NTNU

Kunnskap for en bedre verden

TDT4105 Informasjonsteknologi, grunnkurs

Matlab: Sortering og søking

Amanuensis Terje Rydland
Kontor: ITV-021 i IT-bygget vest (Gløshaugen)
Epost: terjery@idi.ntnu.no
Tlf: 735 91845

Kunnskap for en bedre verden

Palindrom - iterativt

- Def: Tekststreng som kan leses likt begge veier
  - Otto, Radar, Agnes i senga, …
- Test flanke-tegnene
  - **Ulike**: Ikke et palindrom
  - **Like**: Sjekk om teksten uten flanke-tegnene er et palindrom

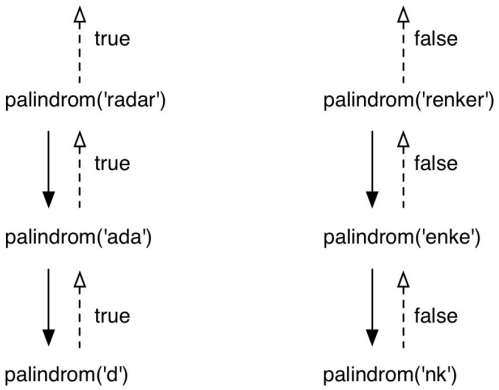
```
Pseudokode
   Finn lengden av teksten
   Gjør om teksten til små bokstaver
   Så lenge det er bokstaver igjen eller endebokstaver ikke er like
      sammenlign første og siste bokstav
      hvis de er like
         ferdig
      sammenlign første og siste bokstav
         fjern første og siste bokstav
```

Palindrom - iterativt

```
function palindrom = isPalindrome(ord)
    ordlengde = length(ord);  %finner lengden av ordet
    ord = lower(ord);         %gjør om til bare små bokstaver
    palindrom = false;        %intialiserer returvariabelen
    start = 1;
    slutt = ordlengde;
    ferdig = false;           %initialiserer stoppkriteriet
    while ~ferdig
        if ord(start) ~= ord(slutt)  %sammenligner bokstaver
            ferdig = true;
        else
            start = start+1;  %hvis bokstavene var like må vi
            slutt = slutt-1;  %plukke ut to nye bokstaver
            if start > slutt
                ferdig = true;
                palindrom = true;
            end %if
        end %if
    end %while
end %function
```


Palindrom - rekursivt

- Def: Tekststreng som kan leses likt begge veier
  - Otto, Radar, Agnes i senga, …
- Stopptilfelle: 0 eller 1 tegn er et palindrom
- Test flanke-tegnene
  - **Ulike**: Ikke et palindrom
  - **Like**: Sjekk om teksten uten flanke-tegnene er et palindrom



palindrom.m

```
function p = palindrom(tekst)
% sjekker om tekst er et palindrom (leses likt begge veier)

   lengde = length(tekst);
   if lengde < 2
       p = true;
   elseif lower(tekst(1)) == lower(tekst(lengde))
       p = palindrom( tekst(2:lengde-1) );
   else
       p = false;
   end

end % function
```

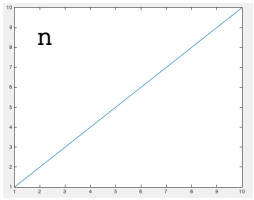
Bruker lower-funksjonen for å gjøre tegnsammenligningene uavhengig av små/store bokstaver, slik at f. eks. 'RADar' blir et palindrom

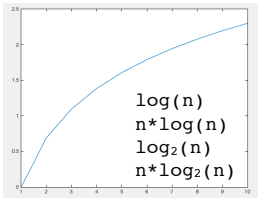
O-notasjon og kompleksitet

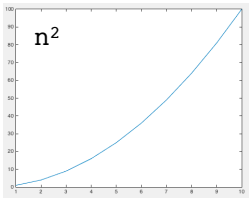
- Ønsker et mål på hvor lang tid en algoritme bruker på å kjøre som en funksjon av mengde inndata, N
- Vi teller enkle oppgaver som antall sammenligninger, antall ombyttinger, regneoperasjoner og lignende som en funksjon av N
- Nøyaktig tid avhenger av programmering, språk og datamaskin
- Ønsker et estimat for økningen av kjøretid i forhold til problemstørrelse

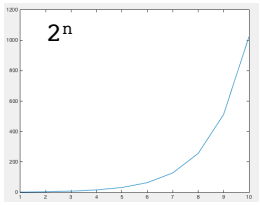
Kompleksitet

N	log(N)	N*log(N)	log2(N)	N*log2(N)	N^2	2^N
10	1	10	3	33	100	1024
100	2	200	7	664	10000	1,26765E+30
1000	3	3000	10	9966	1000000	1,0715E+301
10000	4	40000	13	132877	100000000	"Mye"
100000	5	500000	17	1660964	10000000000	"Mye"
1000000	6	6000000	20	19931569	1E+12	"Mye"
10000000	7	70000000	23	232534967	1E+14	"Mye"
100000000	8	800000000	27	2657542476	1E+16	"Mye"
1000000000	9	9000000000	30	3,E+10	1E+18	"Mye"



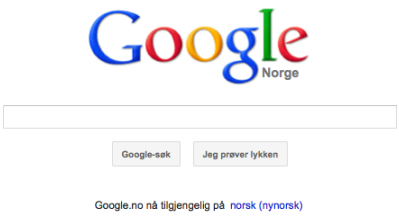






Sortering og søking

- Kanskje det datamaskiner brukes aller mest til
- eks: Google
- Ikke forbeholdt datatekniske fag
- Kommer inn i mange teknologiske og naturvitenskap
- Sentrale "bygge-klosser" å kjenne til.



Pensum

- Matlab-boka: 12.3 og 12.5
- Stoffet oppfattes ofte som forholdsvis vanskelig
- Eksamensrelevant
- Du må prøve ting selv

- Video for å visualisere det som foregår

Læringsmål

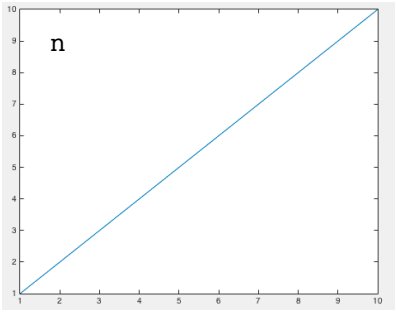
- Forstå prinsipper for søking i sorterte vs. tilfeldige (ikke-sortere) datamengder
- Kunne programmere søking fra grunnen av
- Kunne skrive et program for sortering ut fra en algoritme eller oppgitt pseudokode
- Betydningen av en algoritmes kompleksitet og O-notasjon.

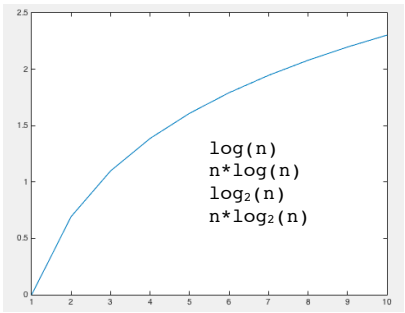
Sortering

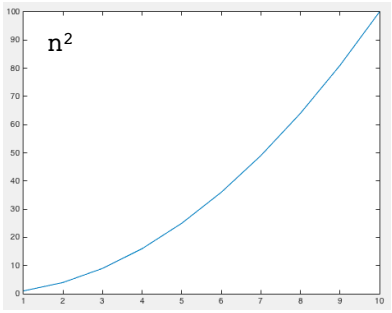
- Sortering er et stort delområde innen datateknikken
- Matlab har en innebygd funksjon sort()
- Skal lære noen enkle sorteringsalgoritmer
- Se wikipedia.org (sorteringsalgoritmer) for en oversikt over sorteringsalgoritmer og deres egenskaper

Antall operasjoner for å sortere data

- En ideell sorteringsalgoritme er O(N).
- De beste sorteringsalgoritmene basert på å sammenligne to og to elementer er O(N * logN)
- De enkleste sorteringsalgoritmene er $O(N^2)$
- Hvorfor vil man ikke bruke $O(N^2)$-algoritmer for å sortere store datamengder?



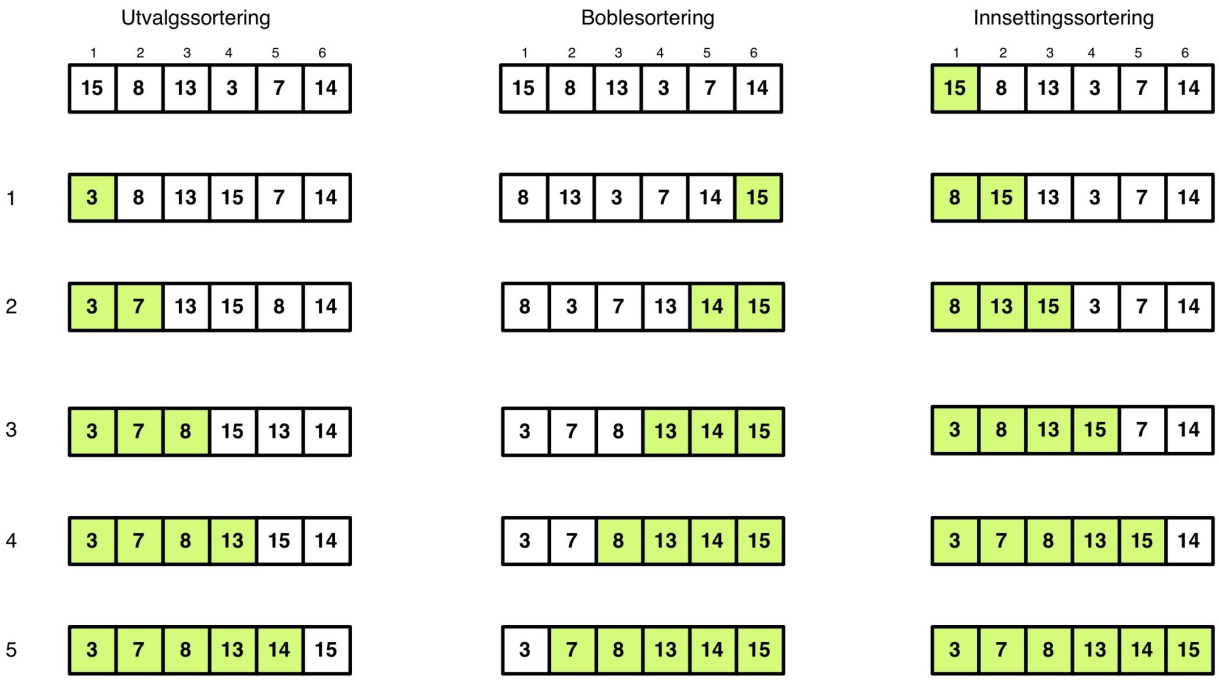




Enkle sorteringsalgoritmer

- Utvelgelsessortering (selection sort)
  - Finner det minste gjenværende elementet i hver runde
- Boblesortering (bubble sort)
  - Sammenligner nabo-elementer
  - Bytter om ved feil rekkefølge
  - Beveger seg mot sortert rekkefølge, minst ett tall kommer på plass i hver runde.
- Sortering ved innsetting (insertion sort)
  - Som i kortspill
  - Setter inn i sortert delmengde som begynner med ett element og vokser til alle.

NTNU



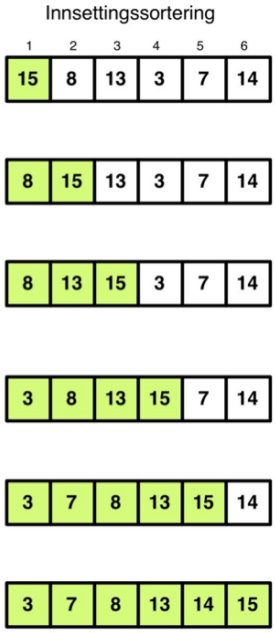
NTNU

Pseudokode for sortering ved innsetting

Input: Ikke-tom liste med tall, L

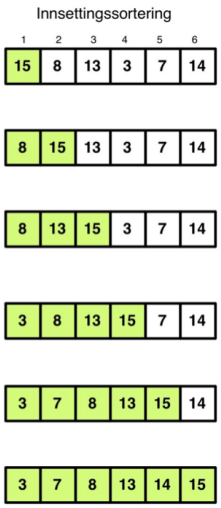
Output: Sortert liste L

```
Foreach index i = 2:lengthof(L) do
  j = i
  while (j > 1) og ( L(j-1) > L(j) )
       bytt om L(j-1) og L(j)
       j = j-1
  end
end
```



innsettingsSortering.m

```
function liste = innsettingsSortering(liste)
    for i = 2:length(liste)
        %setter nr. i inn i sortert del
        j = i;
        while (j > 1) && (liste(j-1) > liste(j))
            % Bytter om elementene
            tmp = liste(j-1);
            liste(j-1) = liste(j);
            liste(j) = tmp;
            j = j - 1;
        end %while
    end %for
end %function
```



Merknader

- Ikke egnet for store datamengder – $O(N^2)$
- Bruker ikke ekstra lagerplass (ekstra tabeller)
- Hvordan blir det hvis vi ønsker synkende sortering?

Fletting av sorterte lister

- To sorterte lister kan enkelt flettes til en sortert liste
  - 2 15 33 40
7 8 30 45 99
  - 2 7 8 15 30 33 40 45 99
- Plukker det minste gjenværende elementet i hvert trinn
- Når en liste er tom, er det bare å legge til det som er igjen av den andre.

flettLister.m

```
function l = flettLister(a,b) %Fletter 2 sorterte lister
    a_lengde = length(a);
    b_lengde = length(b);
    a_neste = 1;    % indekser til neste element i listene
    b_neste = 1;
    l_neste = 1;

    while (a_neste <= a_lengde) && (b_neste <= b_lengde)
        if a(a_neste) <= b(b_neste) % henter fra a
            l(l_neste) = a(a_neste);
            a_neste = a_neste + 1;
        else                        % henter fra b
            l(l_neste) = b(b_neste);
            b_neste = b_neste + 1;
        end %if
        l_neste = l_neste + 1;
    end %while
    if a_neste > a_lengde       % a er tom. Legg inn resten av b
        for j = b_neste:b_lengde
            l(l_neste) = b(j);
            l_neste = l_neste + 1;
        end %for
    else    % b er tom. Legg inn resten av a
        for j = a_neste:a_lengde
            l(l_neste) = a(j);
            l_neste = l_neste + 1;
        end %for
    end % if
end %function
```

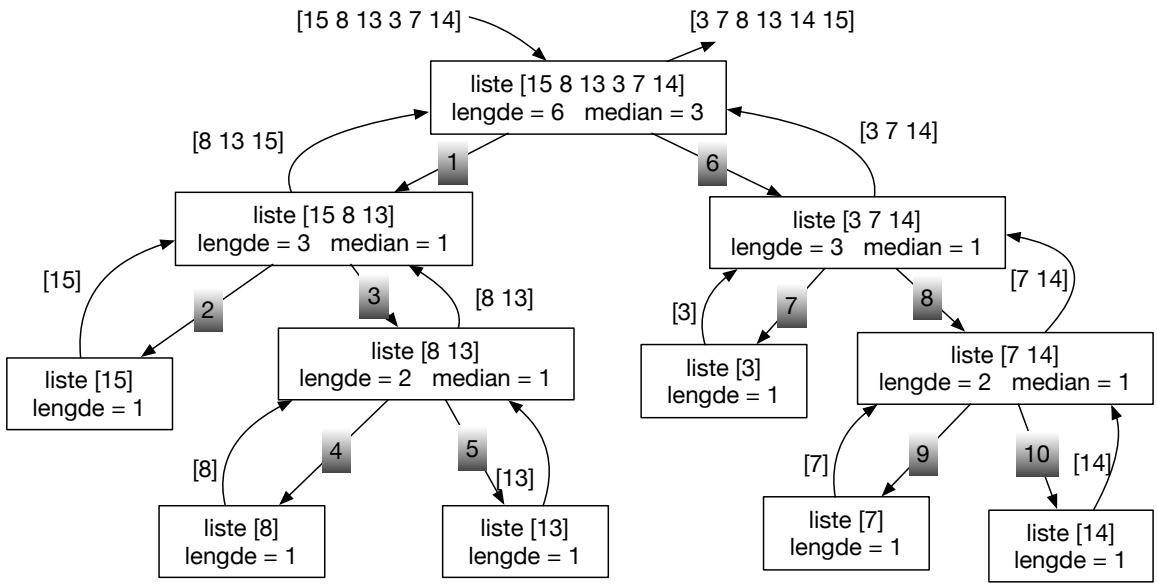

Sortering ved fletting

- Deler listen i to, sorterer halvpartene, fletter dem sammen
- Rekursivt
  - Stopper når en liste har ett element (sortert!)
- O(NlogN)

flettesortering.m

```
function sortert = fletteSortering(liste)
% Sorterer liste med rekursiv flettesortering
    lengde = length(liste);
    if lengde <= 1
        sortert = liste;
    else
        median = floor(lengde/2);
        sortert = flettLister(fletteSortering(liste(1:median)),...
            fletteSortering(liste(median+1:lengde)));
    end %if
end %function
```


Kalltre flettesortering



Merknader

- Noen ulemper med rekursjon.
  - Et funksjonskall er relativt kostbart.
  - Går med minne til lokale variabler for hver gang funksjonen kalles.
- Kan vi gjøre flettesorteringen mer effektiv?
  - Gå over til sortering ved innsetting når listene er korte nok.
  - I tabellen på neste lysark:
    - F/5 – flettesortering med overgang når listelengde <= 5
    - F/100 - … <= 100
    - Osv.

Kjøretider

Antall	I-sort	F-sort	F/5	F/10	F/50	F/100	F/250	F/500
1000	0,02	0,04	0,01	0,008	0,004	0,003	0,005	0,009
2000	0,07	0,09	0,03	0,02	0,009	0,007	0,01	0,02
5000	0,4	0,2	0,06	0,04	0,02	0,02	0,02	0,03
10000	1,6	0,5	0,1	0,08	0,04	0,04	0,05	0,07
25000	10,3	1,2	0,5	0,3	0,1	0,1	0,1	0,2
100000	165	4,6	2	1,1	0,5	0,5	0,6	0,9

Søkealgoritmer

- En søkealgoritme er en algoritme som søker gjennom en datamengde, på jakt etter en bestemt verdi.
- Vi skal begrense oss til det enkleste, søking i lister
- Vi skal se på to typer søk:
  - Sekvensielt søk
  - Binærsøk

Sekvensielt søk

- Går gjennom alle dataelementene fra begynnelse til slutt
- Tidkrevende, må gå gjennom alle elementene
- Fordringsløs – krever ikke noen forhåndsinformasjon om datamengden
- Eksempel:
  - Finne ut om et telefonnr finnes i telefonkatalogen

finnesTall.m

```
function funnet = finnesTall(liste, tall)
% Returnerer true hvis tall finnes i liste, false ellers
% Sekvensiell søkning

    funnet = false;

    for i = 1:length(liste)
        if liste(i) == tall
            funnet = true;
            return
        end %if
    end %for

end % function
```


Varianter

- Finne antall forekomster at tallet i listen
- Finne første indeks (posisjon) til tall i listen
- Finne alle indeksene til tall i listen

- Prøv gjerne å programmere disse selv!

- Sekvensielt søk er så langsomme at man søker alternativer:
  - Sortering og så søking
  - Søketrær (indekser)

29

Binærsøking

- Hvis datamengden er sortert (ordnet) kan søket gjøres mye mer effektivt
- Sjekker midterste element i datamengden
  - Elementet er der -> ferdig
  - Fortsetter å lete i den relevante halvparten
- Arbeidsmengde
  - Antall halveringer ca $\log_2(N)$
  - Sekvensiell søking: ca N.
  - Stor forskjell når N er stor

NTNU

30

Koding av binærsøk

- Koder b_finnesTall(liste, tall) med binærsøking
- Forutsetter at liste er sortert stigende
- Bruker l(ow) og h(igh) for å avgrense et mindre og mindre område der tall kan finnes
- Regner ut elementet midt i dette området
  - median = floor((l+h)/2)
  - Husk at floor runder nedover til nærmeste heltall

NTNU

b_finnesTall(liste,tall)

```
function funnet = b_finnesTall(liste,tall)
% Returnerer true hvis tall finnes i liste, false ellers
% Bruke binærsøk, lista antas å være sortert stigende

    l = 1;              % Minste indeks
    h = length(liste);  % Største indeks

    while l <= h
        % Finner midtpunkt
        median = floor((l+h)/2);

        if liste(median) == tall
            l = h+1; % Tall er funnet. Dette avslutter løkken
        elseif liste(median) < tall
            % Må lete i øvre halvdel
            l = median + 1;
        else
            % Må lete i nedre halvdel
            h = median – 1;
        end %if
    end % while

    % Setter returverdi
    funnet = liste(median) == tall;

end % function
```


b_finnesTall(minliste, 8) - programsporing

Steg	liste	tall	l	h	median	funnet
1	[1 8 16 23 30]	8	-	-	-	-
2			1	-	-	-
3				5	-	-
4					3	-
5				2		-
6					1	-
7			2			-
8					2	-
9			3			-
10						true

b_finnesTall(minliste, 10) - programsporing

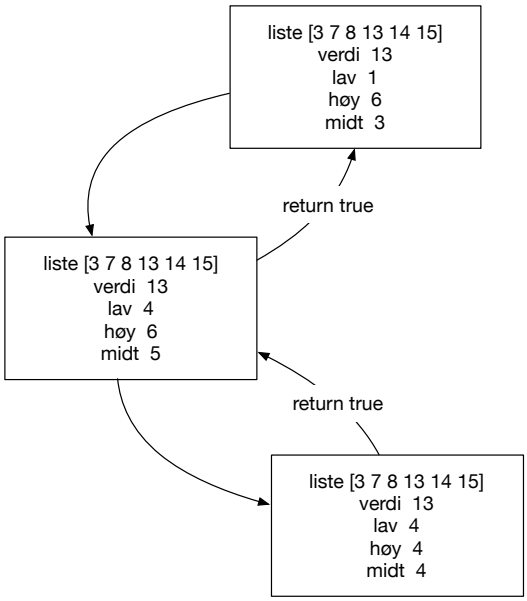
Steg	liste	tall	l	h	median	funnet
1	[1 8 16 23 30]	10	-	-	-	-
2			1	-	-	-
3				5	-	-
4					3	-
5				2		-
6					1	-
7			2			-
8					2	-
9			3			-
10						false

Binærsøking rekursivt

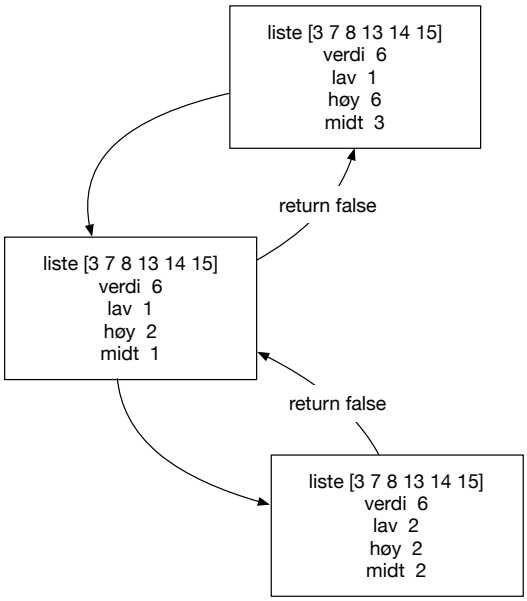
```
function funnet = binaersoek(liste, verdi, lavIndeks, hoyIndeks)
% Rekursivt binærsøk etter verdi i liste

    midten = floor((hoyIndeks + lavIndeks)/2);

    if lavIndeks > hoyIndeks
        funnet = false;
    elseif verdi == liste(midten)
        funnet = true;
    elseif verdi < liste(midten)
        funnet = binaersoek(liste, verdi, lavIndeks, midten – 1);
    else
        funnet = binaersoek(liste, verdi, midten + 1, hoyIndeks);
    end % if

end % function
```

Binærsøking rekursivt graf

Søk etter 13

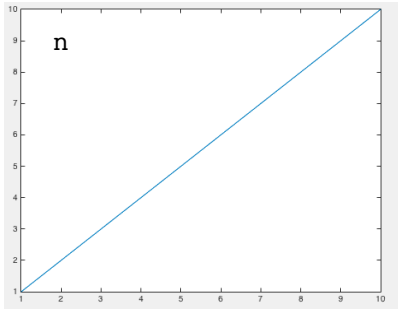


Søk etter 6



Kompleksitet for søking

- Sekvensiell søking vokser med N
  - Kjøretiden vil bli $k_1*N + k_0$
  - For stor N dominerer k_1*N
  - Vi sier at algoritmen er O(N)
  - Vokser N, øker kjøretiden proporsjonalt med N



- Binærsøking gjør $\log_2(N)$ sammenligninger
  - Kjøretiden vil bli $k_1*\log_2(N) + k_0 = k*\log(N) + k_0$
  - Vokser N, øker kjøretiden proporsjonalt med log(N)
  - Vi sier at algoritmen er O(log(N))