ALGORITMER

En algoritme er en „opskrift" på, hvordan man løser et bestemt problem.

Mere formelt:

> 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 key words, som i programmerings-sprog, men også almindelig tekst.

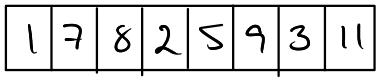
Søg efter et element x i en liste L

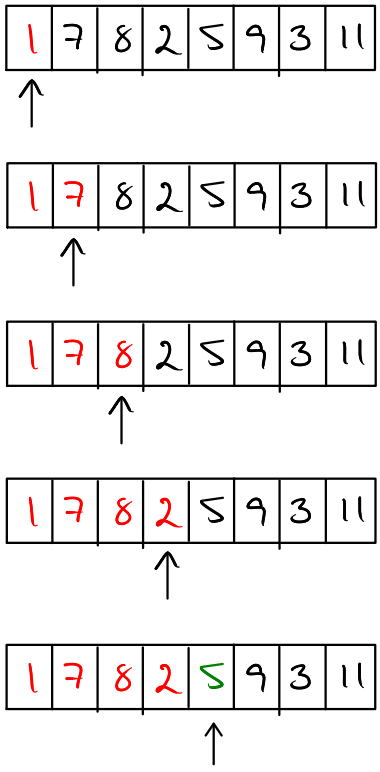
Kan gøres v.h.a. sekventiel søgning (også kaldet lineær søgning):

```
SequentialSearch(L, x)
n = L.length
i = 1
While i≤n and L[i] ≠ x
    i++
If i≤n
    Return i
Else
    Return "Not found"
```

Er dette en algoritme?
Ordered, unambiguous, executable, terminating?

Eks: $x = 5$

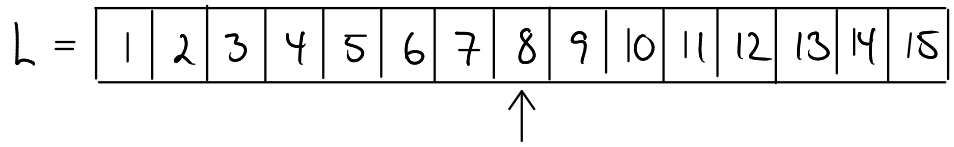
$L =$ 

$L =$ 

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

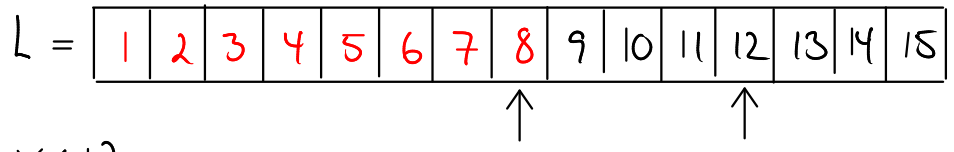
Hvis L er sorteret, kan det gøres mere effektivt o.h.a. binær søgning:

Eks: $x = 10$

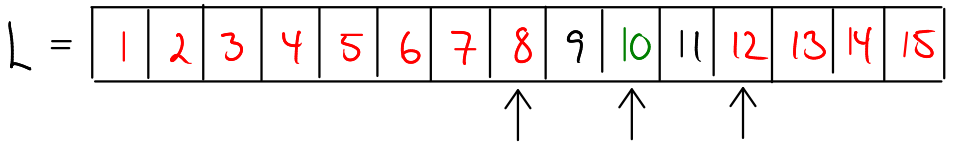


Check midterste element

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



$x < 12$

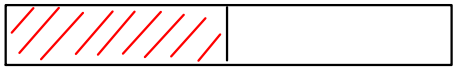


Binary Search (L , x)
$l = 1$, $r = L.length$, $m = \lfloor \frac{l+r}{2} \rfloor$
While $l \leq r$ and $L[m] \neq x$
    If $x < L[m]$
        $r = m - 1$
    Else
        $l = m + 1$
    $m = \lfloor \frac{l+r}{2} \rfloor$
If $l \leq r$
    Return m
Else
    Return "Not found"

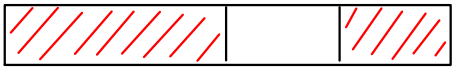
I eksemplet ovenfor checkede vi 3 ud af 15 tal.
I værste tilfælde ville vi have checket 4 tal
(hvis vi f.eks. søgte efter 11).

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.

$$\frac{n}{2^i} \leq 1 \iff n \leq 2^i \iff \log_2 n \leq i$$

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

Bemærk, at den samlede køretid er proportional med #check.
Derfor siger vi, at køretiden er $O(\log n)$.

O-notationen giver en øvre grænse for „størrelses-ordenen" af køretiden.

Bestemmelse af køretid

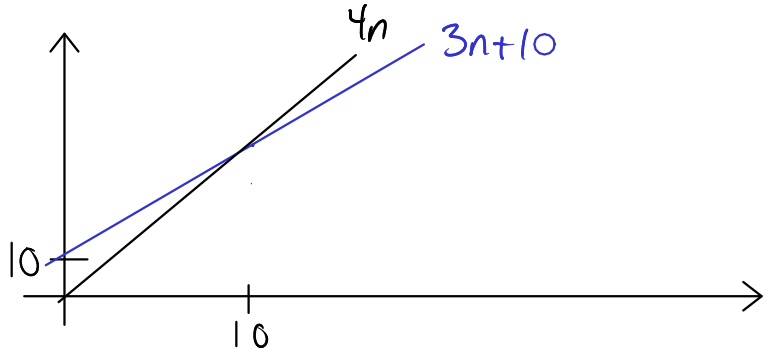
Vælg en karakteristisk operation op., sådan at algoritmens køretid er proportional med den samlede tid brugt på op.
I ovenstående eks. var op. sammenligning af elementer.

Hvad er køretiden af SequentialSearch?
I værste tilfælde sammenligner vi x med hvert af de n elementer i L.
D.v.s. køretiden er $O(n)$.

Hvis vi lavede $n/2$ eller $3n$ sammenligninger, ville køretiden også være $O(n)$.

Definition:
$T(n) \in O(f(n))$, hvis der findes $k, n' \in \mathbb{Z}$, sådan at $T(n) \leq k \cdot f(n)$, for alle $n \geq n'$.

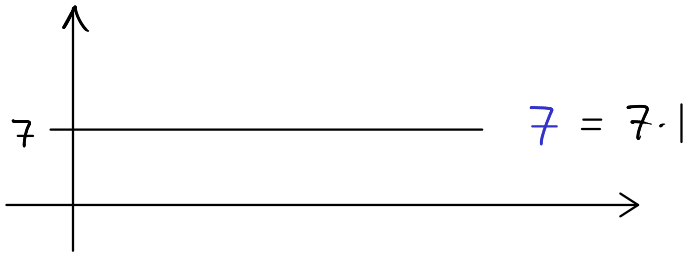
Eks: $3n + 10 \in O(n)$



Her er
$k = 4$ og $n' = 10$

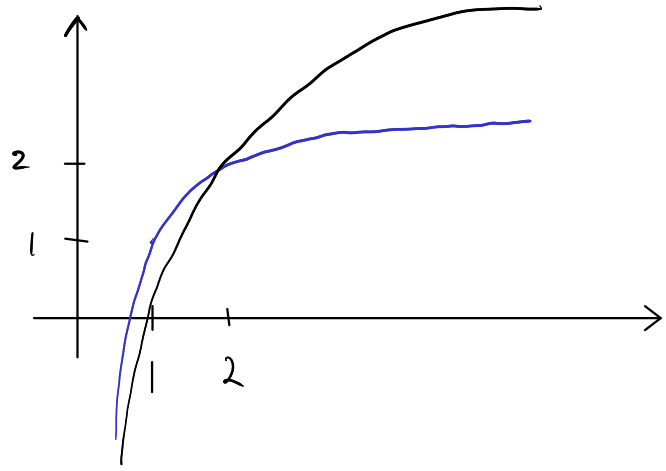
Det, der betyder noget, er, hvor hurtigt $T(n)$ vokser.

Eks: $7 \in O(1)$



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

$\log_2(n)+1 \leq 2\log_2(n)$, for $n \geq 2$:



$k = 2$

$n' = 2$

Bemærk:

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

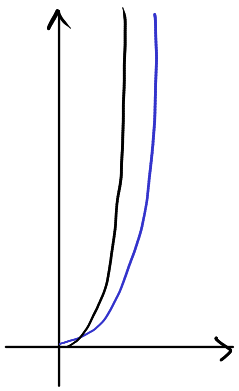
fordi:

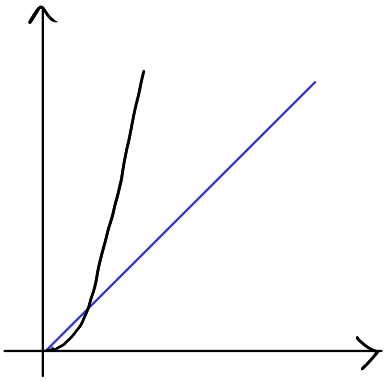
$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)}, \text{ og}$$

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

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

Eks: $10n^2 + n \in O(n^2)$ og $n \in O(n^2)$





Alternativ definition:

$T(n) \in O(f(n))$, hvis der findes $k, n' \in \mathbb{Z}$, så

$$\frac{T(n)}{f(n)} < k \quad \text{for} \quad n \geq n'$$

Eks:

$3n+10 \in O(n)$:

$$\frac{3n+10}{n} = 3 + \frac{10}{n} \leq 4 \quad \text{for} \quad n \geq 10$$

$\log(n)+1 \in O(\log n)$

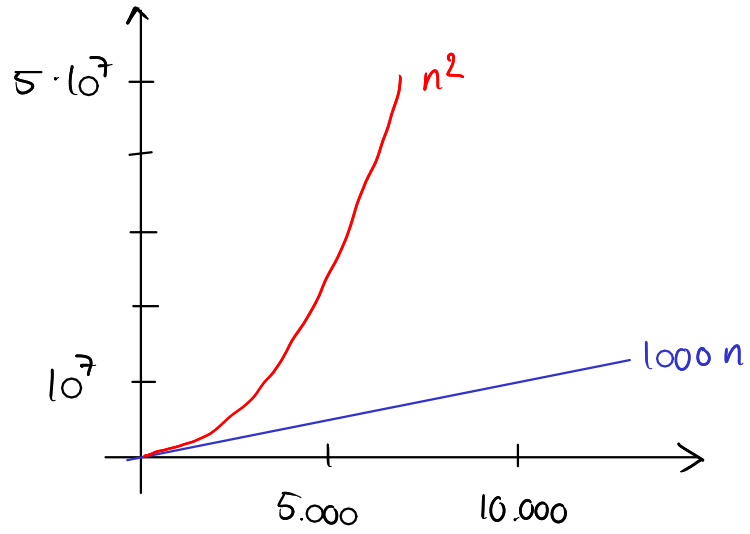
$$\frac{\log(n)+1}{\log(n)} = 1 + \frac{1}{\log n} \leq 2 \quad \text{for} \quad n \geq 2$$

$10n^2+n \in O(n^2)$:

$$\frac{10n^2+n}{n^2} = 10 + \frac{1}{n} \leq 11 \quad \text{for} \quad n \geq 1$$

Det, der betyder noget, er det <u>mest betydende</u> led, d.v.s. det der vokser hurtigst.
Man må altid fjerne mindre betydende led, og konstanter, der ganges på det mest betydende led.

<u>Eks:</u>



Eksempler på køretider:

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

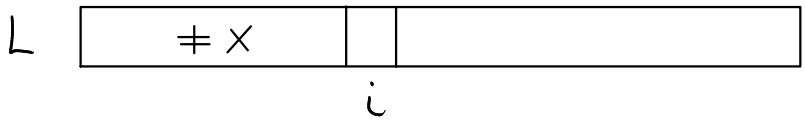
Antag, at der kan udføres 10^9 operationer per sekund. Nedenstående tabel viser, for forskellige køretider, hvor store input, der kan behandles inden for hhv. 1 sek, 1 min, 1 dag og 1 år.

#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

Virker algoritmen, som du skal?
Til at bevise korrekthuden af en iterativ (eller rekursiv) algoritme kan man bruge en løkke-invariant.

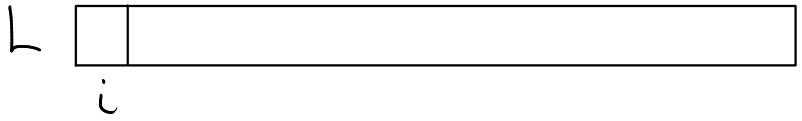
SequentialSearch (L, x)
n = L.length
i = 1
While *I* i≤n and L[i] ≠ x
    i++
If i≤n
    Return i
Else
    Return „Not found"

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



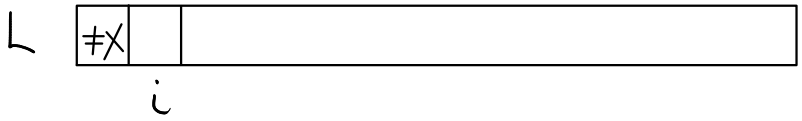
I er opfyldt, hver gang vi kommer til *I*, d.v.s. lige når vi skal til at checke betingelsen i≤n and L[i] ≠ x.
Dette kan bevises v.h.a. induktion, som I lærer om i DM549 om et par uger:

Første gang vi kommer til *I*:

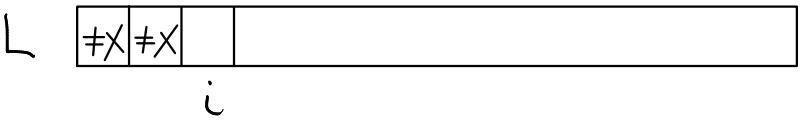


I siger, at der ikke er nogen elementer til venstre for plads i, som er lig med x.
Der er ingen elementer til venstre for plads i, så udsagnet er trivielt opfyldt.

Derefter checker vi, om L[i] ≠ 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] ≠ 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...

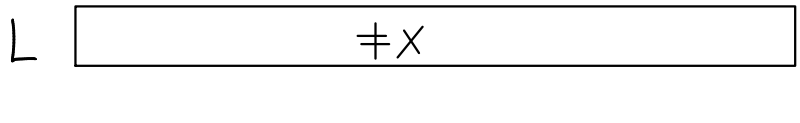
Når vi til sidst forlader while-løkken, skyldes det en af to ting:

- $L[i] = x$

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

- $i = n+1$

  I dette tilfælde returneres „Not found".
  Da $i > n$, følger det af I, at x ikke findes i L.



  D.v.s. 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 $l \geq r$

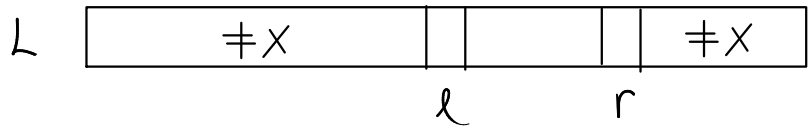
    Return m

Else

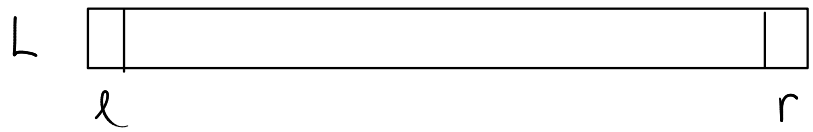
    Return „Not found"

Invariant I:

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

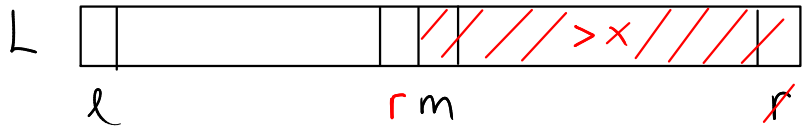


Første gang vi kommer til *I*, er det trivielt sandt:



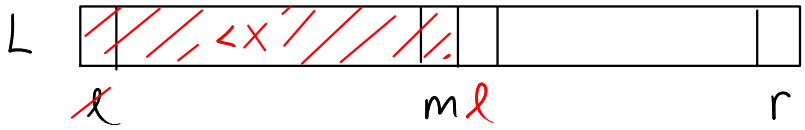
Derefter checker vi, om $x \neq L[m]$.
Hvis det er tilfældet er der to muligheder:

- $x < L[m]$
  I dette tilfælde opdateres r:



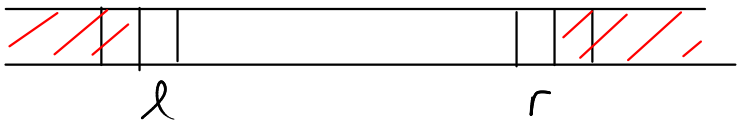
  Herefter er I stadig opfyldt

- $x > L[m]$
  I dette tilfælde opdateres l:

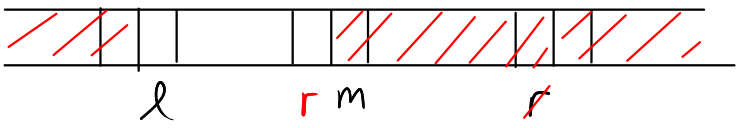


  Herefter er I stadig opfyldt

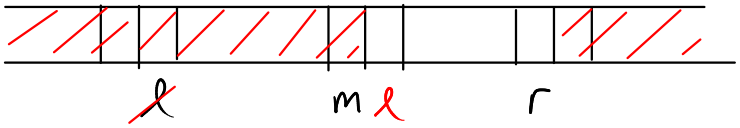
Generelt:
Lige inden vi checker $l \leq r$ and $L[m] \neq x$, gælder I:



Efter gennemløb af løkken gælder I stadig:



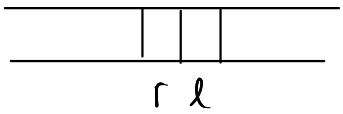
eller



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

- `L[m] = X`
  I dette tilfælde returneres i, og det er indeks til en plads i L, hvor man kan finde X.
  D.v.s. korrekt output i dette tilfælde.

- `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. også i dette tilfælde korrekt output.

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

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

```
BinarySearchRec(L, x, l, r)
If l ≤ r
    m = ⌊(l+r)/2⌋
    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 $L[1] = 2$

    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"

...