger man ikke indrykning til at specificere kroppe i JavaScript, man bruger i stedet tuborg-klammer. Så ”{” markerer starten af en krop og ”}” markerer slutningen af en krop. Det kan herfra udledes at for-løkken i den ovenstående kode har en krop fra linje 2 til 10. Første gang man introducerer en variabel i JavaScript putter man også ordet let foran variabel navnet, som der for eksempel gøres på linje 3 og 4.

## Sammenligning

## Korrekthed

# Diskutér kort lighederne mellem dit bevis ved matematik induktion og metoden, du benyttede til at vise korrektheden af InsertionSort og MergeSort.

# Bibliografi

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C., 2009. *Introduction to Algorithms.* 3. red. Cambridge, Massachusetts: Massachusetts Institute of Technology.

# Bilag