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 et program.
- Datastruktur Metode til at organisere data, så man let kan ændre, analysere, eller søge i data.

Eksempel: Maksimalt tal.

- Maksimalt tal. Givet et array `A[0..n-1]` find `i` så `A[i]` er maksimalt.
  - Input: Array `A[0..n-1]`.
  - Output: Et tal, `i`, $0 \leq$ `i` $< n$, så for $j \neq i$ gælder `A[j]` $\leq$ `A[i]`.
- Algoritme:
  - Gennemløb A og vedligehold index af hidtil største indgang.
  - Returner index.

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

Beskrive algoritmer

- Naturligt sprog.
  - Gennemløb A og vedligehold index af hidtil største indgang.
  - Returner index.
- Program.

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

- Pseudokode.

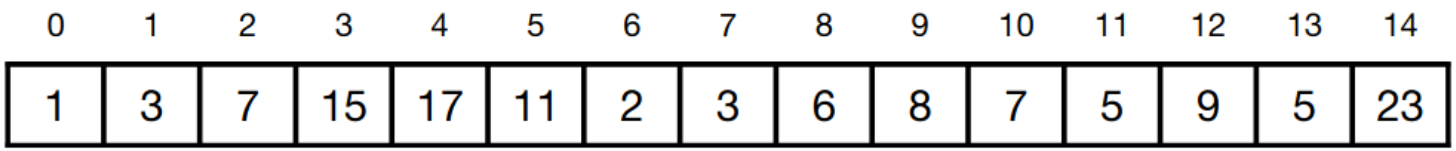
```
FindMax(A, n)
    max = 0
    for i = 0 to n-1
        if (A[i] > A[max]) max = i
    return max
```

Introduktion

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

Toppunkter

- Toppunkt. Element A[i] er toppunkt, hvis det er større end sine naboer.
  - Hvis `i-1` er en indgang i `A`, så er `A[i-1]` $\leq$ `A[i]`,
  - Hvis `i+1` er en indgang i `A`, så er `A[i+1]` $\leq$ `A[i]`,



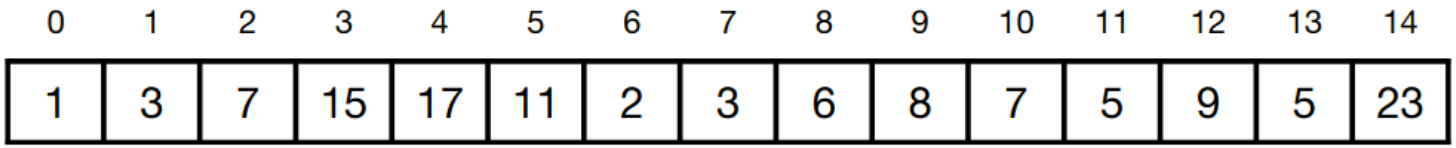
- Toppunktsproblem. Givet et array `A`, find et index i hvor `A[i]` er toppunkt.
  - Input. Array `A` af længde $n \geq 1$.
  - Output. Et index `i` så `A[i]` er toppunkt.

Introduktion

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

Algoritme 1

- Algoritme 1. Gennemløb `A` og check for hvert punkt, om det er et toppunkt. Returner index af første toppunkt.



- Pseudokode:

```
PEAK1(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
```

- Spørgsmål: hvor hurtig er algoritme 1?

Teoretisk analyse

- Kørselstid/køretid.

  Antallet af skridt udført af algoritmen som funktion af inputstørrelse.
- Skridt:
  - Tilgå hukommelse (læse/skrive), f.eks. `x=y;`, `i=i+1;`, `A[i]`, ...
  - Aritmetiske/boolske operationer, såsom `+`,`-`,`*`,`/`,`&&`,`||`,`&`,`|`,`^`,`>>`,`<<` ...
  - Sammenligninger, såsom $<$, $>$, $=$, $\leq$, $\geq$, $\neq$, ...
  - Programflow, såsom if-then-else, while, for, goto, funktionskald.
- Værstefaldskompleksitet (worst-case complexity)
  - Maksimal køretid for en given inputstørrelse.

Teoretisk analyse

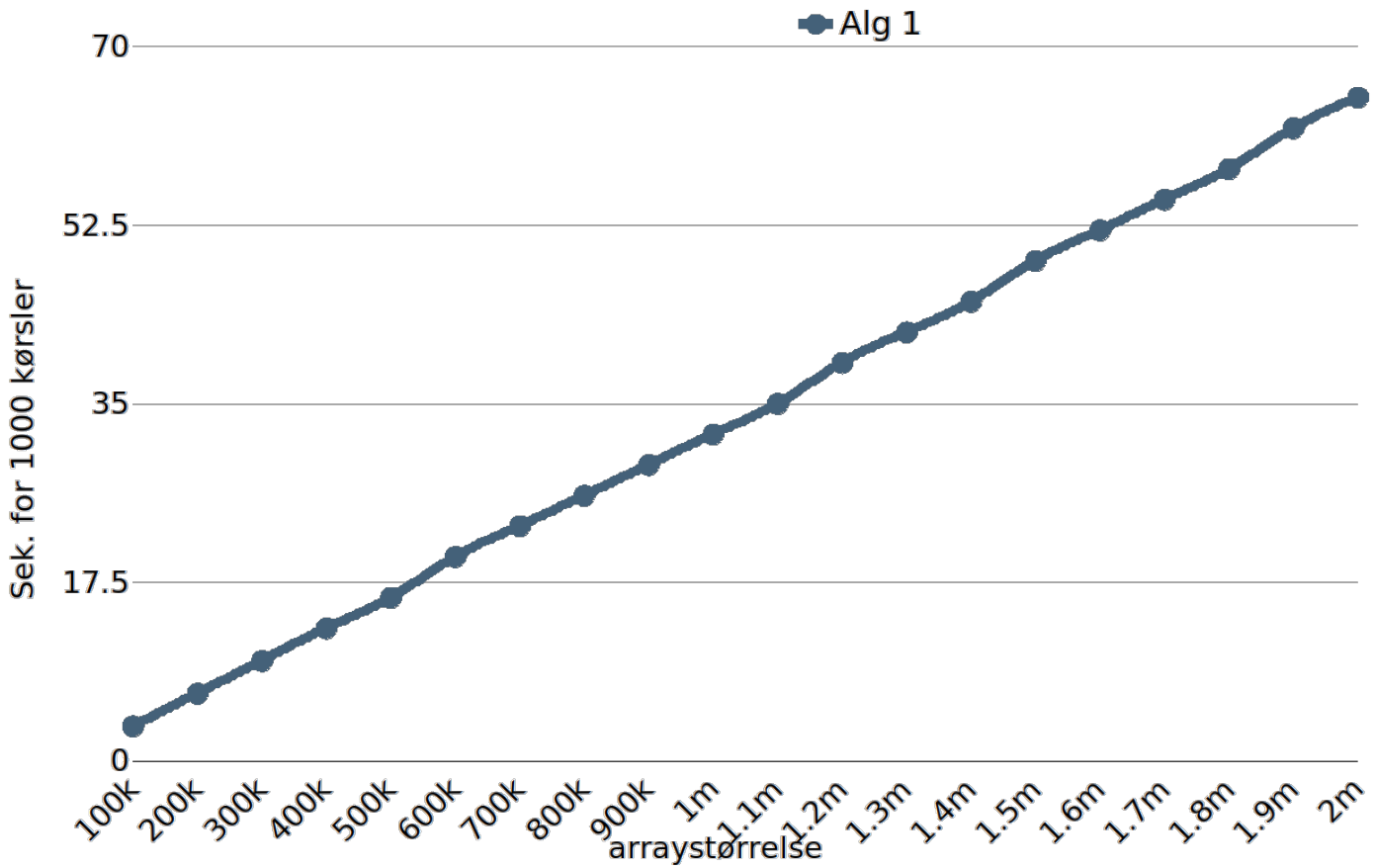
- Kørselstid. Hvor lang tid tager programmet i værste fald?

```
PEAK1(A, n)
    if A[0] ≥ A[1] return 0                          c1 +
    for i = 1 to n-2                                 (n-2) x
        if A[i-1] ≤ A[i] ≥ A[i+1] return i              c2
    if A[n-2] ≤ A[n-1] return n-1                    +c3
```

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

- $T(n)$ er en lineær funktion af n. $T(n) = a \cdot n + b$, hvor a, b konstant.
- Asymptotisk notation: $T(n)$ er $O(n)$.
- Eksperimentel analyse.
  - Hvor hurtigt kører algoritmen i praksis?
  - Hvordan passer den teoretiske analyse med praksis?

Eksperimentel analyse



Toppunkter, algoritme 1, konklusion

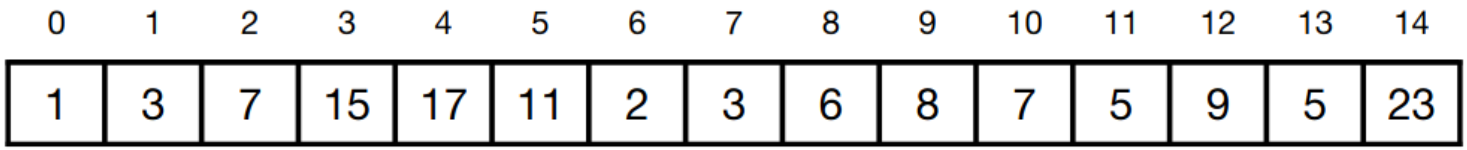
- Algoritme 1 finder toppunkter.
  - Teoretisk analyse: $\Theta(n)$ tid.
  - Eksperimentel analyse: $\Theta(n)$ tid.
- Algoritmisk udfordring: kan vi gøre det hurtigere?

Introduktion

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

Toppunkter: Algoritme 2

- Observer: Et maksimalt element i hele A er et toppunkt.
- Algoritme 2: Find et maksimalt element i A ved at kalde `FindMax(A)`.




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

- Hvor lang tid tager Algoritme 2?

Teoretisk analyse

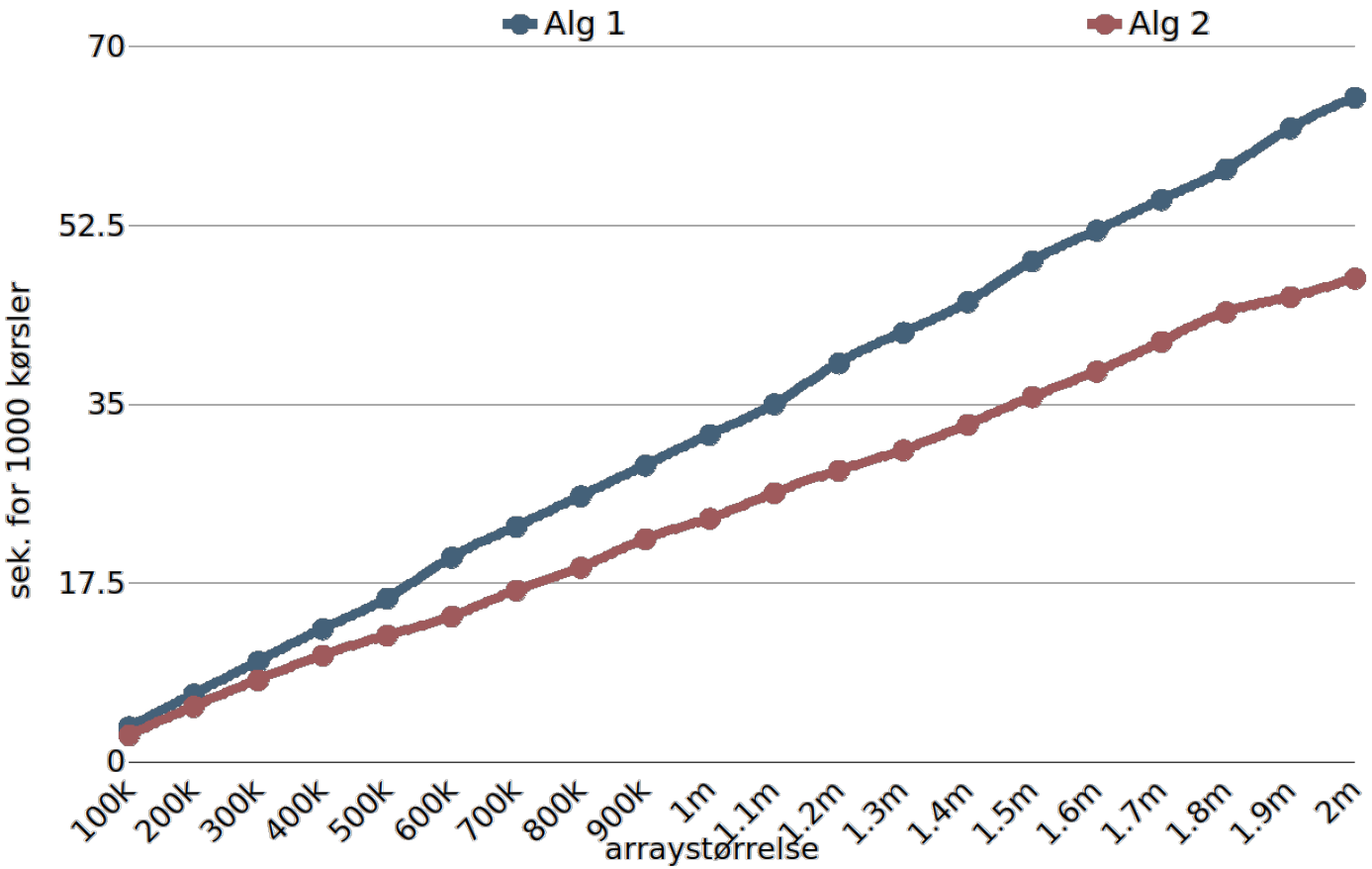
- Kørselstid. Hvor lang tid tager programmet i værste fald?

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

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

- Teoretisk analyse? $T(n)$ er $\Theta(n)$. Tiden er *lineær* i n.
- Eksperimentel analyse?

Eksperimentel analyse



Toppunkter, algoritme 1 og 2, konklusion

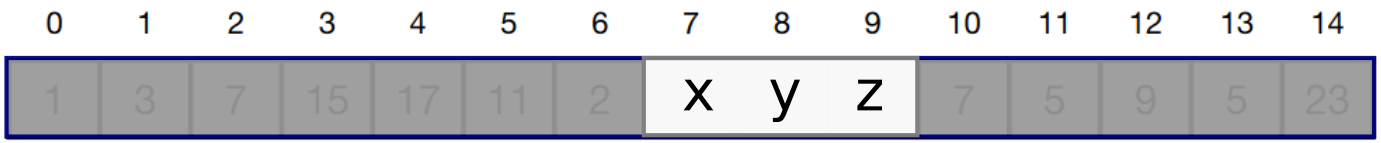
- Algoritme 1 og 2 finder toppunkter.
  - Teoretisk analyse: $\Theta(n)$ tid.
  - Eksperimentel analyse: $\Theta(n)$ tid.
  - Eksperimentel analyse: Algoritme 2 er en konstant faktor hurtigere.
- Algoritmisk udfordring:
  - kan vi gøre det væsentligt hurtigere?

Introduktion

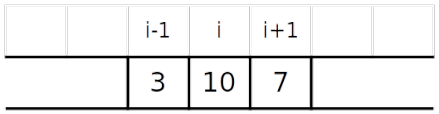
- Algoritmer og datastrukturer
- Toppunkter
  - Algoritme 1
  - Algoritme 2
  - Algoritme 3

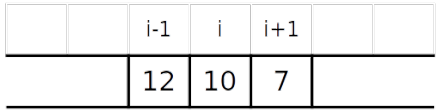
Toppunkter: Algoritme 3

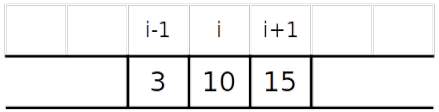
- Snedig indsigt.
  - Kig på (et vilkårligt) `A[i]` og dens naboer

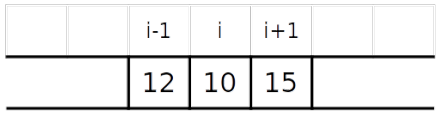


- Forskellige situationer:
  - Hvis `A[i]` selv er toppunkt, er vi færdige.
  - Hvis `A[i+1]` er større end `A[i]`, så vil et toppunkt mellem `i+1` og `n-1` også være et toppunkt for hele `A`.
    Med andre ord: hvis `A[i+1]>A[i]`, findes der et toppunkt mod højre.
  - Ligeledes: Hvis `A[i-1]` er større end `A[i]` findes et toppunkt mod venstre.









- Snedig algoritme?

Toppunkter: Algoritme 3

- Algoritme 3.
  - Kig på midterste indgang `A[m]` og naboer `A[m-1]` og `A[m+1]`
  - Hvis `A[m]` selv er topppunkt returner `m`.
  - Ellers må der være en større nabo.
    fortsæt søgningen rekursivt til den side, hvor den større nabo er.

```
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

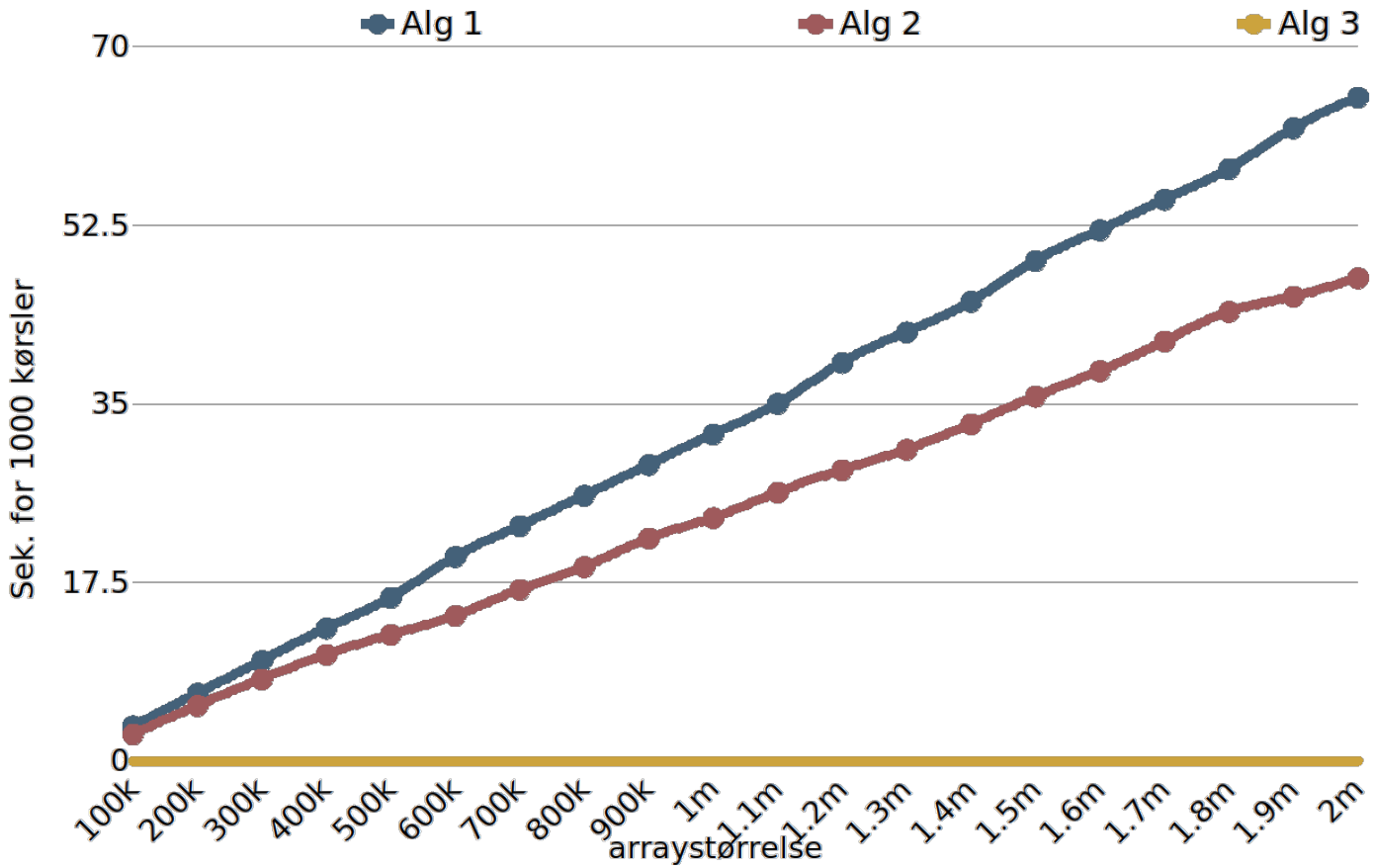
Toppunkter: Algoritme 3 – analyse

- For hvert rekursivt kald, bruger vi konstant tid.
- Hvor mange rekursive kald?
- Hver gang, halverer vi området, vi kigger i.

```
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)
```

- Vi stopper senest, når det aktive område indeholder 1 element.
  - Først kaldes toppunkt med n aktive elementer,
  - Så kaldes funktionen rekursivt med n/2 aktive elementer,
  - Så med n/4 elementer i det aktive område,
  - ...
  - k'te rekursive kald: $n/2^k$ aktive elementer,
- hvis vi sætter $k = \lceil \log_2 n \rceil$, kun ≤ 1 aktive elementer, toppunkt fundet.
- $\Rightarrow$ Kørselstiden er $\Theta(\log n)$.
- Eksperiment?

Eksperimentel analyse



Toppunkter, konklusion

- Algoritmerne 1,2 og 3 finder toppunkter.
  - Algoritmerne 1 og 2 tager $\Theta(n)$ tid i værste fald,
  - Algoritme 3 bruger $\Theta(\log n)$ tid.
  - Algoritmerne 1 og 2 bruger lineær tid i praksis,
  - Algoritme 2 er i praksis en konstant faktor hurtigere end algoritme 1,
  - Algoritme 3 er i praksis meget, meget hurtigere end 1 og 2.

Introduktion

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