

ALGORITMER

Farmålet med denne note er at give en introduktion til:

Algoritmer

- Specifikation, f.eks. v.h.a. pseudokode
- Analyse
 - Køretid, v.h.a. asymptotisk notation
 - Korrektitud, f.eks. v.h.a. løkke-invarianter

Dette emne er et uddrag af kurset
DM578 - Algoritmer og Datastrukturer
som ligger på 2. semester (d.v.s. i foråret 2023)
og undervises af Rolf Fagerberg

En **algoritme** er en „opskrift“ på, hvordan man løser et givet problem.

Mere formelt (Fra Computer Science: An Overview
Brookeshear, Brylaw):

An algorithm is an **ordered** set of **unambiguous**, **executable** steps that define a **terminating** process.

Til at beskrive algoritmer er det nyttigt at bruge **pseudokode**:

„Høj-niveau sprog“, hvor man bruger **keywords**, som i programmerings-sprog, men også **almindelig tekst**.

Eks:

Søg efter et element x i en liste L

Kan gøres v.h.a. *sekventiel søgning* (også kaldet *linear søgning*):

Sequential Search (L, x)

$n := L.length$

$i := 1$

While $i \leq n$ and $L[i] \neq x$

$i++$

If $i \leq n$

Return i

Else

Return "Not found"

Er dette en algoritme?

Ordered, unambiguous, executable, terminating?

Eks: $L = [1, 7, 8, 2, 5, 9, 3, 11]$, $x = 5$

$L = [1, 7, 8, 2, 5, 9, 3, 11]$
↑

$[1, 7, 8, 2, 5, 9, 3, 11]$
↑

$[1, 7, 8, 2, 5, 9, 3, 11]$
↑

$[1, 7, 8, 2, 5, 9, 3, 11]$
↑

$[1, 7, 8, 2, 5, 9, 3, 11]$
↑

x og samtlige elementer til venstre for x checkes.
Hvis x ikke findes i L , checkes alle elementer i L .

Bemærk, at den samlede køretid er proportional med $\#check$.

Derfor siger vi, at køretiden er $O(n)$.

O -notationen giver en øvre grænse for "størrelsesordenen" af køretiden.

Hvis L er sortert, kan det gøres mere effektivt v.h.a. binær søgning:

Eks: $L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$, $x = 10$

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

↑
Check midtste element

$x > 8$, så vi fortsætter søgningen til højre for 8:

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

$x < 12$

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

Binary Search (L, x)

$l := 1$, $r := L.length$, $m := \lfloor \frac{l+r}{2} \rfloor$

While $l \leq r$ and $L[m] \neq x$

If $x < L[m]$

$r := m - 1$

Else

$l := m + 1$

$m := \lfloor \frac{l+r}{2} \rfloor$

If $l \leq r$

Return m

Else

Return „Not found“

Eks: $L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$, $x = 10,5$

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

Diagram showing the initial search range for $x = 10,5$. The array L is shown with indices l at 1, m at 8, and r at 15. The element 8 is highlighted in red.

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

Diagram showing the search range after the first step. The search range is now from $l = 9$ to $r = 15$. The element 12 is highlighted in red.

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

Diagram showing the search range after the second step. The search range is now from $l = 10$ to $r = 15$. The element 10 is highlighted in red.

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

Diagram showing the search range after the third step. The search range is now from $l = 11$ to $r = 15$. The element 11 is highlighted in red.

$L = [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15]$

Diagram showing the search range after the fourth step. The search range is now from $l = 12$ to $r = 15$. The element 11 is highlighted in red.

$r < l \Rightarrow$ „Not found“

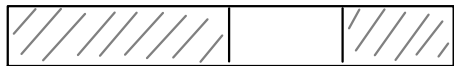
I eksemplerne overfor checkede vi h.h.v. 3 og 4 tal ud af 15. Bemærk, at 4 er worst-case.

Hvor mange tal kommer vi i værste tilfælde til at checke, når $L.length = n$?

Efter første check er der $\leq n/2$ elementer tilbage:



Efter andet check er der $\leq n/4$ elementer tilbage:



...

Efter i 'te check er der $\leq n/2^i$ elementer tilbage.

Det sidste check foretages senest, når der er 1 element tilbage.

$$\frac{n}{2^i} < 1 \quad (\Leftrightarrow) \quad n < 2^i \quad (\Leftrightarrow) \quad \log_2 n < i$$

D.v.s. max # check: $\lfloor \log_2 n \rfloor + 1$

Dermed er køretiden $O(\log n)$.

Bestemmelse af køretid

Vælg en **karakteristisk operation** op., sådan at algoritmens køretid er proportional med den samlede tid brugt på op.

I ovenstående eks. var op. sammenligning af elementer.

Sequential Search laver højst n sammenligninger. Derfor er dens køretid $O(n)$. Det ville den også være, hvis algoritmen lavede $n/2$ eller $3n$ sammenligninger.

Definition:

$T(n) \in O(f(n))$, hvis der findes $k, m \in \mathbb{Z}$, så
 $T(n) \leq k \cdot f(n)$, for alle $n \geq m$.

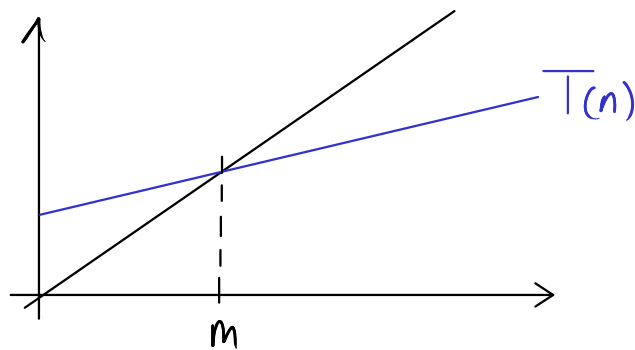
Eller:

$\Leftrightarrow T(n) \in O(f(n))$
 $\exists k, m \in \mathbb{Z} : \forall n \in \mathbb{Z}, n \geq m : T(n) \leq k \cdot f(n)$

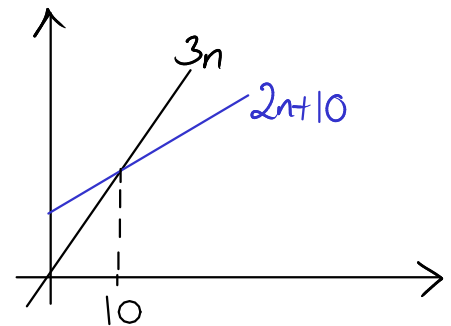
D.v.s. $f(n) \in O(T(n))$, hvis $\frac{f(n)}{T(n)}$ er (begrænset af) et tal; d.v.s. vokser ikke, når n vokser.

Også selvom ovenstående kun gælder for store n -værdier.

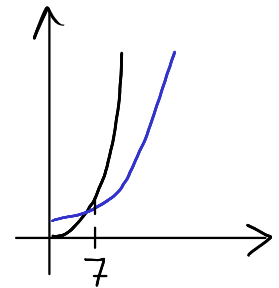
D.v.s. $T(n) \in O(n)$, hvis grafen for $T(n)$ fra et vist punkt ligger under en ret linie gennem $(0,0)$:



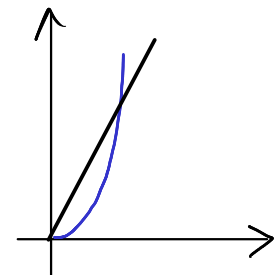
Eks: $2n+10 \in O(n)$:
 $2n+10 \leq 3n$, for $n \geq 10$



Eks: $n^2/2 + 3n + 1 \in O(n^2)$:
 $n^2/2 + 3n + 1 \leq n^2$, for $n \geq 7$



Eks: $n^2 \notin O(n)$:
 $n^2 \leq k \cdot n \Leftrightarrow n \leq k$



Eks: $\lfloor \log_2 n \rfloor + 1 \in O(\log n)$:

$\lfloor \log_2 n \rfloor + 1 \leq 2 \log_2 n$, for $n \geq 2$

Bemærk:

$$\log_a(n) \in O(\log_b(n)), \text{ hvis } a, b \in O(1)$$

fordi:

$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)}, \text{ og}$$

$$a, b \in O(1) \Rightarrow \log_b(a) \in O(1)$$

Derfor skriver vi blot $O(\log n)$ i st. for $O(\log_2 n)$

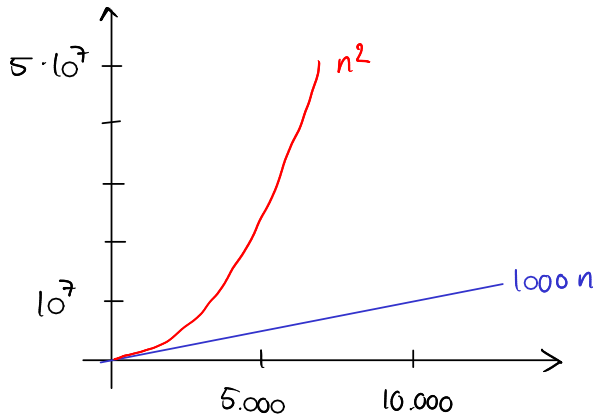
Eks: $a=2, b=10.$

$$\log_2(n) = \frac{\log_{10}(n)}{\log_{10}(2)} \approx \frac{\log_{10}(n)}{0,3} \approx 3,3 \cdot \log_{10}(n)$$

Det, der betyder noget, er det mest betydende led, d.v.s. det der vokser hurtigst.

Man må altid fjerne mindre betydende led, og konstanter, der gænger på det mest betydende led.

Eks:



Hvis n er lille, bekymrer vi os ikke så meget om køretiden, men vælger blot den simpleste algoritme. Hvis n er stor, er n^2 meget større end $1000n$.

Eks: #op. i sek: 10^9

#op	$n=100$		$n=10^6$		$n=10^9$	
	#op.	tid	#op	tid	#op.	tid
$\lceil \log_2 n \rceil$	7	7 ns	20	20 ns	30	30 ns
n	100	100 ns	10^6	1 ms	10^9	1 s
n^2	10.000	10 μ s	10^{12}	17 min	10^{18}	32 år

$$10^9 \approx 30.000.000 \cdot 30$$

D.v.s. om køretiden er $\frac{1}{2} \cdot n$ eller $10 \cdot n$ er relativt uvigtigt. Det, der betyder noget, er, om den er proportional med n eller $\log n$.

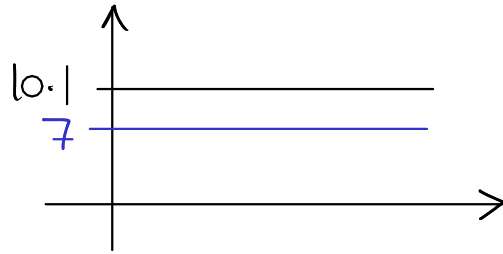
Ovenstående tabel angav, hvor lang tid det tager at løse et problem af en given størrelse v.h.a. en algoritme med en given køretid.

Nu vender vi det rundt: Givet en bestemt køretid, hvor stor en instans kan vi løse inden for den givne tid?

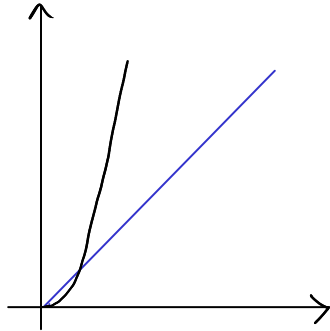
Vi antager stadig, at der kan udføres 10^9 op. i sek. Problem-størrelserne nedover er angivet med et betydende ciffer.

#op. for input af str. n	1 ms	1 s	1 min	1 døgn	1 år
$\log_2 n$	$10^{300.000}$				
n	10^6	10^9		$9 \cdot 10^{13}$	
$n \log_2 n$	$6 \cdot 10^4$			$2 \cdot 10^{12}$	
n^2	10^3			$9 \cdot 10^6$	
n^3	10^2	10^3		$4 \cdot 10^4$	
2^n	20	30			55

Eks: $7 \in O(1)$



Eks: $n \in O(n^2)$



Eksempler på køretider:

- 1 konstant
- $\log(n)$ logaritmisk
- n lineær
- $n \log(n)$
- n^2 kvadratisk
- n^3
- n^{10}
- 2^n
- 10^n

} polynomiel

} eksponentiel

Korrektthed

Virker algoritmen, som den skal?

Til at bevise korrektheden af en iterativ (eller rekursiv) algoritme kan man bruge en løkke-invariant.

Sequential Search (L, x)

$n := L.length$

$i := 1$

While $*I*$ $i \leq n$ and $L[i] \neq x$

$i++$

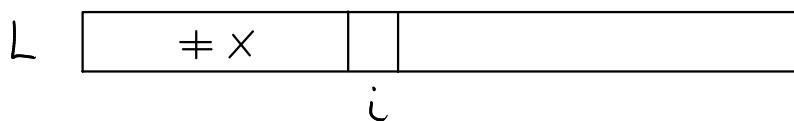
If $i \leq n$

Return i

Else

Return "Not found"

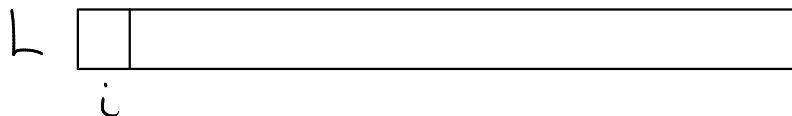
Løkke-invariant I : $x \neq L[j]$, for $1 \leq j \leq i-1$



I er opfyldt, hver gang vi kommer til $*I*$,
d.v.s. lige når vi skal til at checke betingelsen
 $i \leq n$ and $L[i] \neq x$.

Dette kan bevises o.h.a. induktion.

Første gang vi kommer til $*I*$:



I siger, at der ikke er nogen elementer til venstre for plads i , som er lig med x .

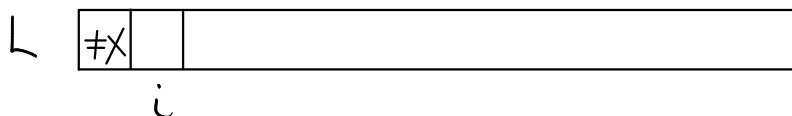
Der er ingen elementer til venstre for plads i , så udsagnet er trivelt opfyldt.

Dette udgør basis-skridtet.

Derefter checker vi, om $L[i] \neq x$.

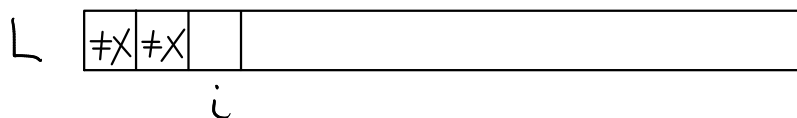
Hvis det er tilfældet, inkrementeres i .

Nu har vi følgende situation:



D.v.s. næste gang, vi kommer til $*I*$, er I igen opfyldt.

Hvis vi igen finder, at $L[i] \neq x$, inkrementeres i endnu en gang. Nu har vi:

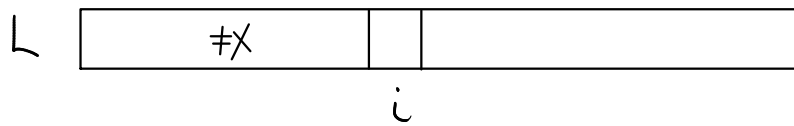


D.v.s. næste gang, vi kommer til $*I*$, er I igen opfyldt.

Og sådan kan vi fortsætte...

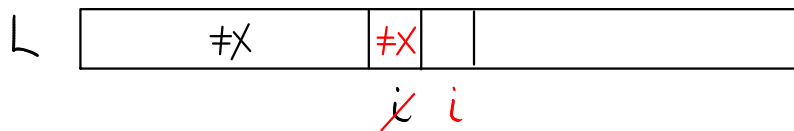
Generelt:

Antag, at I er opfyldt, når vi kommer til $\heartsuit I \heartsuit$:

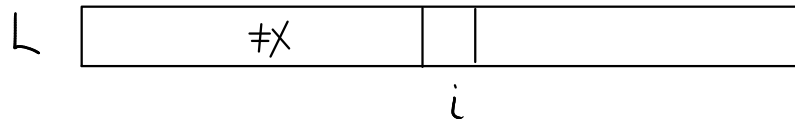


Dette er induktionsantagelsen.

Bemærk, at i kun bliver inkrementeret, hvis $L[i] \neq X$:



D.v.s. næste gang vi kommer til $\heartsuit I \heartsuit$, gælder I igen:



Dette udgør induktionskridtet.

D.v.s. i induktionskridtet viser vi:

Hvis I gælder, sidst vi var ved I ,
så gælder den også denne gang.

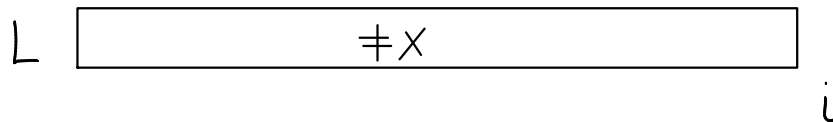
Afslutning:

Når vi til sidst forlader while-løkken, skyldes det en af to ting:

- $i = n+1$

I dette tilfælde returneres "Not found".

Da $i > n$, følger det af I, at x ikke findes i L .



D.v.s. korrekt output i dette tilfælde.

- $L[i] = x$

I dette tilfælde returneres i , og det er indeks til en plads i L , hvor man kan finde x .

D.v.s. også korrekt output i dette tilfælde.

Binary Search (L, x)

$l := 1, r := L.length, m := \lfloor \frac{l+r}{2} \rfloor$

While ~~*I*~~ $l \leq r$ and $L[m] \neq x$

If $x < L[m]$

$r := m - 1$

Else

$l := m + 1$

$m := \lfloor \frac{l+r}{2} \rfloor$

If $l \leq r$

Return m

Else

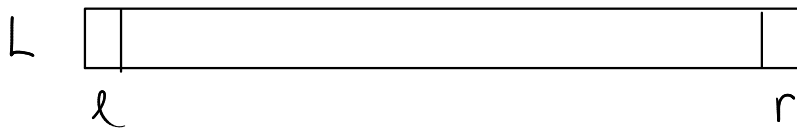
Return „Not found“

Invariant I:

Hvis x findes i L , findes den i $L[l..r]$:



Første gang vi kommer til ~~*I*~~, er det trivielt sandt:

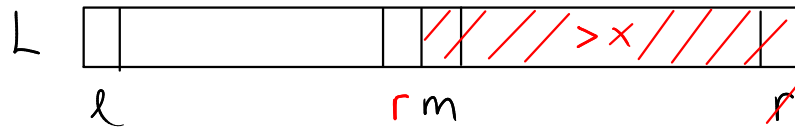


Derefter checker vi, om $x \neq L[m]$.

Hvis det er tilfældet er der to muligheder:

- $x < L[m]$

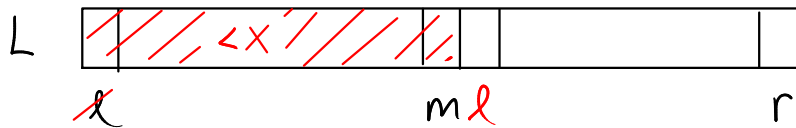
I dette tilfælde opdateres r :



Herefter er I stadig opfyldt

- $x > L[m]$

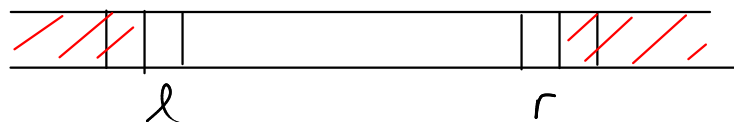
I dette tilfælde opdateres l :



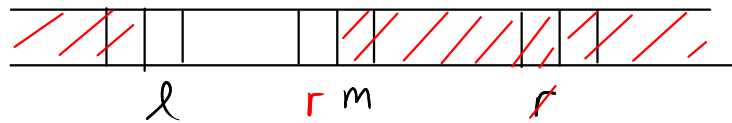
Herefter er I stadig opfyldt

Genselt:

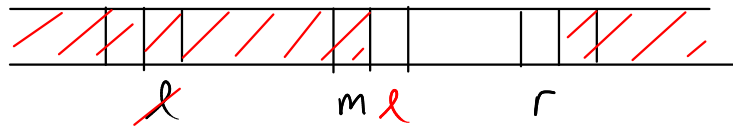
Lige inden vi checker $l \leq r$ and $L[m] \neq x$,
gælder I:



Efter gennemløb af løkken gælder I stadig:



eller

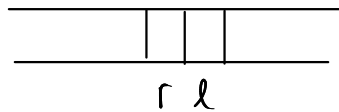


Når vi til sidst forlader while-løkken, skyldes det en af to ting:

- $l > r$

I dette tilfælde returneres "Not found".

Da $l > r$, er $L[l..r]$ tomt:



Dermed følger det af I, at x ikke findes i L .
D.v.s. korrekt output i dette tilfælde.

- $L[m] = x$

I dette tilfælde returneres i , og det er indeks til en plads i L , hvor man kan finde x .

D.v.s. også i dette tilfælde korrekt output.

De to algoritmer er **iterative**, d.v.s. det meste af arbejdet foregår i en (while-)løkke. De kunne også skrives som **rekursive** algoritmer, d.v.s. algoritmer, som kalder sig selv:

Seq Search Rec (L, x, l, n)

```
If  $l \leq n$ 
  If  $L[l] = x$ 
    Return  $l$ 
  Else
    Return SeqSearchRec( $L, x, l+1, n$ )
Else
  Return "Not found"
```

Binary Search Rec (L, x, l, r)

```
If  $l \leq r$ 
   $m := \lfloor \frac{l+r}{2} \rfloor$ 
  If  $L[m] = x$ 
    Return  $m$ 
  Else if  $x < L[m]$ 
    Return BinarySearchRec( $L, x, l, m-1$ )
  Else
    Return BinarySearchRec( $L, x, m+1, r$ )
Else
  Return "Not found"
```

Eks : $X = 10$

$L = [7, 9, 10, 3, 2, 8]$

SeqSearchRec($L, 10, 1, 6$)

If $1 \leq 6$

If $L[1] = 10$

Return 1

Else

Return SeqSearchRec($L, 10, 2, 6$)

Else

Return "Not found"

If $2 \leq 6$

If $L[2] = 10$

Return 2

Else

Return SeqSearchRec($L, 10, 3, 6$)

Else

Return "Not found"

If $3 \leq 6$

If $L[3] = 10$

Return 3

Else

Return SeqSearchRec($L, 10, 4, 6$)

Else

Return "Not found"

...