

# Introduktion

---

- Algoritmer og datastrukturer
- Toppunkter
  - Algoritme 1
  - Algoritme 2
  - Algoritme 3

# Introduktion

---

- Algoritmer og datastrukturer
- Toppunkter
  - Algoritme 1
  - Algoritme 2
  - Algoritme 3

# Algoritmer og datastrukturer

---

- **Algoritmisk problem.** Præcist defineret relation mellem input og output.
- **Algoritme.** Metode til at løse et algoritmisk problem.
  - Beskrevet i **diskrete** og **entydige** skridt.
  - Matematisk abstraktion af program.
- **Datastruktur.** Metode til at organisere data så det kan søges i eller manipuleres.

# Eksempel: Maksimalt tal

---

- **Maksimalt tal.** Givet en tabel  $A[0..n-1]$ , find et tal  $i$ , således at  $A[i]$  er maksimalt.
  - **Input.** Tabel  $A[0..n-1]$ .
  - **Output.** Et tal  $i$ ,  $0 \leq i < n$ , så  $A[i] \geq A[j]$  for alle indgange  $j \neq i$ .
- **Algoritme.**
  - Gennemløb  $A$  og vedligehold indeks af nuværende maksimale indgang.
  - Returner indeks.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	7	15	17	11	2	3	6	8	7	5	9	5	23

# Beskrivelse af algoritmer

---

- Naturligt sprog.

- Gennemløb A og vedligehold indeks af nuværende maksimale indgang.
- Returner indeks.

- Program.

```
public static int findMax(int[] A) {  
    int max = 0;  
    for(i = 0; i < A.length; i++)  
        if (A[i] > A[max]) max = i;  
    return max;  
}
```

- Pseudokode.

```
FINDMAX(A, n)  
    max = 0  
    for i = 0 to n-1  
        if (A[i] > A[max]) max = i  
    return max
```

# Introduktion

---

- Algoritmer og datastrukturer
- **Toppunkter**
  - Algoritme 1
  - Algoritme 2
  - Algoritme 3

# Toppunkter

---

- **Toppunkt.** Indgang  $A[i]$  er et **toppunkt** (peak point) hvis  $A[i]$  er mindst ligeså stort som dets **naboer**:
  - $A[i]$  toppunkt hvis  $A[i-1] \leq A[i] \geq A[i+1]$  for  $i \in \{1, \dots, n-2\}$
  - $A[0]$  toppunkt  $A[0] \geq A[1]$  og  $A[n-1]$  er toppunkt hvis  $A[n-2] \leq A[n-1]$ . (Tænk  $A[-1] = A[n] = -\infty$ ).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	7	15	17	11	2	3	6	8	7	5	9	5	23

- **Toppunktsproblem.** Givet en tabel  $A[0..n-1]$ , find *et* tal  $i$ , således at  $A[i]$  toppunkt.
  - **Input.** En tabel  $A[0..n-1]$ .
  - **Output.** Et tal  $i$ ,  $0 \leq i < n$ , så  $A[i]$  er et toppunkt.

# Introduktion

---

- Algoritmer og datastrukturer
- Toppunkter
  - **Algoritme 1**
  - Algoritme 2
  - Algoritme 3



# Algoritme 1

---

- **Algoritme 1.** For hver indgang i A, check om den er et toppunkt. Returner indeks på første toppunkt.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	7	15	17	11	2	3	6	8	7	5	9	5	23

- **Pseudokode.**

```
TOPPUNKT1(A, n)
  if A[0] ≥ A[1] return 0
  for i = 1 to n-2
    if A[i-1] ≤ A[i] ≥ A[i+1] return i
  if A[n-2] ≤ A[n-1] return n-1
```

- **Udfordring.** Hvordan analyserer vi algoritmen?

# Teoretisk analyse

---

- **Køretid/tidskompleksitet (running time/time complexity).**
  - $T(n)$  = antallet af **skridt** som algoritmen udfører på et input af størrelse  $n$ .
- **Skridt.**
  - Læsning/skrivning til hukommelse ( $x := y$ ,  $A[i]$ ,  $i = i + 1$ , ...)
  - Arithmetiske/boolske operationer ( $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ ,  $\&\&$ ,  $\|\|$ ,  $\&$ ,  $|$ ,  $\wedge$ ,  $\sim$ )
  - Sammenligninger ( $<$ ,  $>$ ,  $=<$ ,  $=>$ ,  $=$ ,  $\neq$ )
  - Programflow (if-then-else, while, for, goto, funktionskald, ..)
- **Værstefaldstidskompleksitet (worst-case complexity).**
  - Interessert (næsten altid) i **værstefaldstidskompleksitet** = maksimal køretid over alle input af størrelse  $n$ .

# Teoretisk analyse

---

- **Køretid.** Hvad er køretiden  $T(n)$  for algoritmen?

```
TOPPUNKT1(A, n)
  if A[0] ≥ A[1] return 0
  for i = 1 to n-2
    if A[i-1] ≤ A[i] ≥ A[i+1] return i
  if A[n-2] ≤ A[n-1] return n-1
```

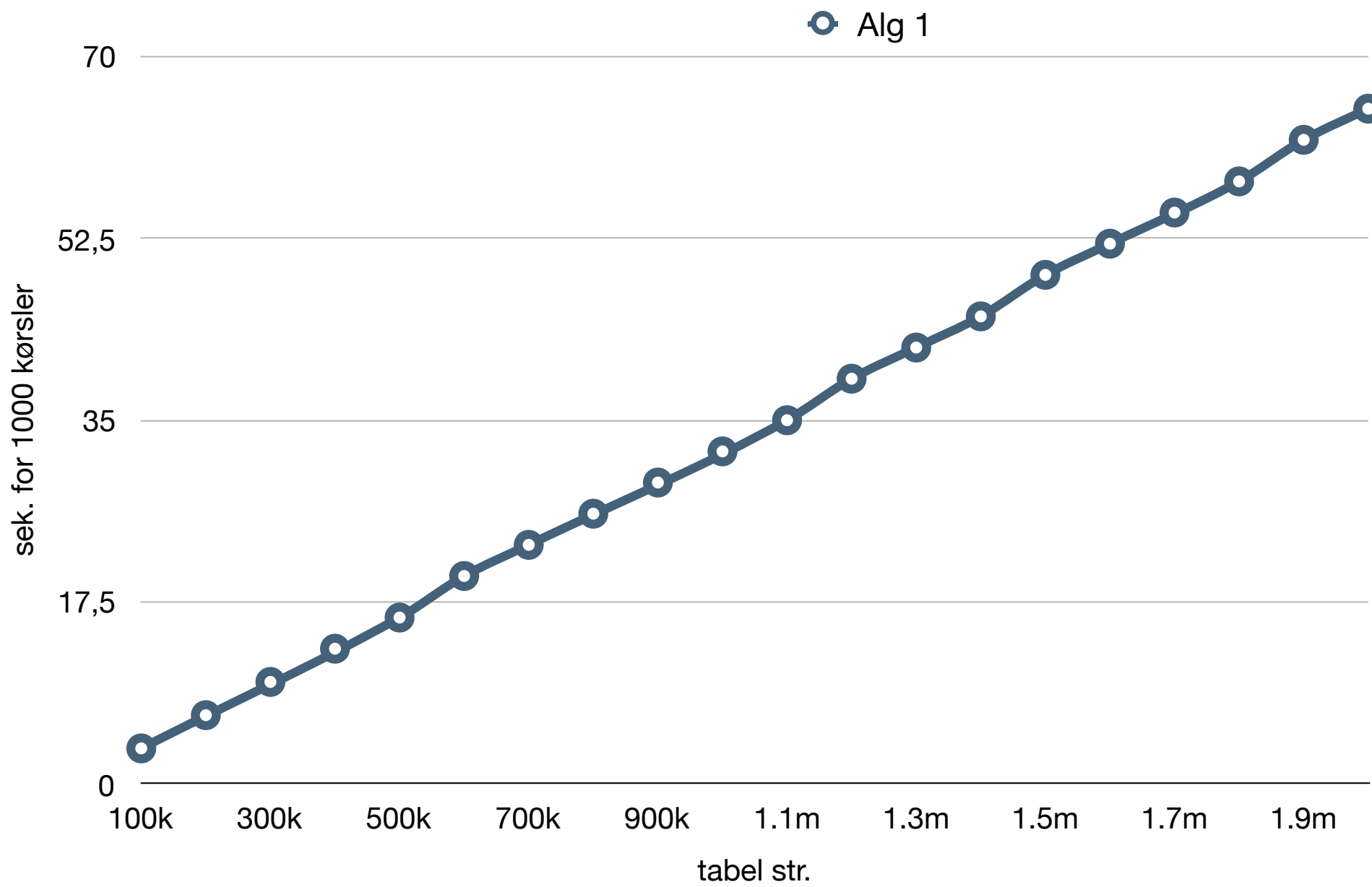
$c_1$

$(n-2) \cdot c_2$

$c_3$

$$T(n) = c_1 + (n-2) \cdot c_2 + c_3$$

- $T(n)$  er en *lineær funktion* af  $n$ :  $T(n) = an + b$ , for passende konstanter  $a$  og  $b$ .
- **Asymptotisk notation.**  $T(n) = \Theta(n)$
- **Eksperimentiel analyse.**
  - Hvad er køretid af algoritmen i praksis?
  - Hvordan passer den teoretisk analyse med praksis?



# Toppunkter

---

- Algoritme 1 finder et toppunkt i  $\Theta(n)$  tid.
- Stemmer overens med praksis.
- **Udfordring.** Kan vi gøre det bedre?

# Introduktion

---

- Algoritmer og datastrukturer
- Toppunkter
  - Algoritme 1
  - **Algoritme 2**
  - Algoritme 3

# Algoritme 2

---

- **Observation.** Et (globalt) maksimalt tal i A er et toppunkt.
- **Algoritme 2.** Find et maksimalt tal i A med `FINDMAX(A, n)`.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	7	15	17	11	2	3	6	8	7	5	9	5	23

```
FINDMAX(A, n)
  max = 0
  for i = 0 to n-1
    if (A[i] > A[max]) max = i
  return max
```

# Teoretisk analyse

---

- **Køretid.** Hvad er køretiden  $T(n)$  for algoritmen?

```

FINDMAX(A, n)
  max = 0
  for i = 0 to n-1
    if (A[i] > A[max]) max = i
  return max

```

$C_4$

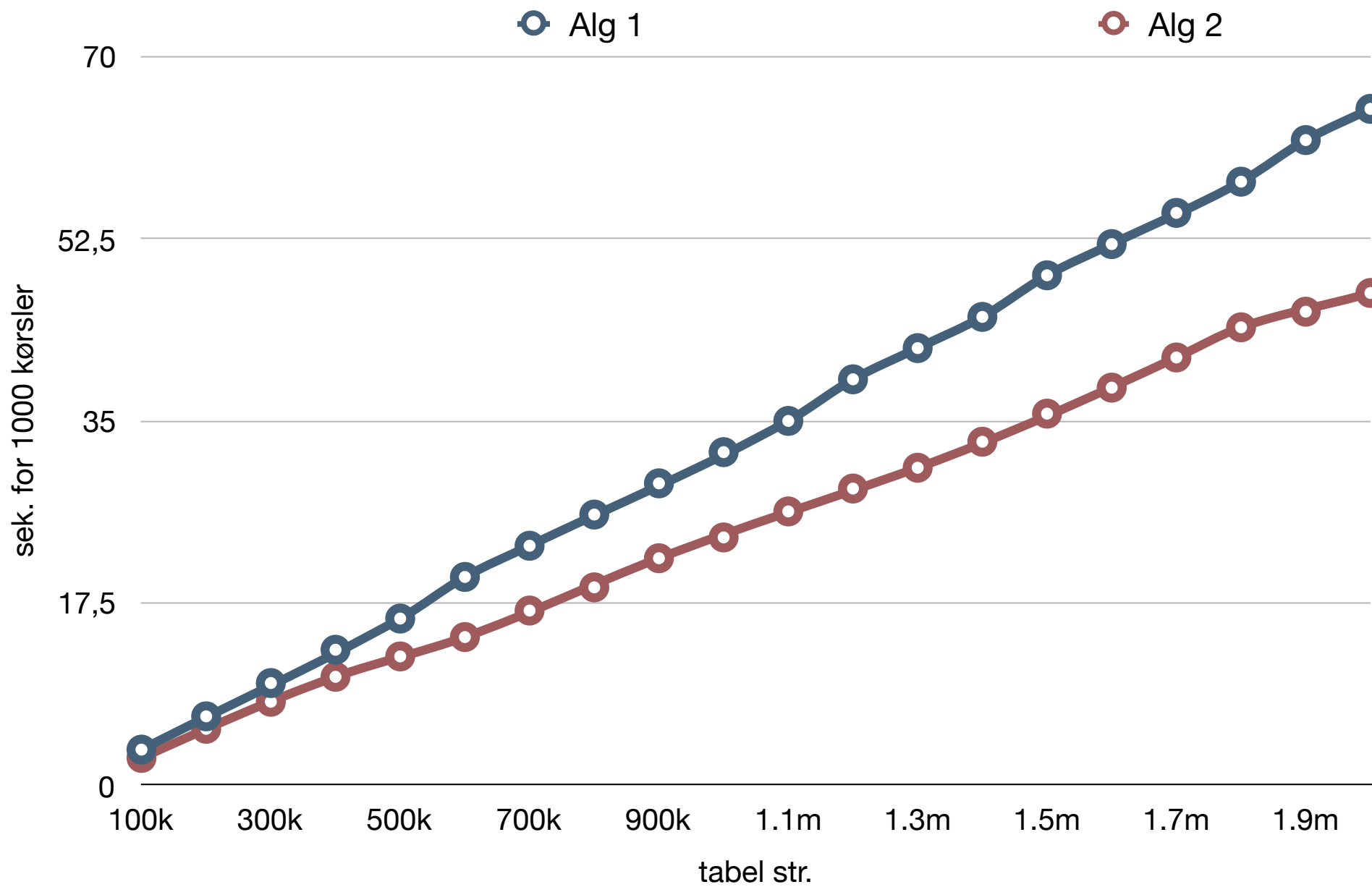
$n \cdot C_5$

$C_6$

$$T(n) = c_4 + n \cdot c_5 + c_6 = \Theta(n)$$

- **Ekperimentiel analyse.** Bedre konstanter?





# Toppunkter

---

- Teoretisk

- Algoritme 1 og 2 finder et toppunkt i  $\Theta(n)$  tid.

- Eksperimentielt

- Algoritme 1 og 2 kører i  $\Theta(n)$  tid i praksis.
- Algoritme 2 er en konstant faktor hurtigere end algoritme 1.

- Udfordring. Kan vi gøre det **betydeligt** bedre?

# Introduktion

---

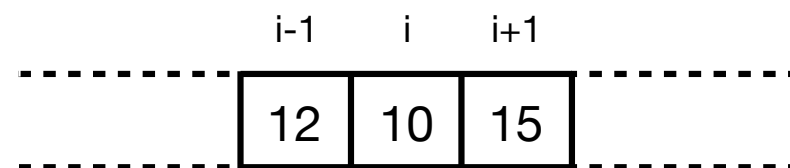
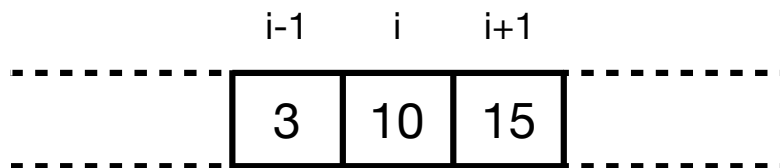
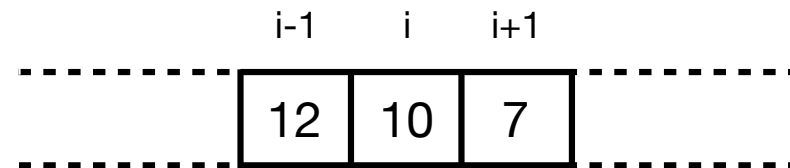
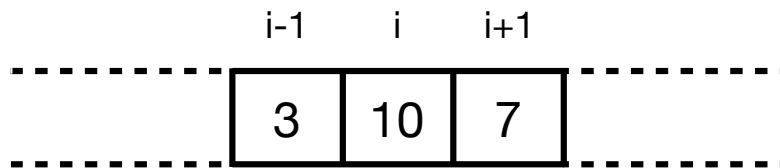
- Algoritmer og datastrukturer
- Toppunkter
  - Algoritme 1
  - Algoritme 2
  - Algoritme 3

# Algoritme 3

---

- **Snedig ide.**

- Kig på en vilkårlig indgang  $A[i]$  og dens naboer  $A[i-1]$  og  $A[i+1]$ .
- Hvor kan vi finde et toppunkt ifht.  $A[i]$ ?
  - Naboer er  $\leq A[i] \Rightarrow A[i]$  er toppunkt.
  - Ellers er  $A$  voksende i **mindst** en retning  $\Rightarrow$  der findes toppunkt i den retning.



- **Udfordring.** Hvordan kan vi bruge ide til at lave en effektiv algoritme?

# Algoritme 3

---

- Algoritme 3.

- Kig på **midterste** indgang  $A[m]$  og naboer  $A[m-1]$  og  $A[m+1]$ .
- Hvis  $A[m]$  er toppunkt, returner  $m$ .
- Ellers fortsæt søgning **rekursivt** i halvdel af tabel hvor nabo er større end  $A[m]$ .

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	7	15	17	11	2	3	6	8	7	5	9	5	23

# Algoritme 3

---

- Algoritme 3.

- Kig på **midterste** indgang  $A[m]$  og naboer  $A[m-1]$  og  $A[m+1]$ .
- Hvis  $A[m]$  er toppunkt, returner  $m$ .
- Ellers fortsæt søgning **rekursivt** i halvdel af tabel hvor nabo er større end  $A[m]$ .

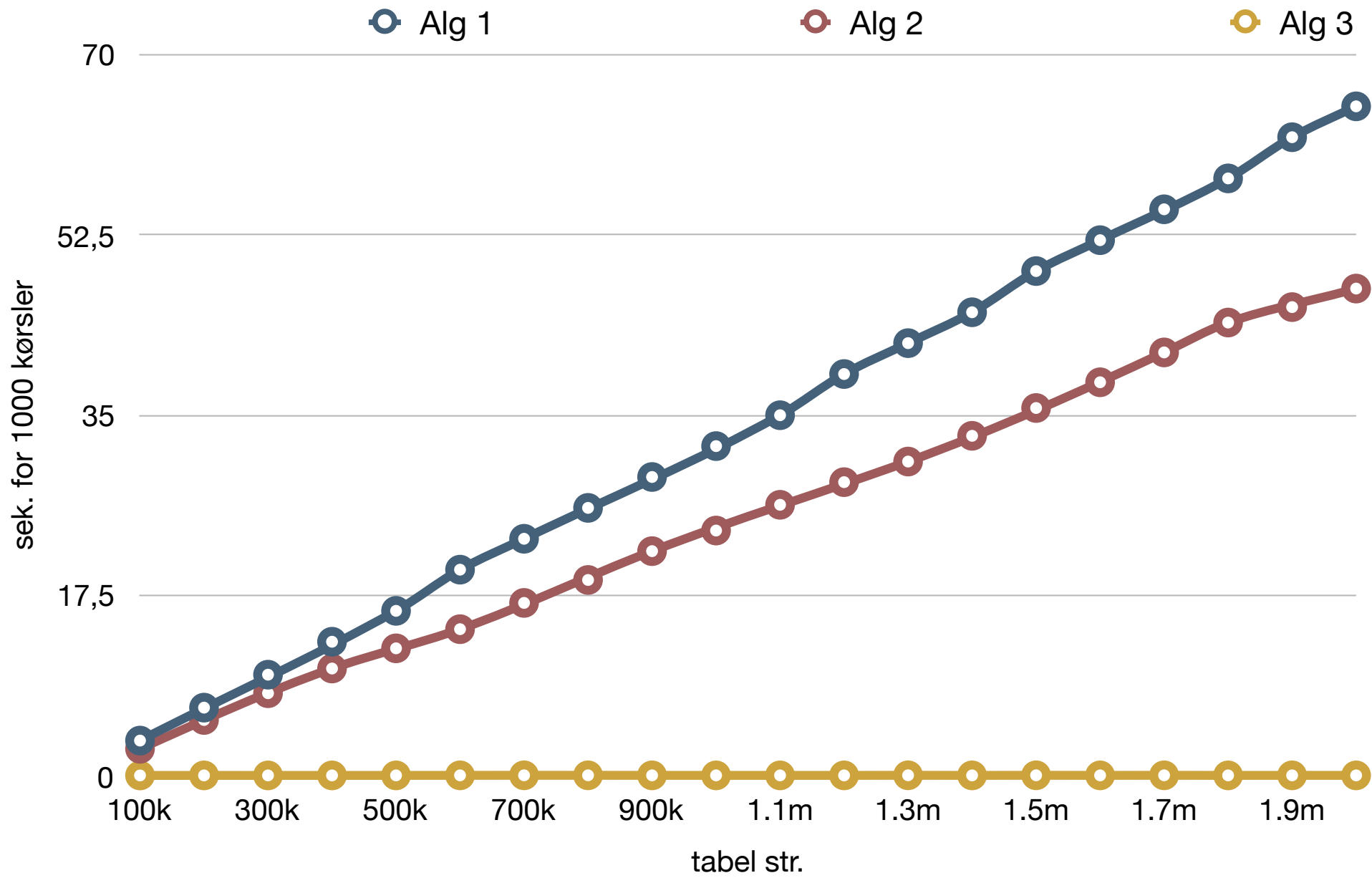
```
TOPPUNKT3(A,i,j)
  m = ⌊(i+j)/2⌋
  if A[m] ≥ naboer return m
  elseif A[m-1] > A[m]
    return TOPPUNKT3(A,i,m-1)
  elseif A[m] < A[m+1]
    return TOPPUNKT3(A,m+1,j)
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	7	15	17	11	2	3	6	8	7	5	9	5	23

# Algoritme 3

- **Køretid.**
- Et rekursivt kald tager konstant tid.
- Hvor mange rekursive kald laver vi?
- Et rekursivt kald **halverer** størrelsen af tabellen vi kigger på. Vi stopper når tabellen har størrelse 1.
  - 1. rekursive kald:  $n/2$
  - 2. rekursive kald:  $n/4$
  - ....
  - $k$ 'te. rekursive kald:  $n/2^k$
  - ....
- $\Rightarrow$  Efter  $\sim \log_2 n$  rekursive kald har tabellen størrelse  $\leq 1$ .
- $\Rightarrow$  Køretiden er  $\Theta(\log n)$
- **Ekperimentiel analyse.** Betydeligt bedre?

```
TOPPUNKT3(A, i, j)
  m =  $\lfloor (i+j)/2 \rfloor$ 
  if A[m]  $\geq$  naboer return m
  elseif A[m-1] > A[m]
    return TOPPUNKT3(A, i, m-1)
  elseif A[m] < A[m+1]
    return TOPPUNKT3(A, m+1, j)
```





# Toppunkter

---

- Teoretisk

- Algoritme 1 og 2 finder et toppunkt i  $\Theta(n)$  tid.
- Algoritme 3 finder et toppunkt i  $\Theta(\log n)$  tid.

- Eksperimentielt

- Algoritme 1 og 2 kører i  $\Theta(n)$  tid i praksis.
- Algoritme 2 er en konstant faktor hurtigere end algoritme 1.
- Algoritme 3 er meget, meget hurtigere end algoritme 1 og 2.

# Introduktion

---

- Algoritmer og datastrukturer
- Toppunkter
  - Algoritme 1
  - Algoritme 2
  - Algoritme 3