# Brug for algoritmikere

## Danmark vil mangle 22.000 it-professionelle i 2030

Manglen på it-folk fortsætter så langt øjet rækker, og vil være næsten tre gange så høj om ti år

## Ny FN-måling: Danmark er fortsat verdensmestre i offentlig digitalisering

12-07-2020 | Strategier

FN har netop offentliggjort E-Government Survey 2020, og Danmark ligger for anden gang i træk på en flot første plads. I 2018 rykkede Danmark fra en tidligere 9. plads til første pladsen.



# DIKU i verdensklasse!

## CSRankings: Computer Science Rankings

CSRankings is a metrics-based ranking of top computer science institutions around the world. **Click on a triangle** (▶) to expand areas or institutions. **Click on a name** to go to a faculty member's home page. **Click on a chart icon** (the after a name or institution) to see the distribution of their publication areas as a bar chart. **Click on a Google Scholar icon** () to see publications, and **click on the DBLP logo** () to go to a DBLP entry. *Applying to grad school? Read this first.* **Do you find CSrankings useful? Sponsor CSrankings on GitHub.**

Rank institutions in the world by publications from 2012 to 2022

**All Areas** [off | on]

**AI** [off | on]

- ▶ Artificial intelligence
- ▶ Computer vision
- ▶ Machine learning & data mining
- ▶ Natural language processing
- ▶ The Web & information retrieval

**Systems** [off | on]

- ▶ Computer architecture
- ▶ Computer networks
- ▶ Computer security
- ▶ Databases
- ▶ Design automation
- ▶ Embedded & real-time systems
- ▶ High-performance computing
- ▶ Mobile computing
- ▶ Measurement & perf. analysis
- ▶ Operating systems
- ▶ Programming languages
- ▶ Software engineering

**Theory** [off | on]

- ▶ Algorithms & complexity ☑
- ▶ Cryptography
- ▶ Logic & verification

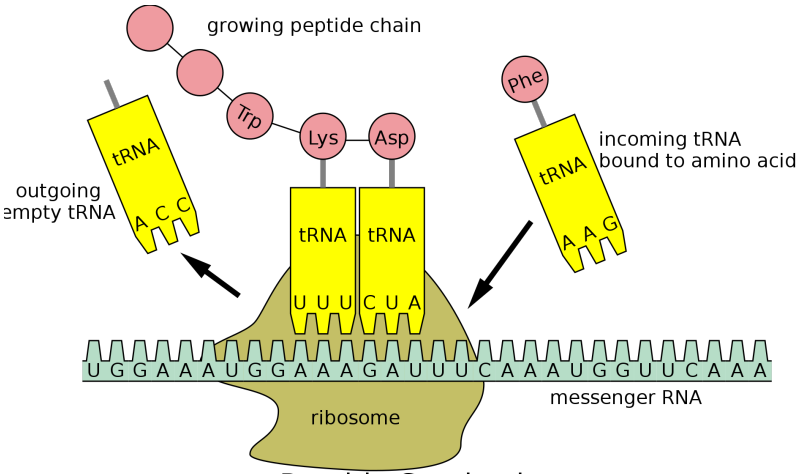
**Interdisciplinary Areas** [off | on]

- ▶ Comp. bio & bioinformatics
- ▶ Computer graphics
- ▶ Economics & computation
- ▶ Human-computer interaction
- ▶ Robotics
- ▶ Visualization

| # | Institution | Count | Faculty |
|---|---|---|---|
| 1 | ▶ Carnegie Mellon University | 94.0 | 19 |
| 2 | ▶ Massachusetts Institute of Technology | 81.3 | 20 |
| 3 | ▶ Tel Aviv University | 61.6 | 20 |
| 4 | ▶ Stanford University | 54.2 | 12 |
| 5 | ▶ University of Michigan | 50.3 | 12 |
| 6 | ▶ Princeton University | 47.7 | 11 |
| 7 | ▶ Columbia University | 46.1 | 12 |
| 8 | ▶ Univ. of Illinois at Urbana-Champaign | 40.8 | 9 |
| 9 | ▶ University of Washington | 39.7 | 11 |
| 10 | ▶ Univ. of California - San Diego | 37.1 | 11 |
| 11 | ▶ Weizmann Institute of Science | 35.2 | 10 |
| 12 | ▶ University of Copenhagen | 35.0 | 8 |
| 13 | ▶ Univ. of California - Berkeley | 34.1 | 14 |
| 14 | ▶ EPFL | 32.9 | 12 |
| 15 | ▶ University of Waterloo | 32.6 | 16 |
| 16 | ▶ University of Warsaw | 28.0 | 17 |
| 17 | ▶ University of Pennsylvania | 27.9 | 10 |
| 18 | ▶ Bar-Ilan University | 27.6 | 8 |
| 19 | ▶ Cornell University | 25.3 | 15 |
| 20 | ▶ Rutgers University | 25.1 | 9 |
| 21 | ▶ Hebrew University of Jerusalem | 24.8 | 15 |
| 22 | ▶ University of Texas at Austin | 24.2 | 10 |
| 23 | ▶ Univ. of California - Los Angeles | 24.0 | 4 |

# Algoritmer før (moderne) computere

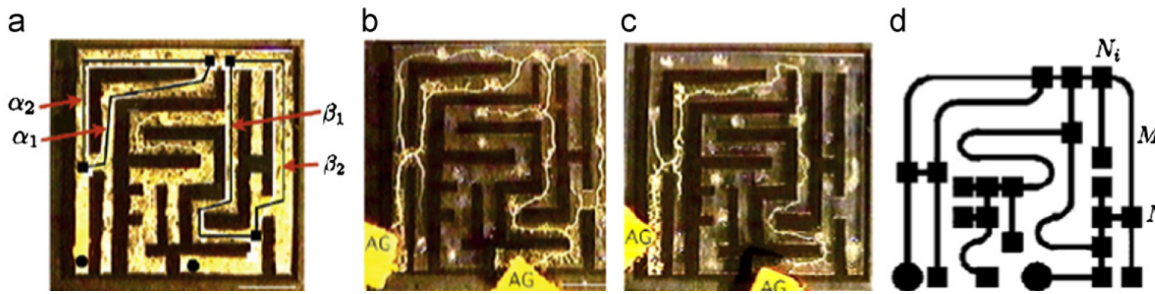
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."



Peptide Synthesis

**A Biological Solution to a Fundamental Distributed Computing Problem**

Yehuda Afek,[1]* Noga Alon,[1,2]* Omer Barad,[3]* Eran Hornstein,[3] Naama Barkai,[3]† Ziv Bar-Joseph[4]†



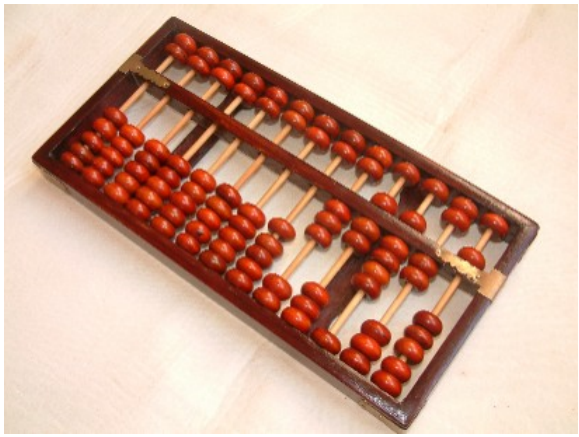

Physarum can compute shortest paths ☆

Vincenzo Bonifaci [a,*,1], Kurt Mehlhorn [b], Girish Varma [c,1]

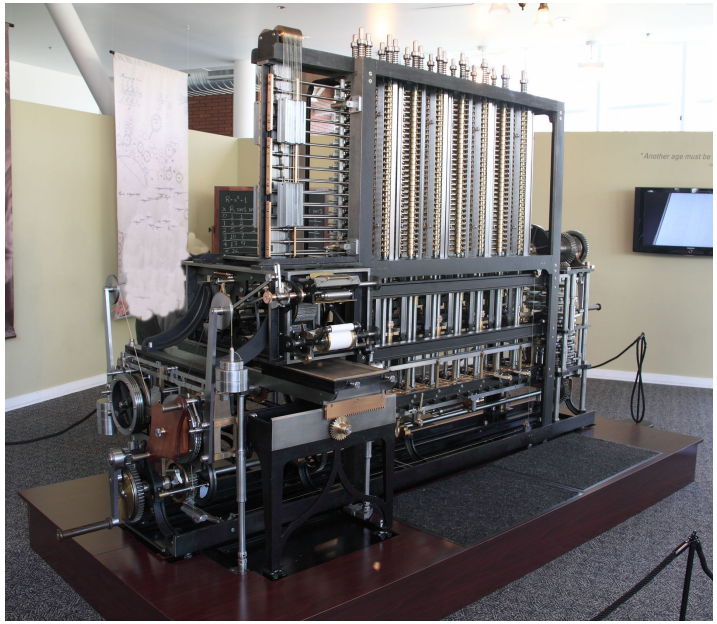
[a] *Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti" – CNR, Rome, Italy*
[b] *MPI for Informatics, Saarbrücken, Germany*
[c] *Tata Institute of Fundamental Research, Mumbai, India*

# Algoritmer før (moderne) computere

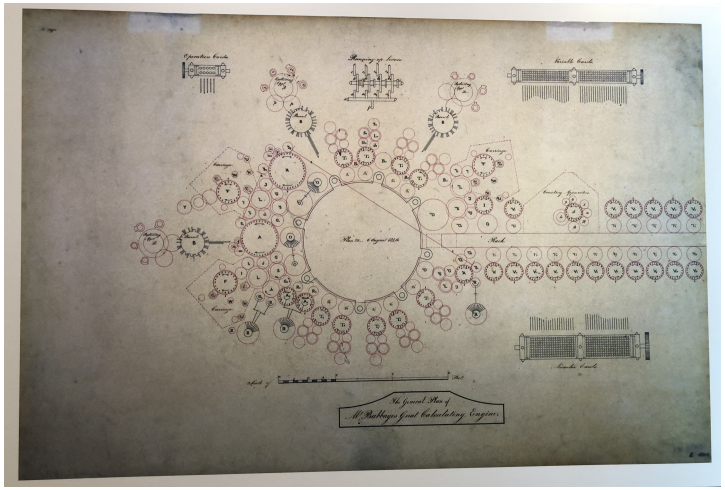
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."




By Ezrdr - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=21954723



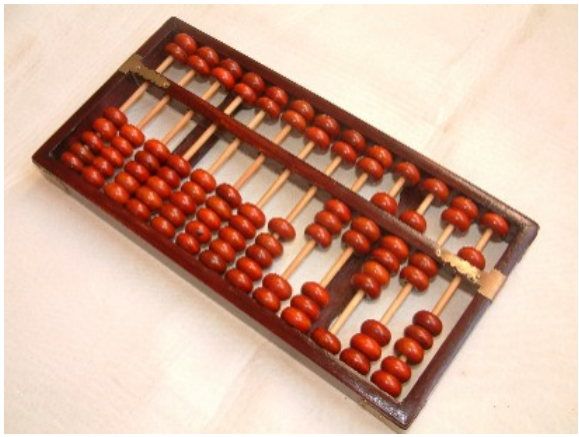
By Canticle at English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10532728



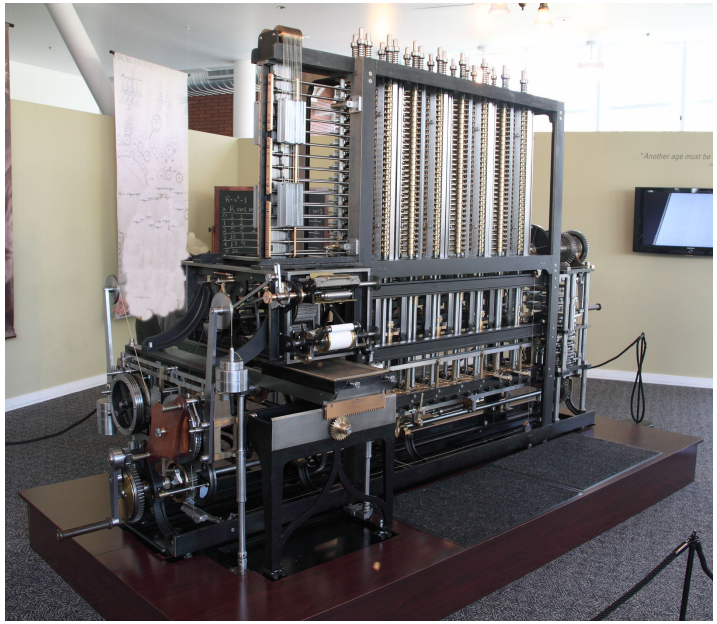
Charles Babbage: The Analytical Engine

# Algoritmer før (moderne) computere

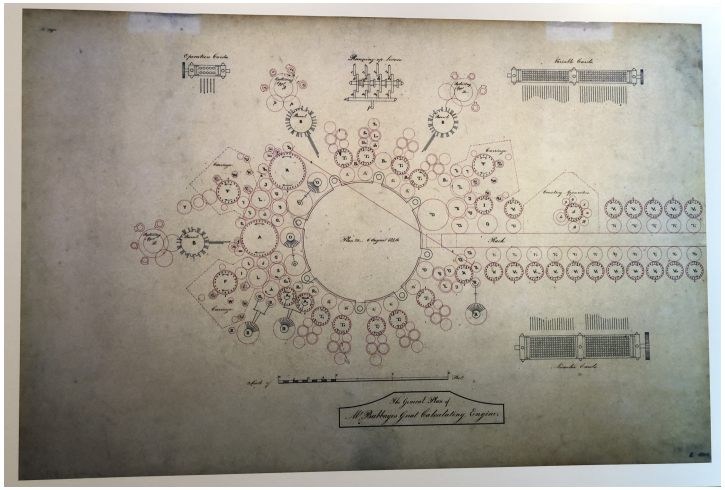
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."



By Ezrdr - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=21954723



By Canticle at English Wikipedia, CB BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10532728

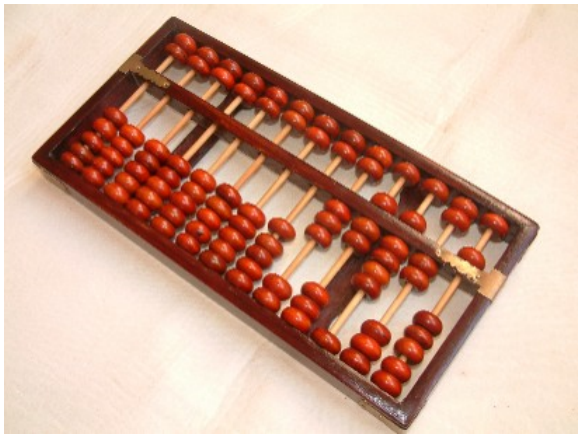


Charles Babbage: The Analytical Engine

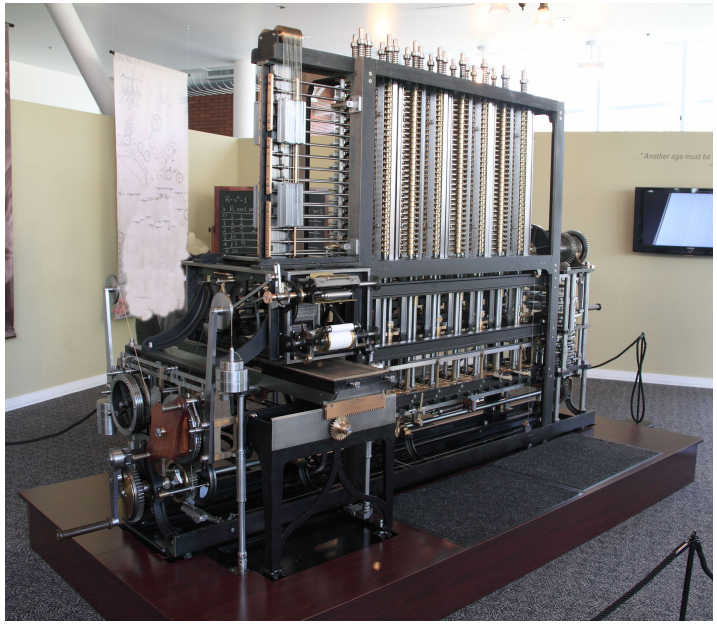
```
 1 9 7 3 4
+5 4 2 8 3
---------
```

# Algoritmer før (moderne) computere

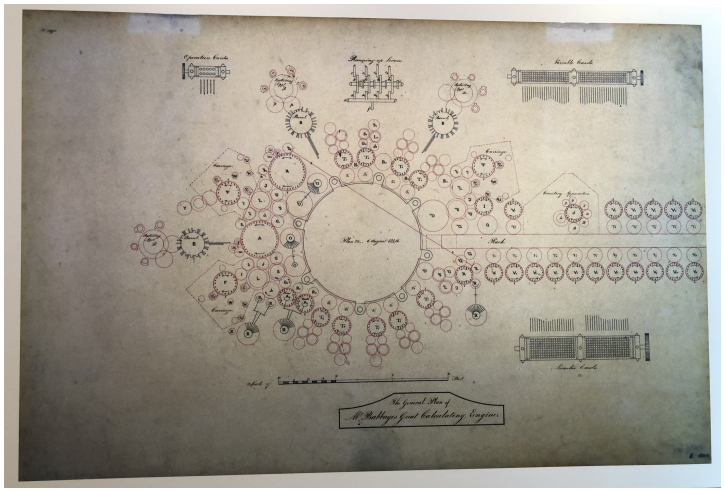
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."



By Ezrdr - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=21954723



By Canticle at English Wikipedia, CB BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10532728

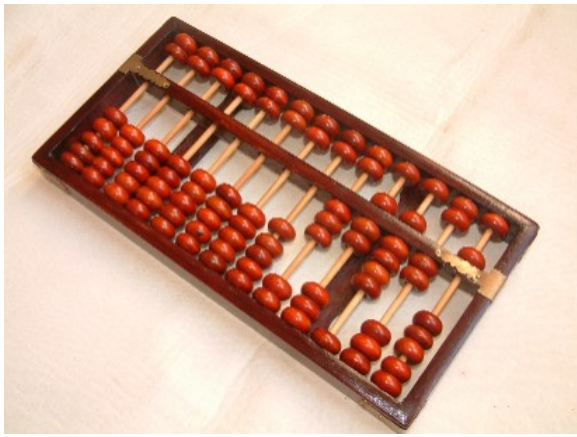


Charles Babbage: The Analytical Engine

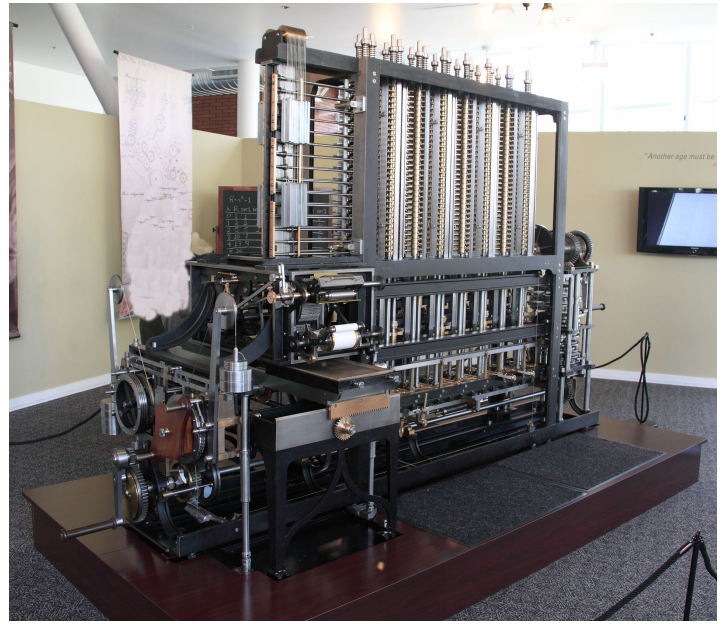
```
 1 9 7 3 4
+5 4 2 8 3
-----------
           7
```

# Algoritmer før (moderne) computere

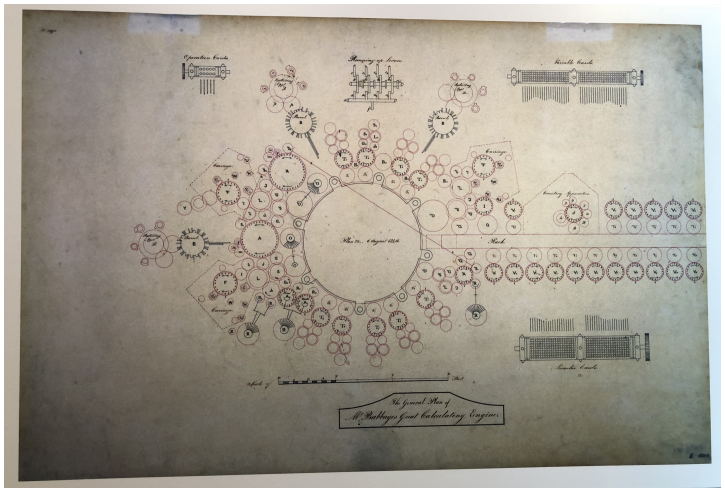
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."



By Ezrdr - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=21954723



By Canticle at English Wikipedia, CB BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10532728

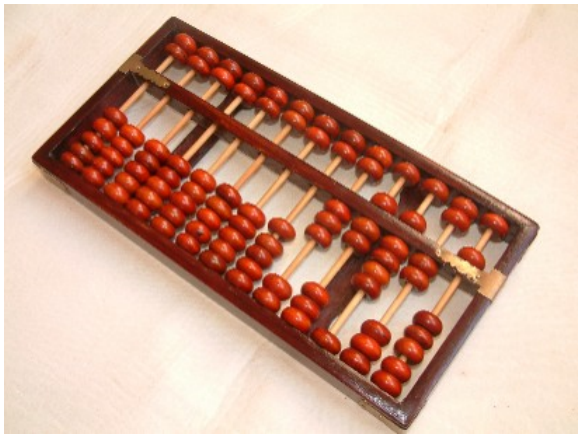


Charles Babbage: The Analytical Engine

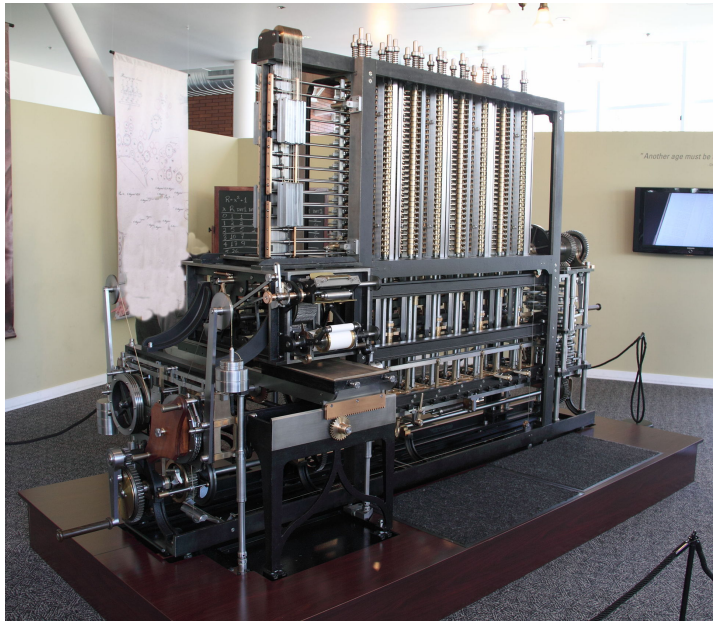
$$\begin{array}{r} \phantom{+}{\scriptstyle 1}\phantom{0000} \\ \phantom{+}19734 \\ +54283 \\ \hline \phantom{+}17 \end{array}$$

# Algoritmer før (moderne) computere

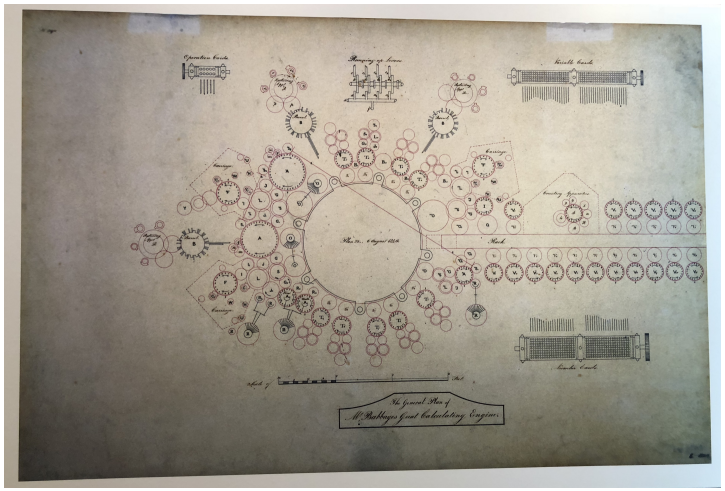
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."



By Ezrdr - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=21954723



By Canticle at English Wikipedia, CB BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10532728

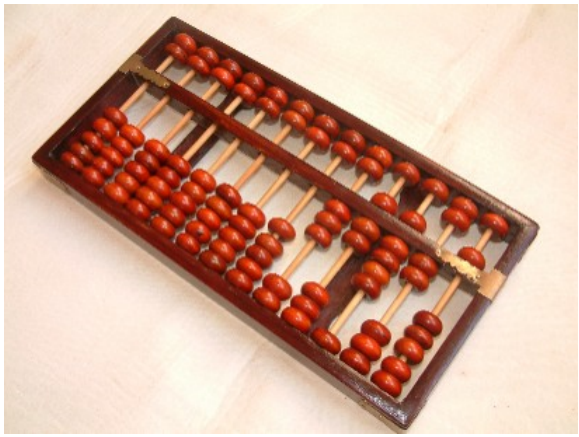


Charles Babbage: The Analytical Engine

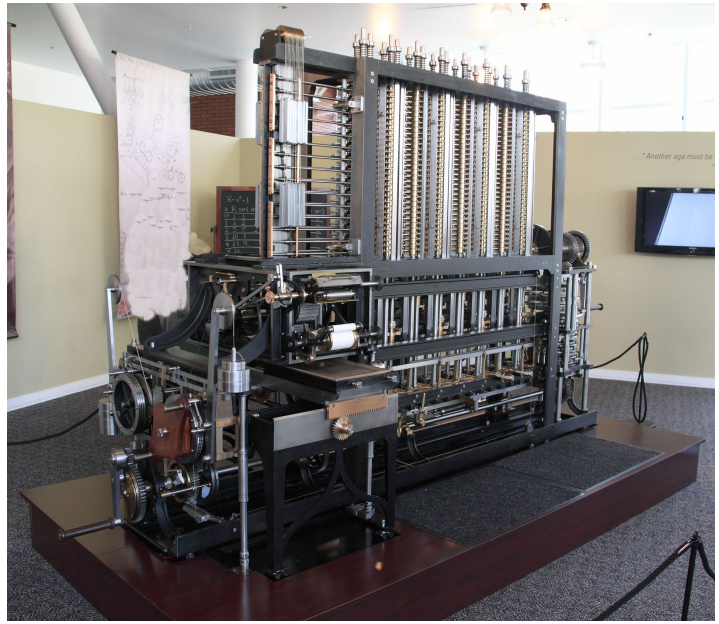
```
  1 1
  1 9 7 3 4
+5 4 2 8 3
-----------
    0 1 7
```

# Algoritmer før (moderne) computere

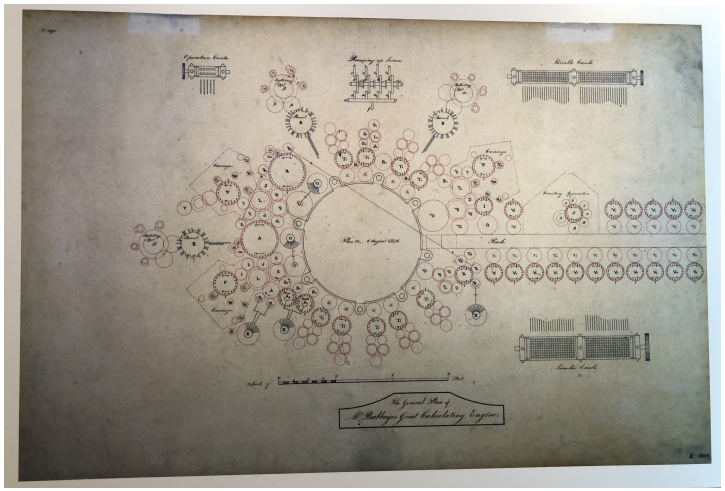
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."



By Ezrdr - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=21954723



By Canticle at English Wikipedia, CB BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10532728

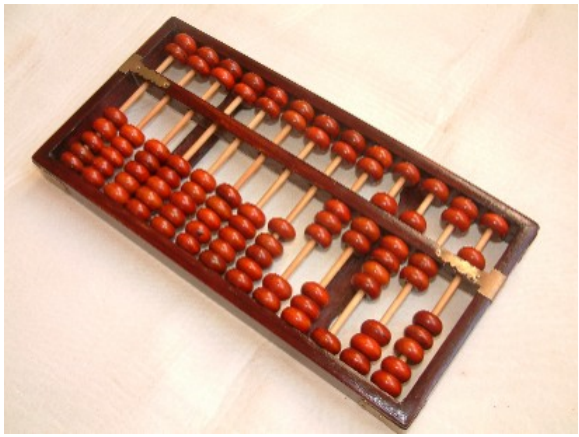


Charles Babbage: The Analytical Engine

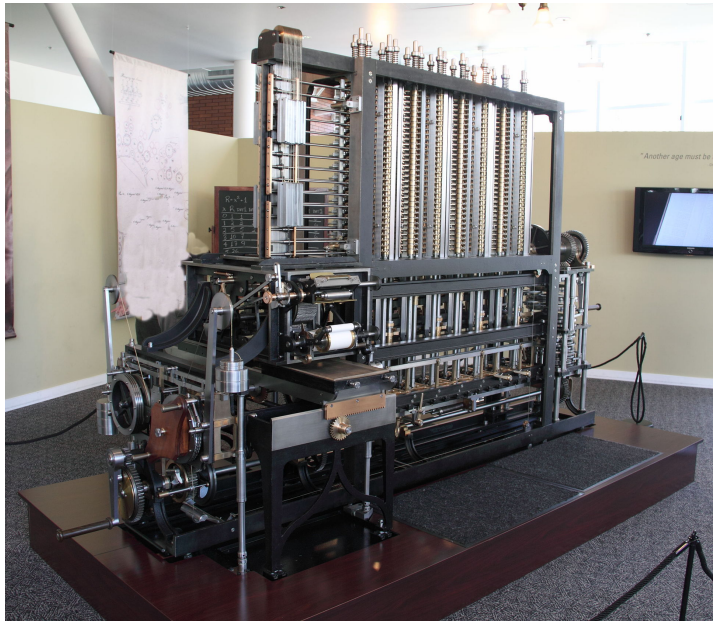
$$\begin{array}{r} {\scriptstyle 1\ \ 1\ \ 1\ \ \ \ \ } \\ 1\ 9\ 7\ 3\ 4 \\ +5\ 4\ 2\ 8\ 3 \\ \hline 4\ 0\ 1\ 7 \end{array}$$

# Algoritmer før (moderne) computere

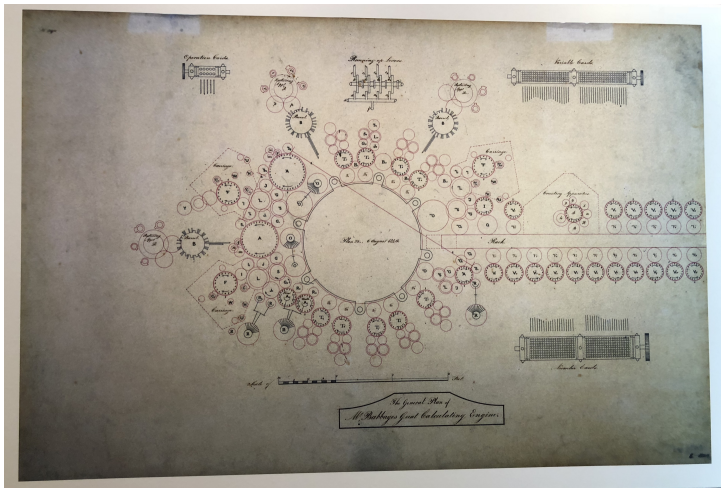
"Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing."



By Ezrdr - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=21954723



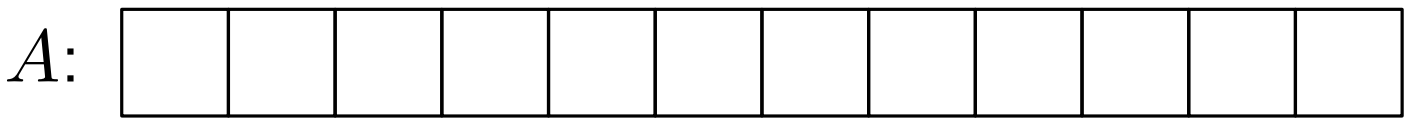
By Canticle at English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10532728



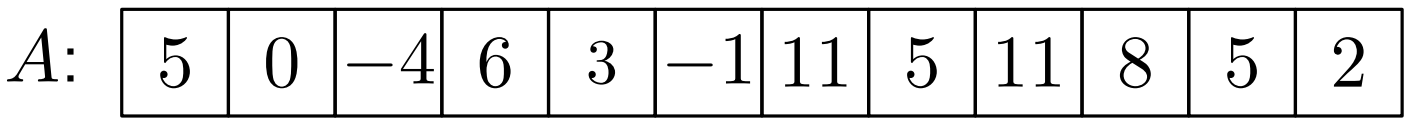
Charles Babbage: The Analytical Engine

```
  1  1  1
  1  9  7  3  4
+5  4  2  8  3
-----------------
  7  4  0  1  7
=================
```

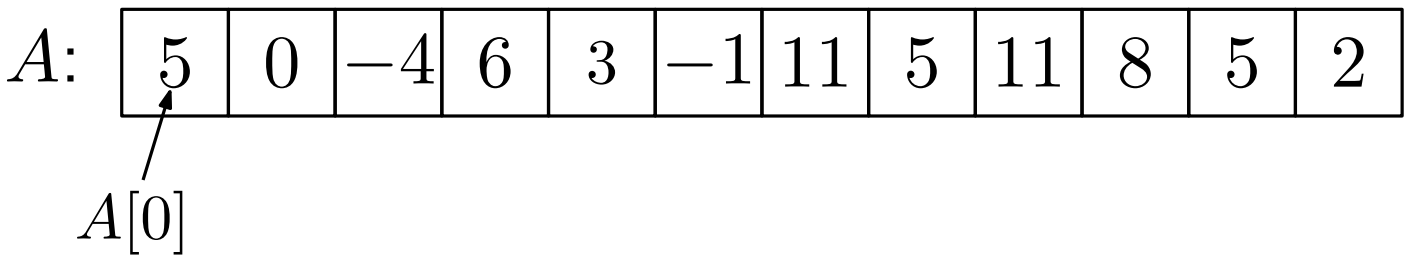
# Arrays (tabeller)



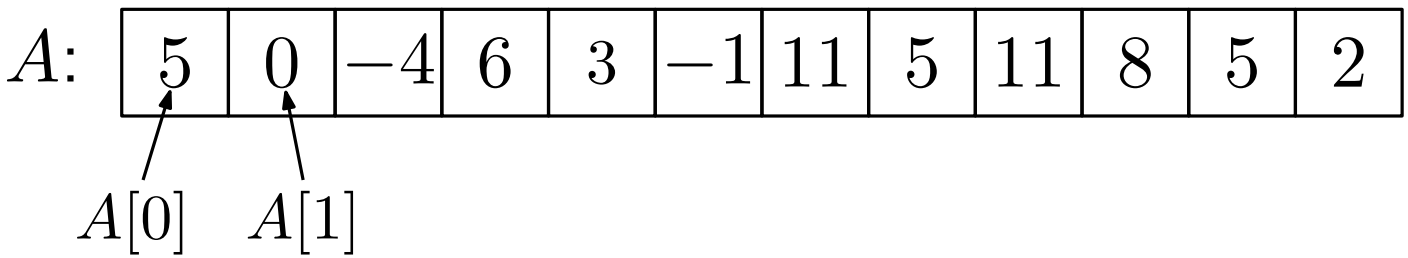
# Arrays (tabeller)



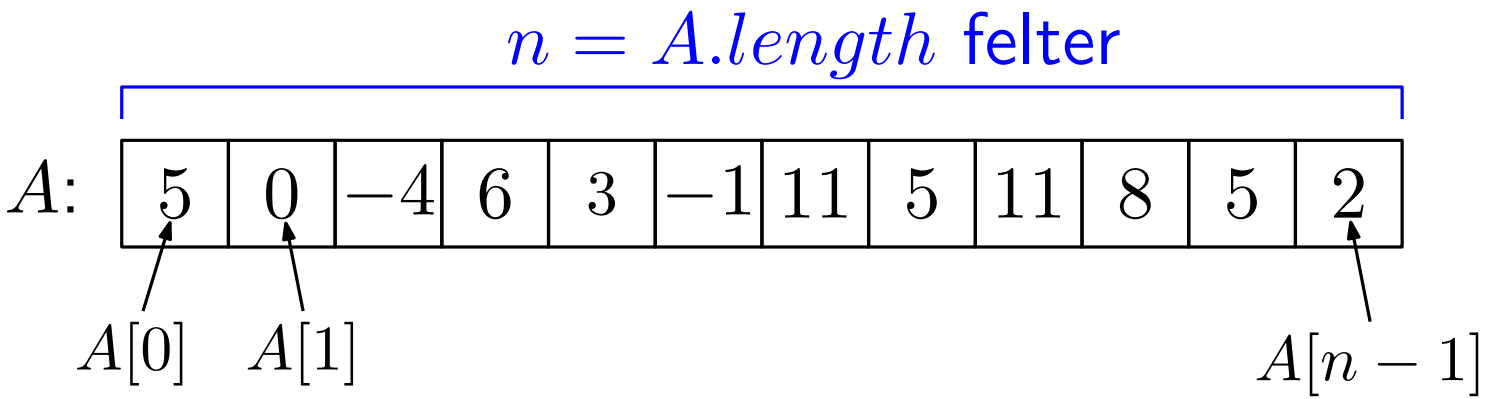
# Arrays (tabeller)



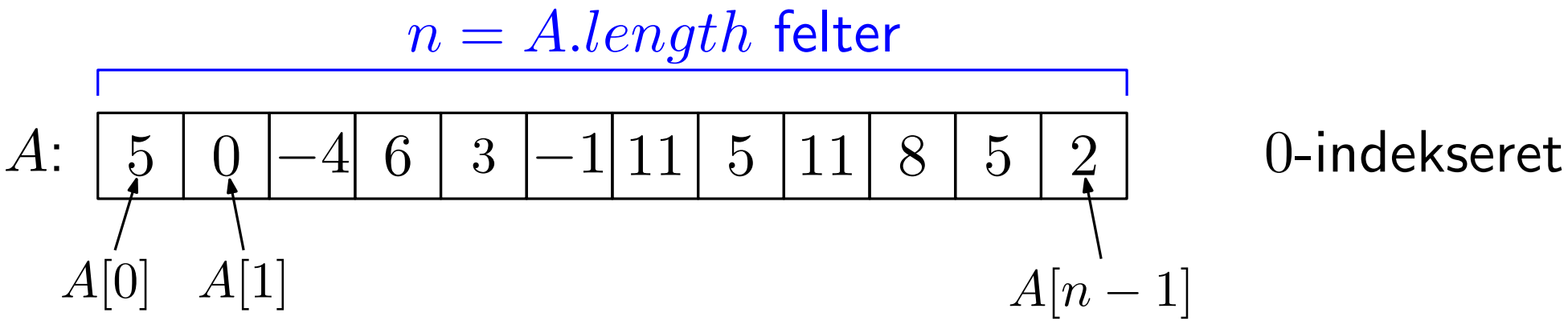
# Arrays (tabeller)



# Arrays (tabeller)

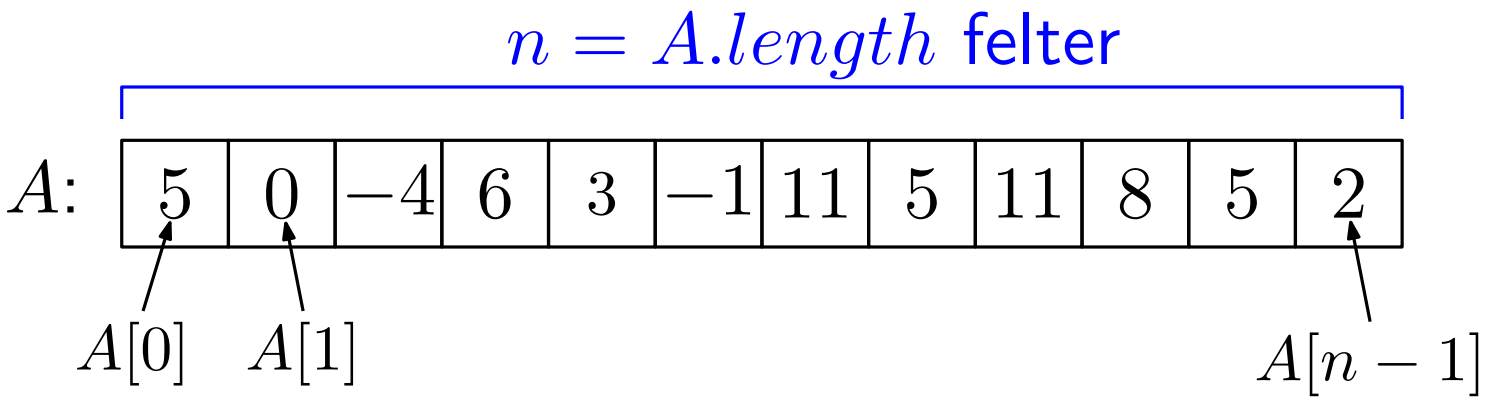


# Arrays (tabeller)

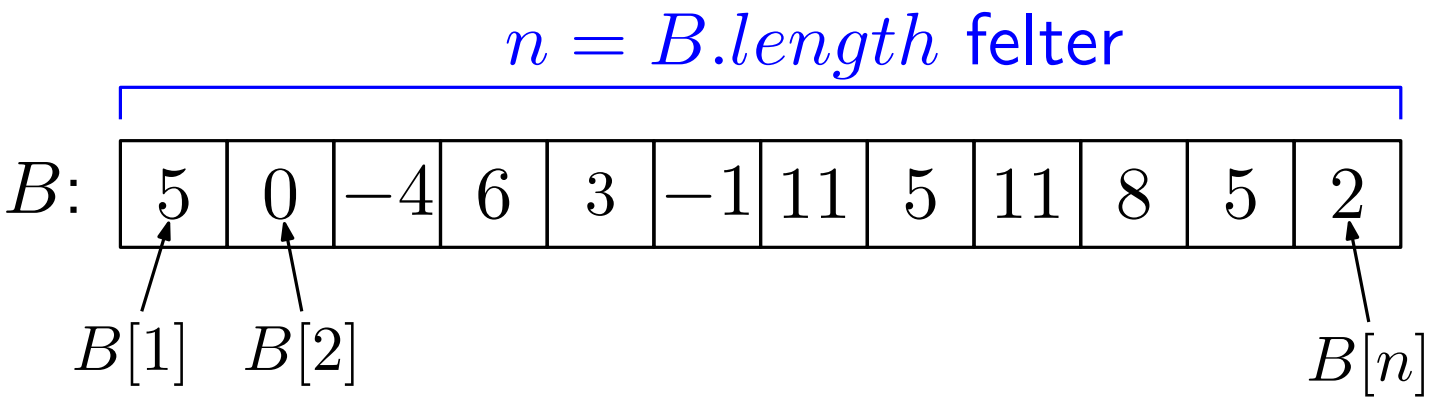



0-indekseret

# Arrays (tabeller)

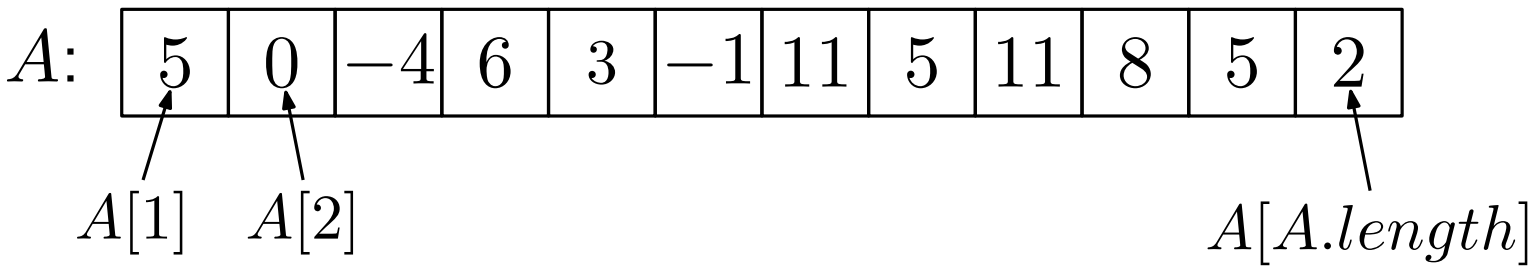



0-indekseret



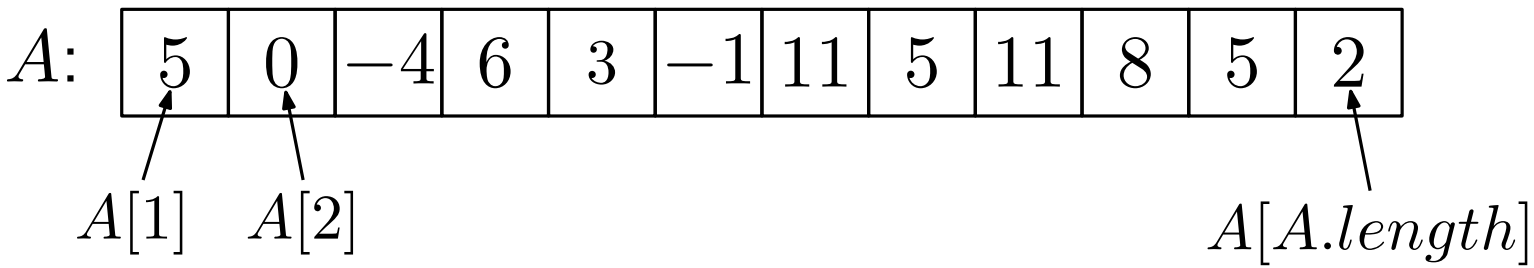

1-indekseret

# Maximum i array




```
findMax1(A)
    max = 1
    for i = 1 to A.length
        if A[i] > A[max]
            max = i
    return max
```

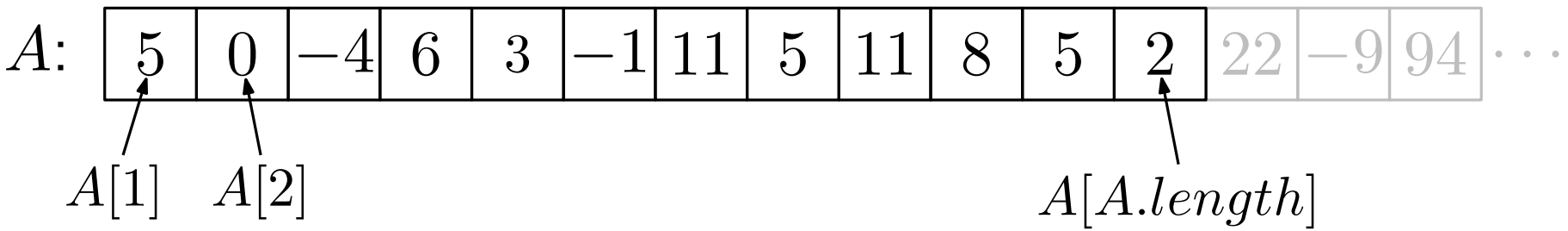
# Maximum i array




```
findMax1(A)
    max = 1
    for i = 1 to A.length
        if A[i] > A[max]
            max = i
    return max
```

```
findMax2(A, m)
    max = 1
    for i = 1 to m
        if A[i] > A[max]
            max = i
    return max
```

# Maximum i array




```
findMax1(A)
    max = 1
    for i = 1 to A.length
        if A[i] > A[max]
            max = i
    return max
```

```
findMax2(A, m)
    max = 1
    for i = 1 to m
        if A[i] > A[max]
            max = i
    return max
```

# Maksimum i array

**Input:** Array $A$ af størrelse $n$, dvs. $A[0 : n-1]$.
**Output:** Et index $i$, $0 \leq i < n$, så $A[i] = \max\{A[0], \ldots, A[n-1]\}$.

# Maksimum i array

**Input:** Array $A$ af størrelse $n$, dvs. $A[0 : n-1]$.
**Output:** Et index $i$, $0 \leq i < n$, så $A[i] = \max\{A[0], \ldots, A[n-1]\}$.

**Algoritme:**

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

# Maksimum i array

**Input:** Array $A$ af størrelse $n$, dvs. $A[0 : n-1]$.
**Output:** Et index $i$, $0 \leq i < n$, så $A[i] = \max\{A[0], \ldots, A[n-1]\}$.

**Algoritme:**

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

Gennemløb $A$ og vedligehold index af hidtil største indgang. Returnér index.

# Maksimum i array

**Input:** Array $A$ af størrelse $n$, dvs. $A[0 : n-1]$.
**Output:** Et index $i$, $0 \leq i < n$, så $A[i] = \max\{A[0], \ldots, A[n-1]\}$.

**Algoritme:**

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

Gennemløb $A$ og vedligehold index af hidtil største indgang. Returnér index.

```
let findMax (A: int[]) (n: int) =
    let mutable maxIdx = 0
    for i in 0 .. (n - 1) do
        if A.[i] > A.[maxIdx] then
            maxIdx <- i
    maxIdx
```

# Maksimum i array

**Input:** Array $A$ af størrelse $n$, dvs. $A[0 : n-1]$.
**Output:** Et index $i$, $0 \leq i < n$, så $A[i] = \max\{A[0], \ldots, A[n-1]\}$.

**Algoritme:**

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

Ligesom rigtig kode, men uden besværet.
Skrevet til at blive læst af *mennesker*, ikke en computer.
Meget mere præcis end rent sproglig beskrivelse.

Gennemløb $A$ og vedligehold index af hidtil største indgang. Returnér index.

```
let findMax (A: int[]) (n: int) =
    let mutable maxIdx = 0
    for i in 0 .. (n - 1) do
        if A.[i] > A.[maxIdx] then
            maxIdx <- i
    maxIdx
```

# Tidskompleksitet: Hvilken maskine?

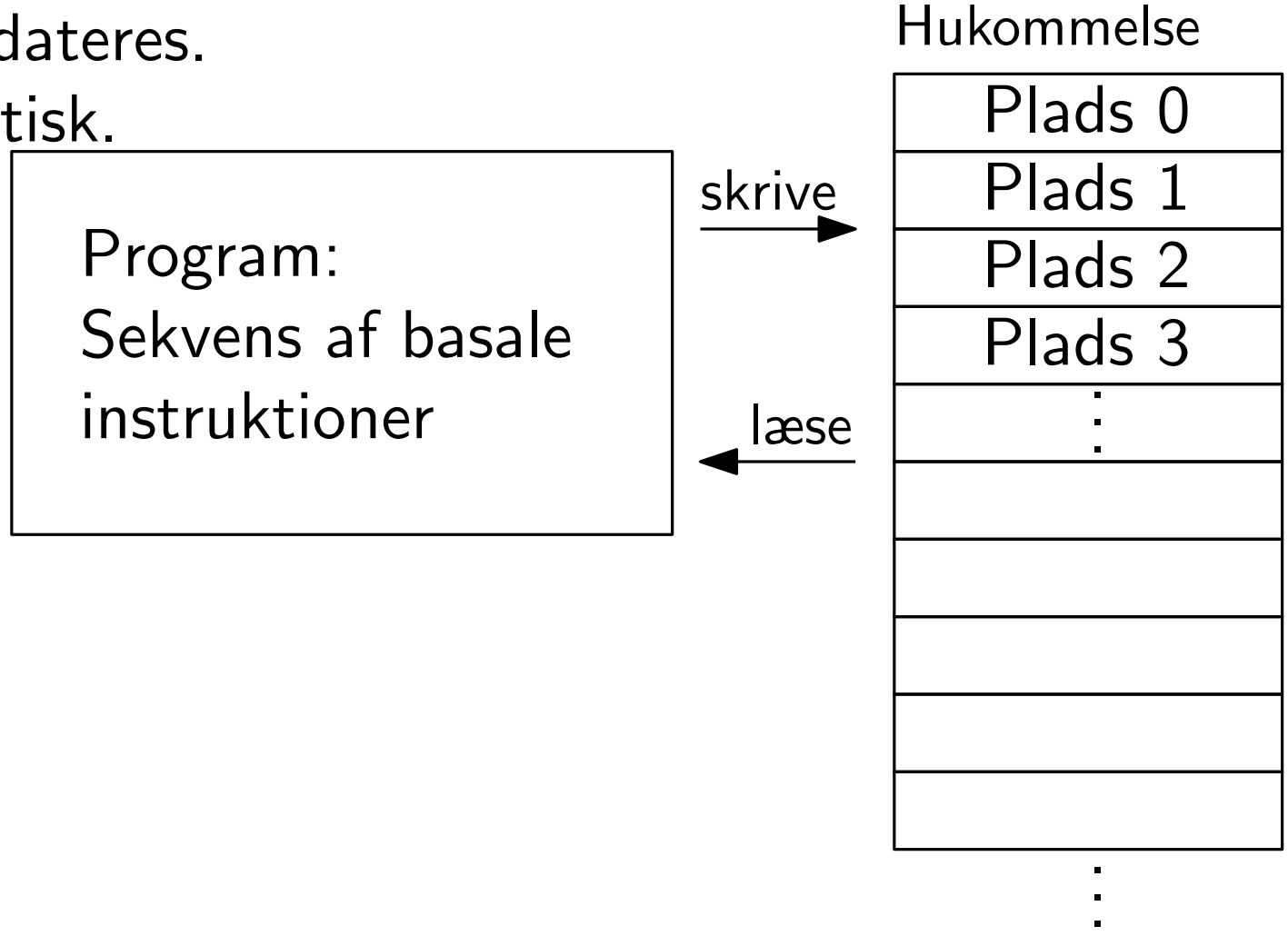

- Forskellige instruktioner.
- Forskellige hastigheder.
- Der kommer en ny model i morgen.

# Løsning: RAM-modellen!

- Abstrakt matematisk model af en computer.
- Simplere end alle virkelige computere.
- Skal aldrig opdateres.
- Alligevel realistisk.

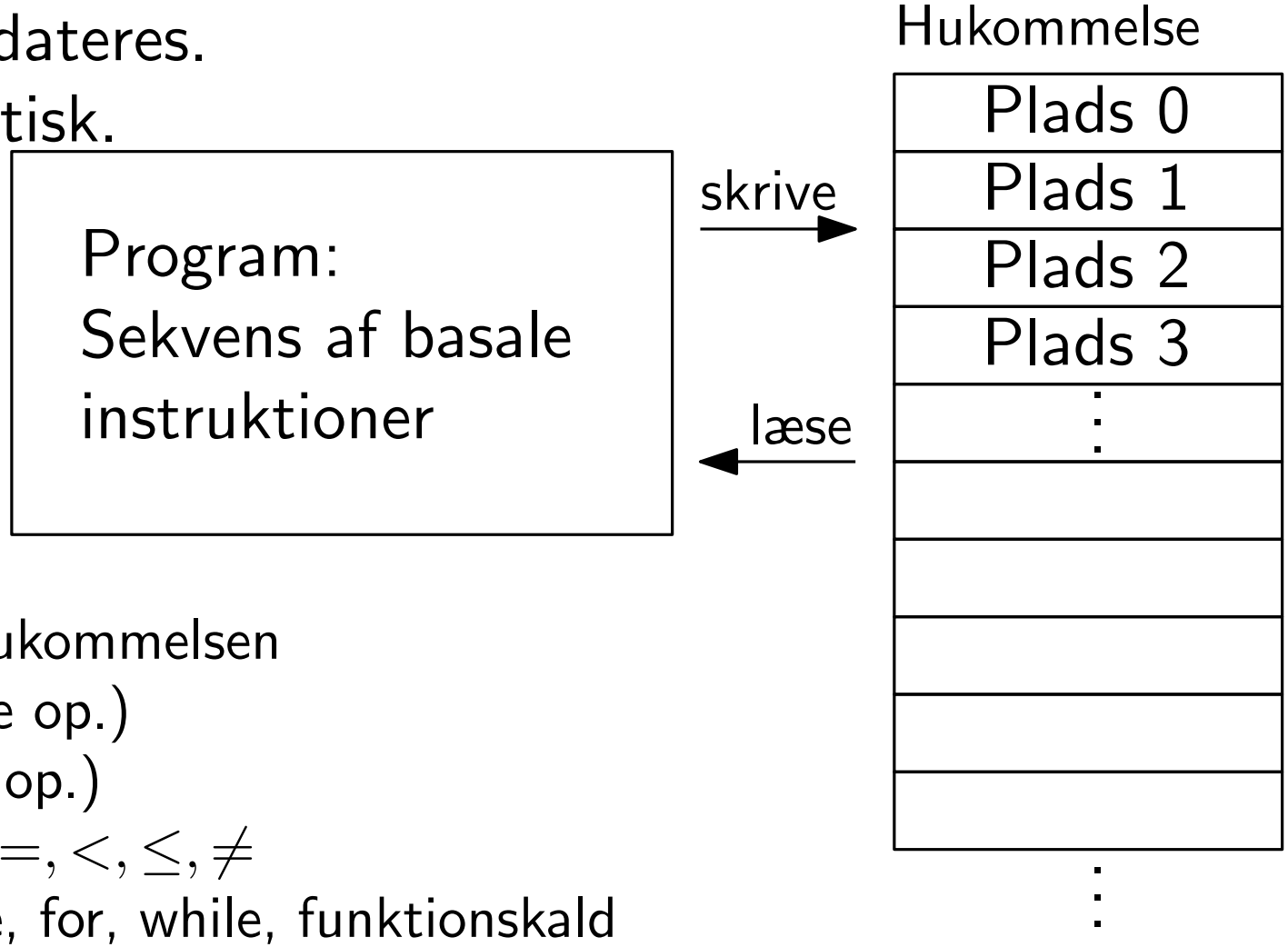
# Løsning: RAM-modellen!

- Abstrakt matematisk model af en computer.
- Simplere end alle virkelige computere.
- Skal aldrig opdateres.
- Alligevel realistisk.



# Løsning: RAM-modellen!

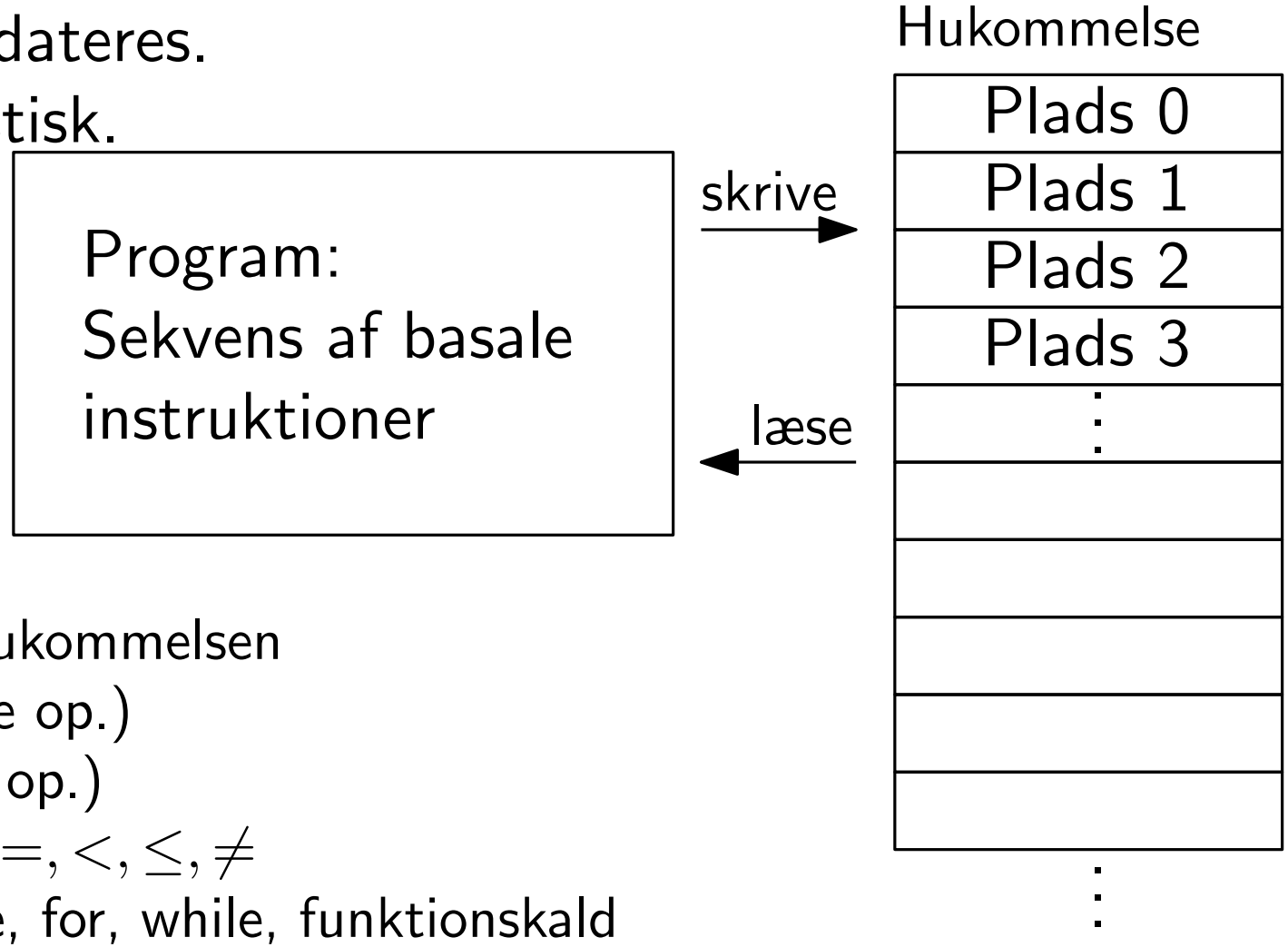
- Abstrakt matematisk model af en computer.
- Simplere end alle virkelige computere.
- Skal aldrig opdateres.
- Alligevel realistisk.




Basale instruktioner:
Læse fra/skrive til hukommelsen
$+, -, \cdot, /$ (aritmetiske op.)
and, or, not (logiske op.)
Sammenligninger: $==, <, \leq, \neq$
Program flow: if-else, for, while, funktionskald

# Løsning: RAM-modellen!

- Abstrakt matematisk model af en computer.
- Simplere end alle virkelige computere.
- Skal aldrig opdateres.
- Alligevel realistisk.



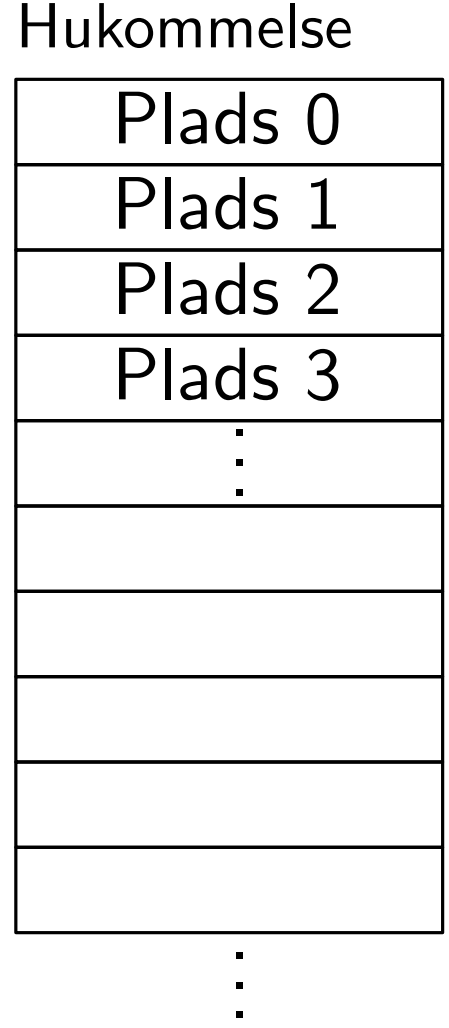

Basale instruktioner:
Læse fra/skrive til hukommelsen
$+, -, \cdot, /$ (aritmetiske op.)
and, or, not (logiske op.)
Sammenligninger: $==, <, \leq, \neq$
Program flow: if-else, for, while, funktionskald
Hver instruktion tager konstant tid.
Køretid = #instruktioner

# Ord/word

Hver plads kan indeholde et ord/word, fx:

- $w$ bits, konstant $w$.
- Et heltal mindre end en (stor) konstant $N$.
- Et tegn (bogstav/symbol).

Ikke lange tekster, mange tal eller vilkårligt store tal!
Disse må gemmes som flere ord.



# Ord/word

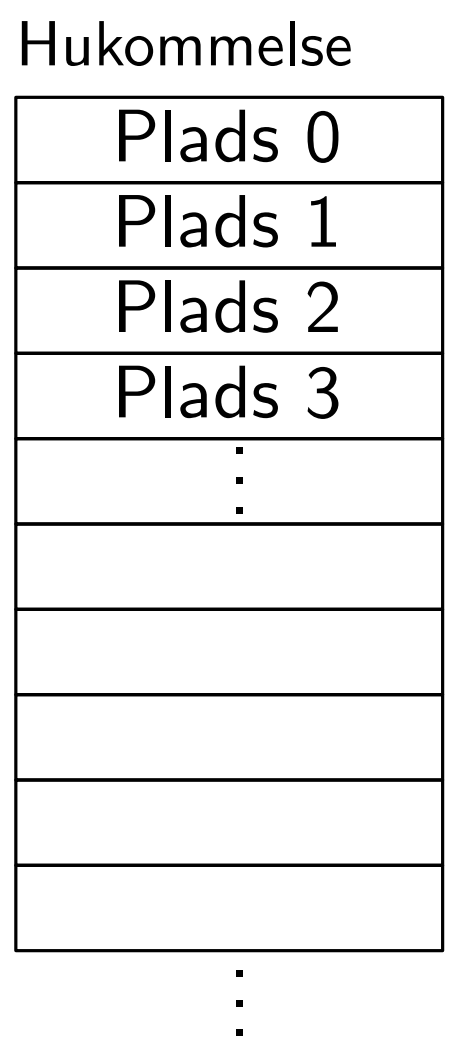
Hver plads kan indeholde et ord/word, fx:

- $w$ bits, konstant $w$.
- Et heltal mindre end en (stor) konstant $N$.
- Et tegn (bogstav/symbol).

Ikke lange tekster, mange tal eller vilkårligt store tal!
Disse må gemmes som flere ord.

Variable er læsevenlige navne for pladser i hukommelsen:

Pseudokode:



# Ord/word

Hver plads kan indeholde et ord/word, fx:

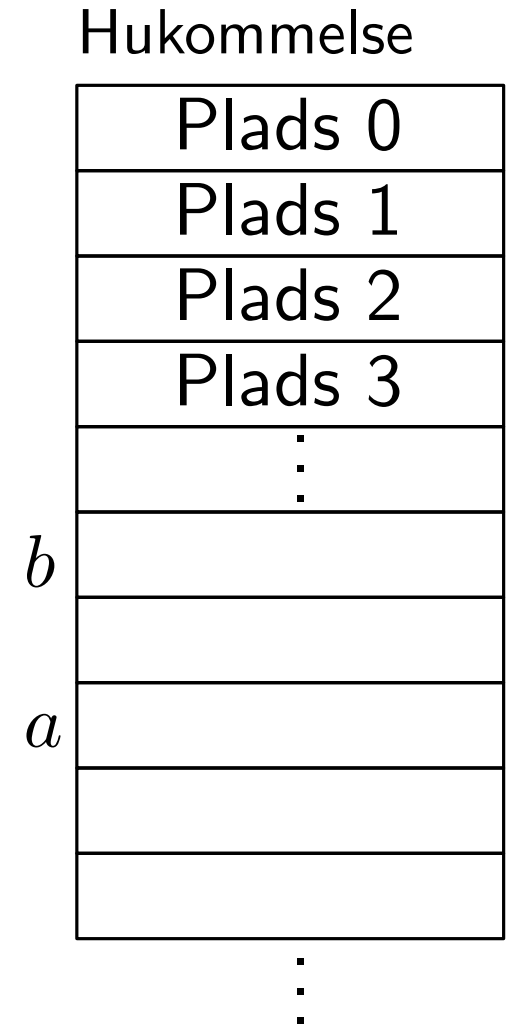
- $w$ bits, konstant $w$.
- Et heltal mindre end en (stor) konstant $N$.
- Et tegn (bogstav/symbol).

Ikke lange tekster, mange tal eller vilkårligt store tal!
Disse må gemmes som flere ord.

Variable er læsevenlige navne for pladser i hukommelsen:

Pseudokode:

$a = 4$
$b = 1$
$b = b + 8$
$a = a + b$



# Ord/word

Hver plads kan indeholde et ord/word, fx:

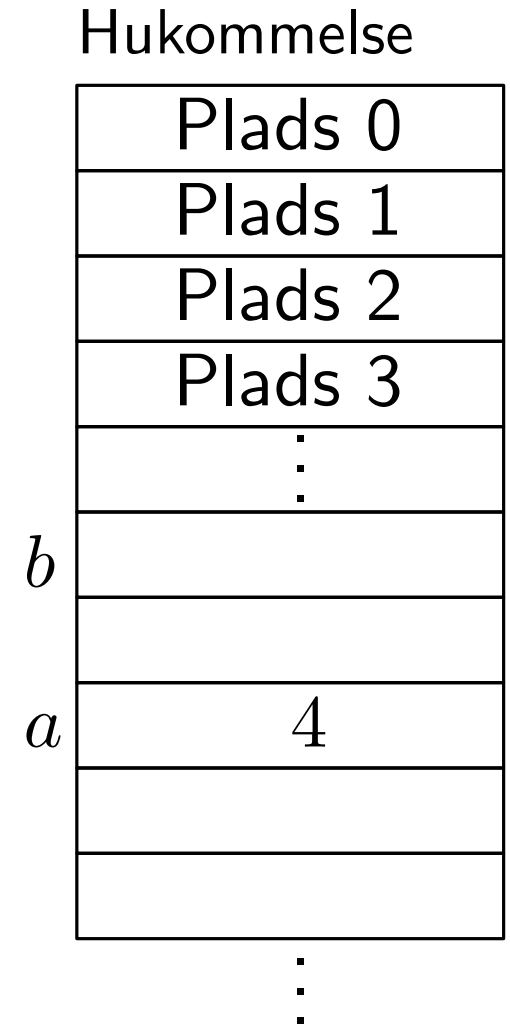
- $w$ bits, konstant $w$.
- Et heltal mindre end en (stor) konstant $N$.
- Et tegn (bogstav/symbol).

Ikke lange tekster, mange tal eller vilkårligt store tal!
Disse må gemmes som flere ord.

Variable er læsevenlige navne for pladser i hukommelsen:

Pseudokode:

$a = 4$
$b = 1$
$b = b + 8$
$a = a + b$



# Ord/word

Hver plads kan indeholde et ord/word, fx:

- $w$ bits, konstant $w$.
- Et heltal mindre end en (stor) konstant $N$.
- Et tegn (bogstav/symbol).

Ikke lange tekster, mange tal eller vilkårligt store tal!
Disse må gemmes som flere ord.

Variable er læsevenlige navne for pladser i hukommelsen:

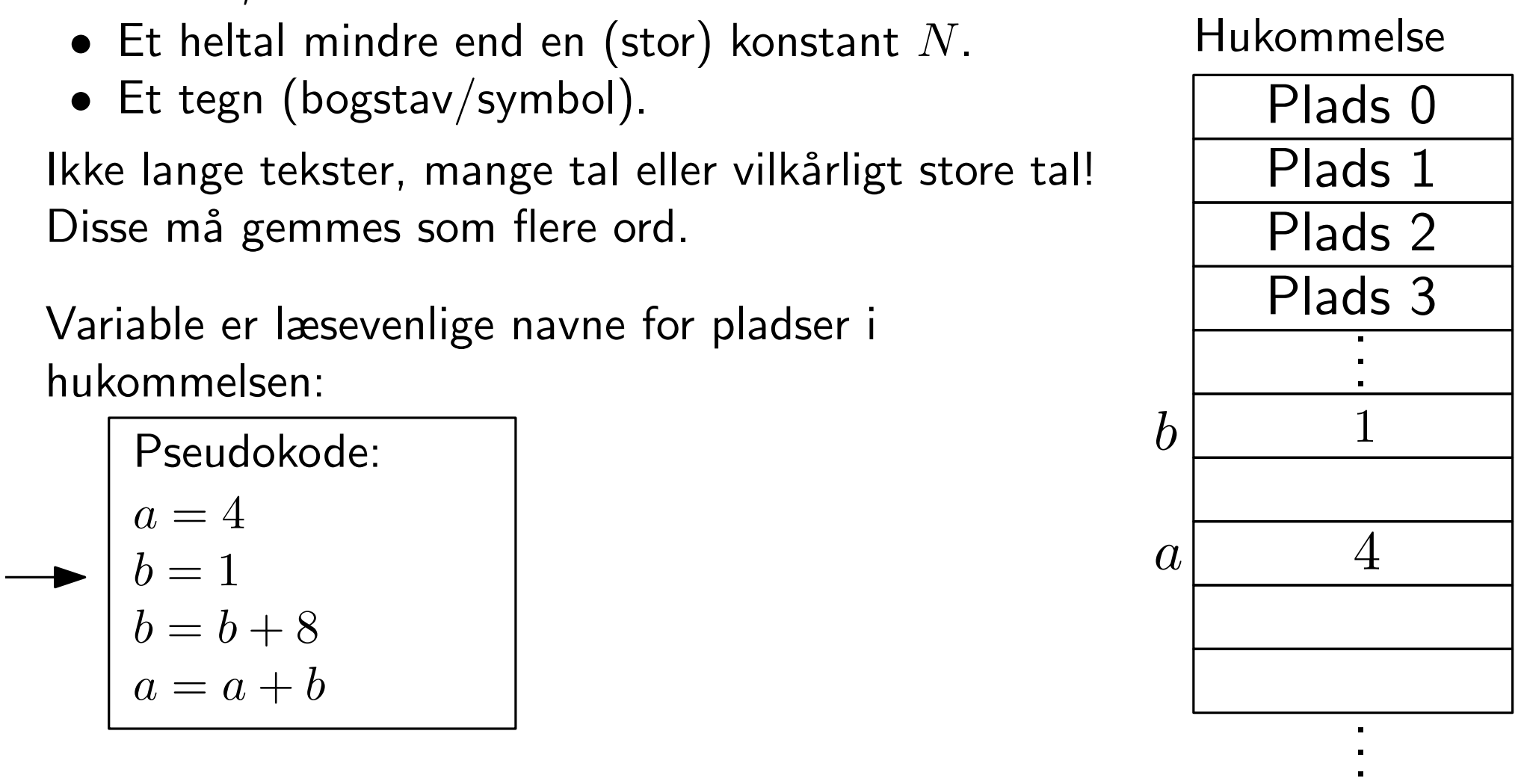
Pseudokode:

$a = 4$

$b = 1$

$b = b + 8$

$a = a + b$



# Ord/word

Hver plads kan indeholde et ord/word, fx:

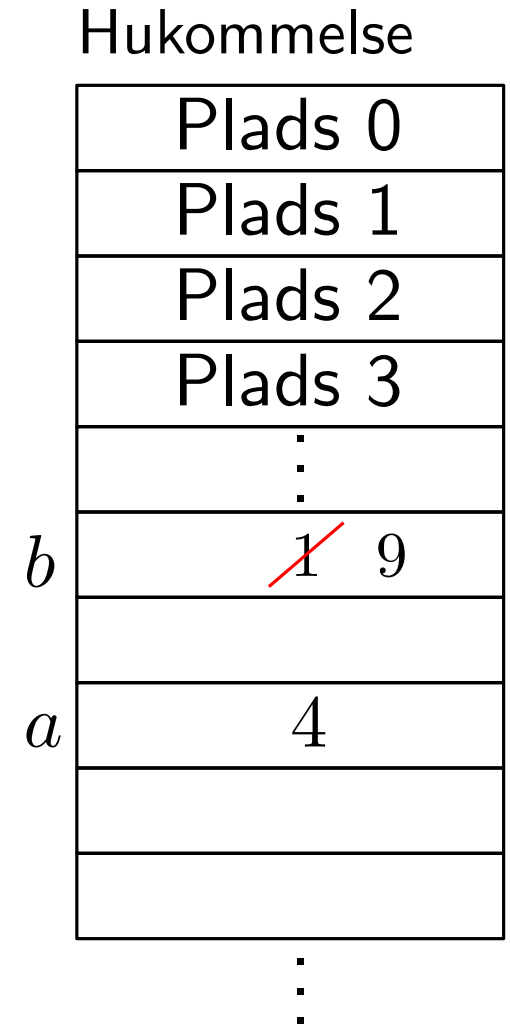
- $w$ bits, konstant $w$.
- Et heltal mindre end en (stor) konstant $N$.
- Et tegn (bogstav/symbol).

Ikke lange tekster, mange tal eller vilkårligt store tal!
Disse må gemmes som flere ord.

Variable er læsevenlige navne for pladser i hukommelsen:

Pseudokode:

$a = 4$
$b = 1$
→ $b = b + 8$
$a = a + b$



# Ord/word

Hver plads kan indeholde et ord/word, fx:

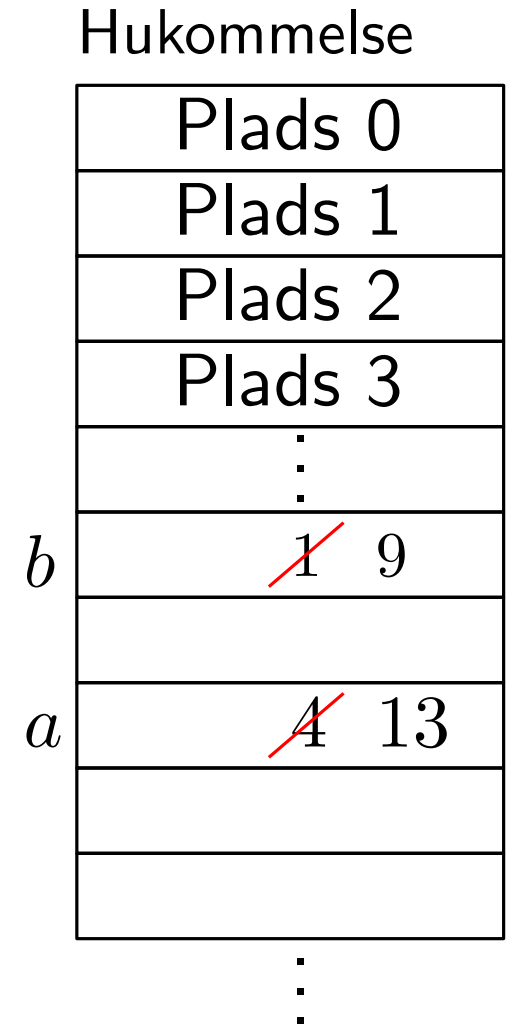
- $w$ bits, konstant $w$.
- Et heltal mindre end en (stor) konstant $N$.
- Et tegn (bogstav/symbol).

Ikke lange tekster, mange tal eller vilkårligt store tal!
Disse må gemmes som flere ord.

Variable er læsevenlige navne for pladser i hukommelsen:

Pseudokode:

$a = 4$
$b = 1$
$b = b + 8$
→ $a = a + b$

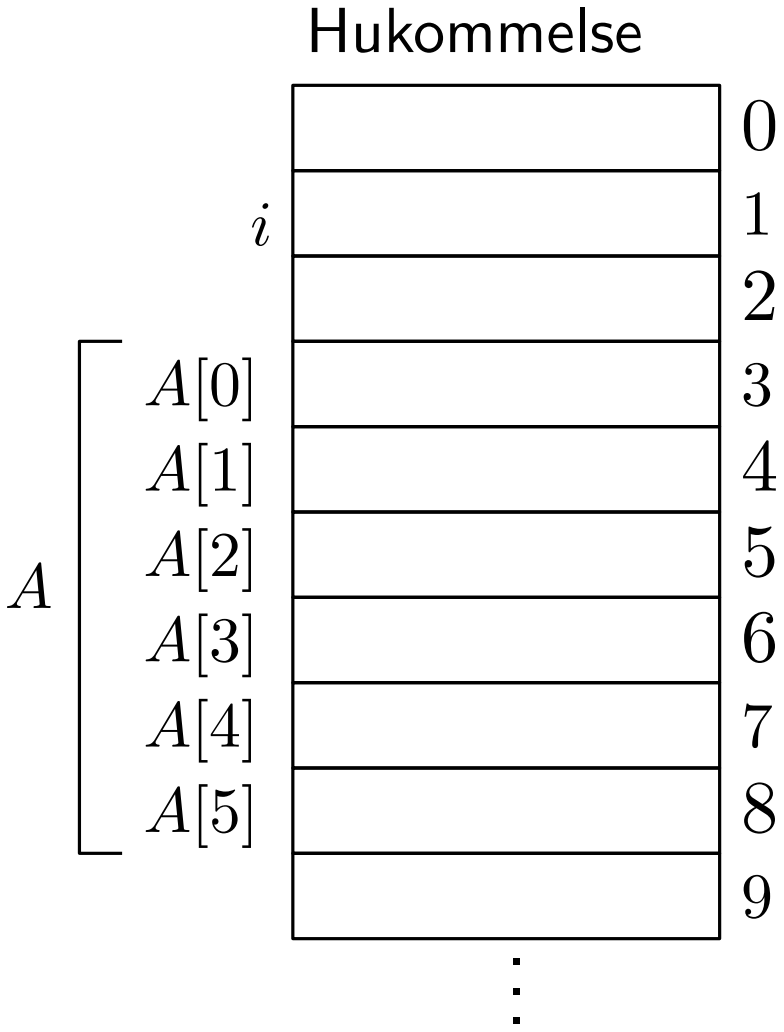


# Arrays/tabeller

Array: Sammenhængende del af hukommelsen.

Pseudokode:

$A[0] = 5$
$A[4] = 6$
$i = 2$
$A[i] = 8$

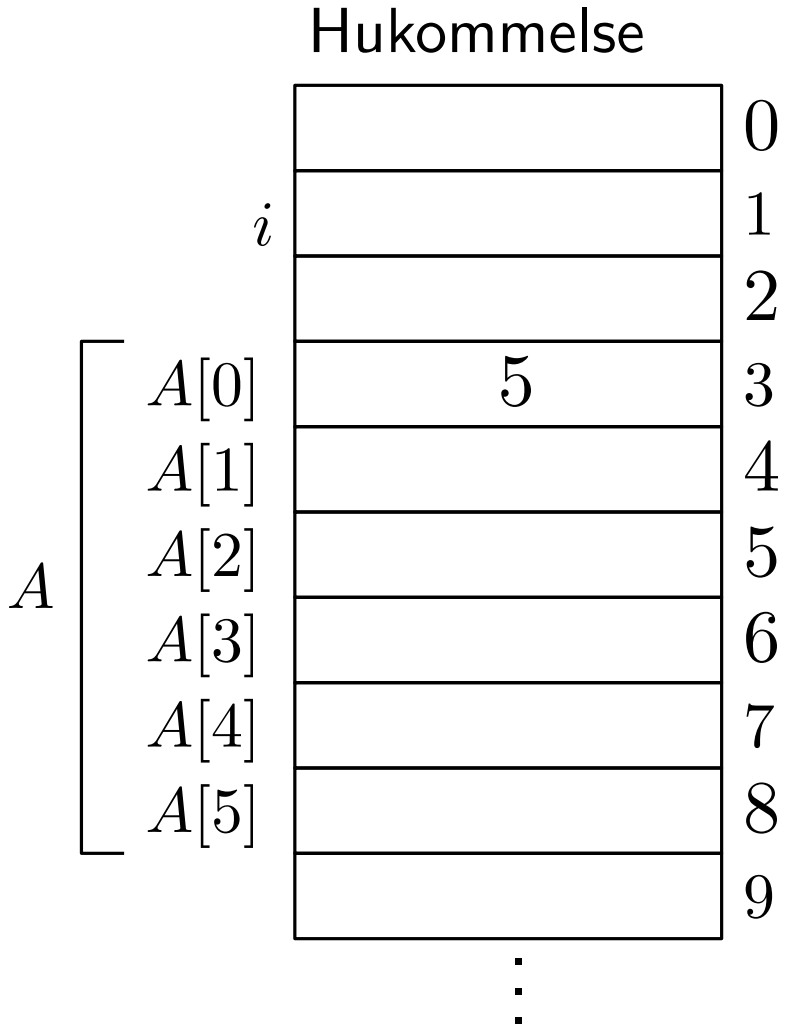


# Arrays/tabeller

Array: Sammenhængende del af hukommelsen.

Pseudokode:

$A[0] = 5$
$A[4] = 6$
$i = 2$
$A[i] = 8$



# Arrays/tabeller

Array: Sammenhængende del af hukommelsen.

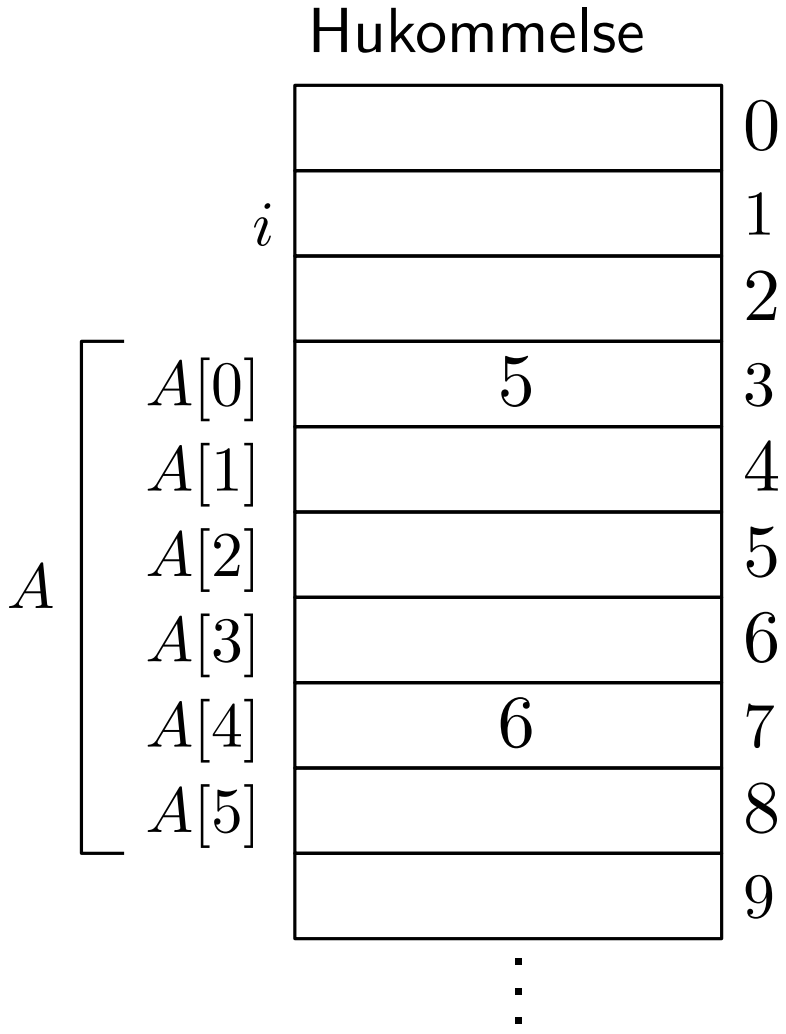
Pseudokode:

$A[0] = 5$

$A[4] = 6$

$i = 2$

$A[i] = 8$



# Arrays/tabeller

Array: Sammenhængende del af hukommelsen.

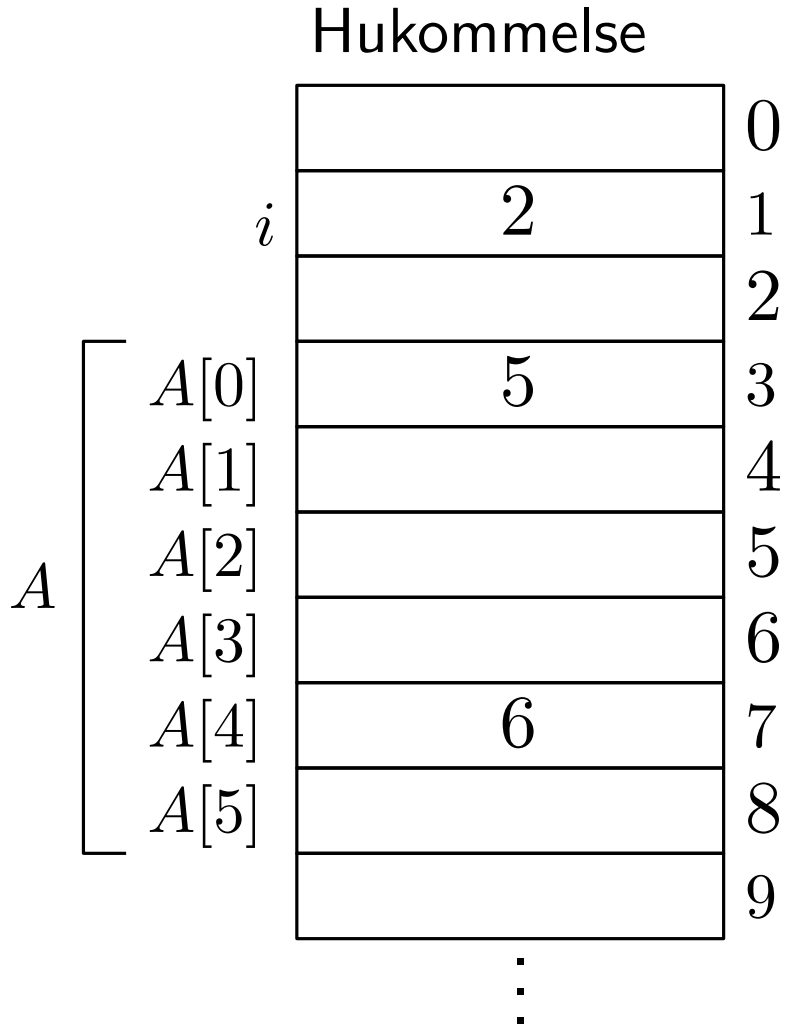
Pseudokode:

$A[0] = 5$

$A[4] = 6$

$i = 2$

$A[i] = 8$

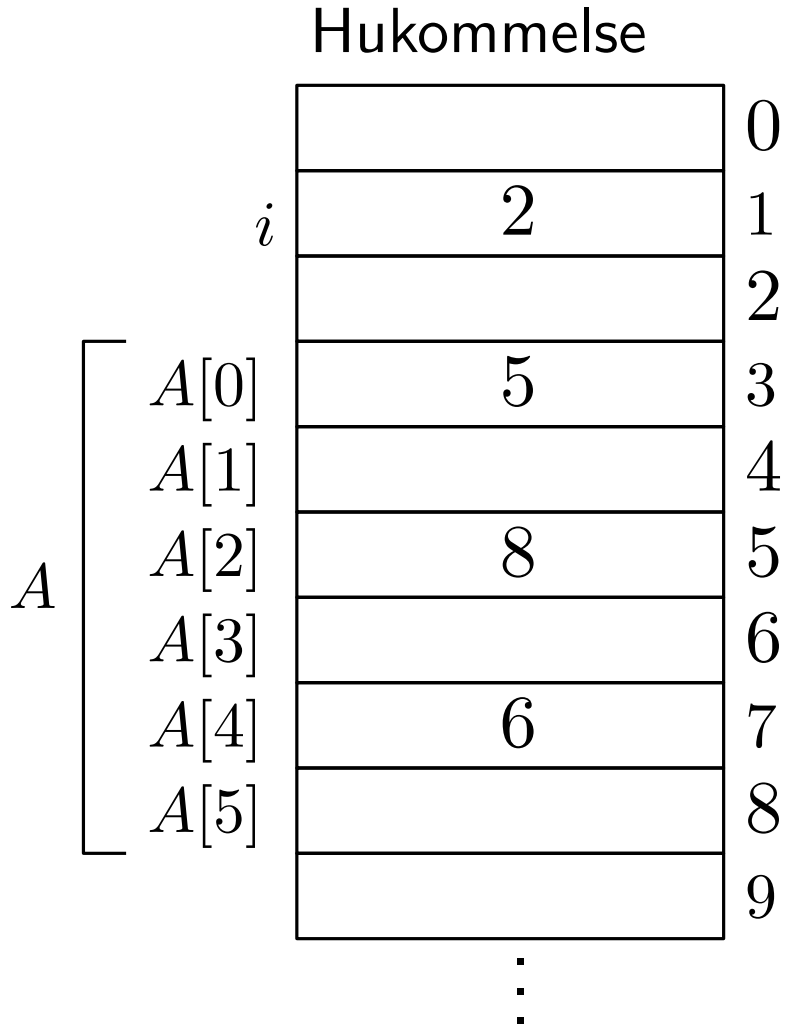


# Arrays/tabeller

Array: Sammenhængende del af hukommelsen.

Pseudokode:

$A[0] = 5$
$A[4] = 6$
$i = 2$
$A[i] = 8$



# Køretid/tidskompleksitet

$T(n) = \#$ instruktioner som algoritmen laver på et input af størrelse $n$

# Køretid/tidskompleksitet

$T(n) = \#$ instruktioner som algoritmen laver på et input af størrelse $n$

Instruktioner:

- Læse fra/skrive til hukommelse ($x = y$, $A[i]$, $i = i + 1$,...)
- Aritmetiske/logiske operationer ($+, -, \cdot, /$, and, or, not)
- Sammenligninger ($<, >, \leq, \geq, ==, \neq$)
- Programflow (if-else, while, for, funktionskald)

# Køretid/tidskompleksitet

$T(n) = \#$ instruktioner som algoritmen laver på et input af størrelse $n$

Instruktioner:

- Læse fra/skrive til hukommelse ($x = y$, $A[i]$, $i = i + 1$,...)
- Aritmetiske/logiske operationer ($+, -, \cdot, /$, and, or, not)
- Sammenligninger ($<, >, \leq, \geq, ==, \neq$)
- Programflow (if-else, while, for, funktionskald)

Tidskompleksitet i værste fald (worst-case complexity):
Maximal køretid for alle input af størrelse $n$.

# Eksempel

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

# Eksempel

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

| skridt i linje | |
|---|---|
| $c_1$ | små konstanter |
| $c_2$ | |
| $c_3$ | |
| $c_4$ | |
| $c_5$ | |

# Eksempel

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

| skridt i linje | |
|---|---|
| $c_1$ | |
| $c_2$ | små konstanter |
| $c_3$ | |
| $c_4$ | |
| $c_5$ | |

skridt hver gang:

for

læse $i$

udregnet $i + 1$

gemme $i + 1$ som $i$

læse $n$

udregne $n - 1$

sammenligne ny $i$ med $n - 1$

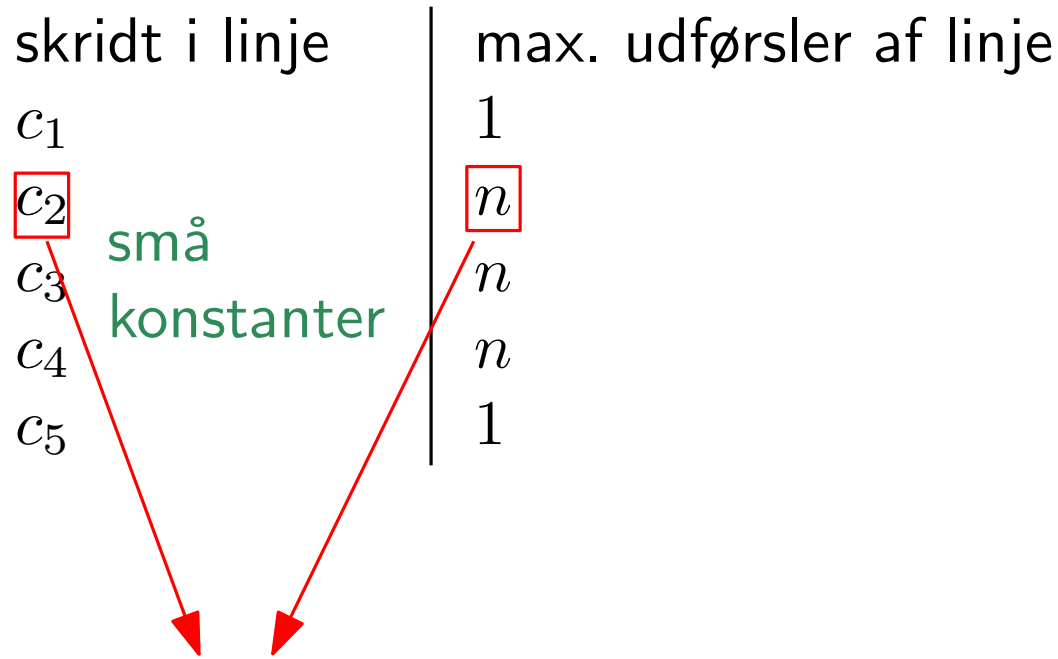
# Eksempel

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

| skridt i linje | | max. udførsler af linje |
|---|---|---|
| $c_1$ | | $1$ |
| $c_2$ | små konstanter | $n$ |
| $c_3$ | | $n$ |
| $c_4$ | | $n$ |
| $c_5$ | | $1$ |

# Eksempel

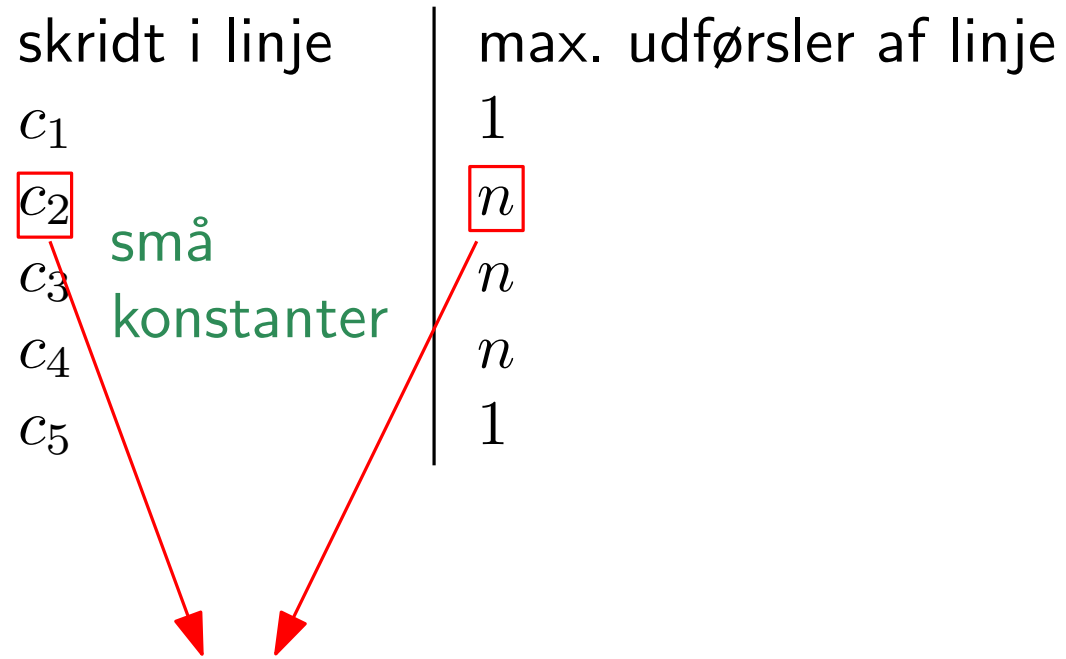
```
findMax(A, n)
    max = 0
    for i = 0 to n − 1
        if A[i] > A[max]
            max = i
    return max
```




$T(n) =$ køretid $=$ skridt i alt $= c_1 + c_2 n + c_3 n + c_4 n + c_5 =$
$(c_2 + c_3 + c_4)n + c_1 + c_5 = \Theta(n)$

# Eksempel

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




$T(n) =$ køretid $=$ skridt i alt $= c_1 + c_2 n + c_3 n + c_4 n + c_5 =$
$(c_2 + c_3 + c_4)n + c_1 + c_5 = \Theta(n)$

Asymptotisk notation: Køretiden er lineær. Langsomt voksende led og konstanter ignoreres.

# Eksempel

```
doubleLoop(n)
    x = 0
    for i = 0 to n − 1
        x = x + 1
        for j = 0 to n − 1
            x = x + 1
    return x
```

# Eksempel

```
doubleLoop(n)
    x = 0
    for i = 0 to n − 1
        x = x + 1
        for j = 0 to n − 1
            x = x + 1
    return x
```

| # skridt i linje |
|---|
| $c_1$ |
| $c_2$ |
| $c_3$ |
| $c_4$ |
| $c_5$ |
| $c_6$ |

# Eksempel

```
doubleLoop(n)
    x = 0
    for i = 0 to n − 1
        x = x + 1
        for j = 0 to n − 1
            x = x + 1
    return x
```

| # skridt i linje | max. # udførsler af linje |
|---|---|
| $c_1$ | $1$ |
| $c_2$ | $n$ |
| $c_3$ | $n$ |
| $c_4$ | $n^2$ |
| $c_5$ | $n^2$ |
| $c_6$ | $1$ |

# Eksempel

| doubleLoop($n$) | # skridt i linje | max. # udførsler af linje |
|---|---|---|
| $x = 0$ | $c_1$ | $1$ |
| for $i = 0$ to $n - 1$ | $c_2$ | $n$ |
| $\quad x = x + 1$ | $c_3$ | $n$ |
| $\quad$ for $j = 0$ to $n - 1$ | $c_4$ | $n^2$ |
| $\quad\quad x = x + 1$ | $c_5$ | $n^2$ |
| return $x$ | $c_6$ | $1$ |

$$T(n) = c_1 + c_2n + c_3n + c_4n^2 + c_5n^2 + c_6 =$$
$$(c_4 + c_5)n^2 + (c_2 + c_3)n + c_1 + c_6 = \Theta(n^2)$$

# Eksempel

| doubleLoop($n$) | # skridt i linje | max. # udførsler af linje |
|---|---|---|
| $x = 0$ | $c_1$ | $1$ |
| for $i = 0$ to $n - 1$ | $c_2$ | $n$ |
| $\quad x = x + 1$ | $c_3$ | $n$ |
| $\quad$ for $j = 0$ to $n - 1$ | $c_4$ | $n^2$ |
| $\quad\quad x = x + 1$ | $c_5$ | $n^2$ |
| return $x$ | $c_6$ | $1$ |

$$T(n) = c_1 + c_2n + c_3n + c_4n^2 + c_5n^2 + c_6 =$$
$$(c_4 + c_5)n^2 + (c_2 + c_3)n + c_1 + c_6 = \boxed{\Theta(n^2)}$$

Asymptotisk notation: Køretiden er kvadratisk. Langsomt voksende led og konstanter ignoreres.

# Eksempel

| doubleLoop($n$) | # skridt i linje | max. # udførsler af linje |
|---|---|---|
| $x = 0$ | $c_1$ | $1$ |
| for $i = 0$ to $n - 1$ | $c_2$ | $n$ |
| $\quad x = x + 1$ | $c_3$ | $n$ |
| $\quad$ for $j = 0$ to $n - 1$ | $c_4$ | $n^2$ |
| $\quad\quad x = x + 1$ | $c_5$ | $n^2$ |
| return $x$ | $c_6$ | $1$ |

$$T(n) = c_1 + c_2n + c_3n + c_4n^2 + c_5n^2 + c_6 =$$
$$(c_4 + c_5)n^2 + (c_2 + c_3)n + c_1 + c_6 = \boxed{\Theta(n^2)}$$

Asymptotisk notation: Køretiden er kvadratisk. Langsomt voksende led og konstanter ignoreres.

```
twoLoops(n)
    x = 0
    for i = 0 to n - 1
        x = x + 1
    for j = 0 to n - 1
        x = x + 1
    return x
```

# Eksempel

| doubleLoop($n$) | # skridt i linje | max. # udførsler af linje |
|---|---|---|
| $x = 0$ | $c_1$ | 1 |
| for $i = 0$ to $n - 1$ | $c_2$ | $n$ |
| $x = x + 1$ | $c_3$ | $n$ |
| for $j = 0$ to $n - 1$ | $c_4$ | $n^2$ |
| $x = x + 1$ | $c_5$ | $n^2$ |
| return $x$ | $c_6$ | 1 |

$T(n) = c_1 + c_2n + c_3n + c_4n^2 + c_5n^2 + c_6 =$
$(c_4 + c_5)n^2 + (c_2 + c_3)n + c_1 + c_6 = \Theta(n^2)$

Asymptotisk notation: Køretiden er kvadratisk. Langsomt voksende led og konstanter ignoreres.

| twoLoops($n$) | # skridt i linje | max. # udførsler af linje |
|---|---|---|
| $x = 0$ | $c_1$ | 1 |
| for $i = 0$ to $n - 1$ | $c_2$ | $n$ |
| $x = x + 1$ | $c_3$ | $n$ |
| for $j = 0$ to $n - 1$ | $c_4$ | $n$ |
| $x = x + 1$ | $c_5$ | $n$ |
| return $x$ | $c_6$ | 1 |

$T(n) = c_1 + c_2n + c_3n + c_4n^2 + c_5n^2 + c_6 =$
$(c_2 + c_3 + c_4 + c_5)n + c_1 + c_6 = \Theta(n)$

# Quiz

```
Q(A, n) \\antag log_2 n er et heltal
    s = 0
    for i from 1 to n
        for j from 1 to log_2 n
            s = s + A[i] + A[j]
    return s
```

# Quiz

```
Q(A, n) \\antag log_2 n er et heltal
    s = 0
    for i from 1 to n
        for j from 1 to log_2 n
            s = s + A[i] + A[j]
    return s
```

Hvad er køretiden af Q$(A, n)$?

socrative.com $\rightarrow$ Student login.

Room name: ABRAHAMSEN3464

(a) $\Theta(n)$

(b) $\Theta(n^2)$

(c) $\Theta(n + \log_2 n)$

(d) $\Theta(n \cdot \log_2 n)$

(e) $\Theta(n\sqrt{n})$