DMA uge 2, fredag

Plan for i dag:

- Afrunding på Ω-, O- og Θ-notation (meget mere senere i kurset).
- Afrunding på rekursivt definerede køretider (binary search og merge sort).
- Løkkeinvarianter.
- Overblik over hvad vi har lært.

Θ, Ω og O

Vi skriver:

- $T(n) = \Omega(n \log n)$ hvis

  vi underdriver

  $$c_1 \cdot n \log n \leq T(n)$$

  for en konstant $c_1 > 0$.
- $T(n) = O(n \log n)$ hvis

  vi overdriver

  $$T(n) \leq c_2 \cdot n \log n$$

  for en konstant $c_2 > 0$.
- $T(n) = \Theta(n \log n)$ hvis

  vi er præcise

  $$c_1 \cdot n \log n \leq T(n) \leq c_2 \cdot n \log n$$

  for konstanter $c_1, c_2 > 0$, dvs. $T(n) = \Omega(n \log n)$ og $T(n) = O(n \log n)$.

Skal gælde for alle tilstrækkeligt store n. I praksis $n \geq 2$.

Θ, Ω og O

Vi skriver:

- $T(n) = \Omega(\overset{f(n)}{\cancel{n \log n}})$ hvis

  vi underdriver

  $$c_1 \cdot \overset{f(n)}{\cancel{n \log n}} \le T(n)$$

  for en konstant $c_1 > 0$.
- $T(n) = O(\underset{f(n)}{\cancel{n \log n}})$ hvis

  vi overdriver

  $$T(n) \le c_2 \cdot \overset{f(n)}{\cancel{n \log n}}$$

  for en konstant $c_2 > 0$.
- $T(n) = \Theta(\underset{f(n)}{\cancel{n \log n}})$ hvis

  vi er præcise

  $$c_1 \cdot \overset{f(n)}{\cancel{n \log n}} \le T(n) \le c_2 \cdot \overset{f(n)}{\cancel{n \log n}}$$

  for konstanter $c_1, c_2 > 0$, dvs. $T(n) = \Omega(\overset{f(n)}{\cancel{n \log n}})$ og $T(n) = O(\cancel{n \log n}).f(n)$

Eksempler vi har set:
$f(n) = \log n$
$f(n) = n$
$f(n) = n \log n$
$f(n) = n^2$

Skal gælde for alle tilstrækkeligt store n. I praksis $n \ge 2$.

Fjerne langsomme led og konstanter

Merge sort: $T(n) = c \cdot n \lg n + c \cdot n$. Vis $T(n) = \Theta(n \log n)$.

Fjerne langsomme led og konstanter

Merge sort: $T(n) = c \cdot n \lg n + c \cdot n$. Vis $T(n) = \Theta(n \log n)$.

Fjerne langsomme led giver *nedre grænse* (underdrivelse):

$$T(n) = c \cdot n \lg n + c \cdot n \geq c \cdot n \lg n,$$

så $T(n) = \Omega(n \boxed{\log} n)$.

Logaritme med uspecificeret base.
Tænk $\log = \lg = \log_2$.

Fjerne langsomme led og konstanter

Merge sort: $T(n) = c \cdot n \lg n + c \cdot n$. Vis $T(n) = \Theta(n \log n)$.

Fjerne langsomme led giver *nedre grænse* (underdrivelse):

$$T(n) = c \cdot n \lg n + c \cdot n \geq c \cdot n \lg n,$$

så $T(n) = \Omega(n \boxed{\log} n)$.

Logaritme med uspecificeret base.
Tænk $\log = \lg = \log_2$.

Runde langsomme led op giver *øvre grænse* (overdrivelse):

$$T(n) = c \cdot n \lg n + c \cdot n \leq c \cdot n \lg n + c \cdot n \lg n = 2c \cdot n \lg n,$$

så $T(n) = O(n \log n)$.

Fjerne langsomme led og konstanter

Merge sort: $T(n) = c \cdot n \lg n + c \cdot n$. Vis $T(n) = \Theta(n \log n)$.

Fjerne langsomme led giver *nedre grænse* (underdrivelse):

$$T(n) = c \cdot n \lg n + c \cdot n \geq c \cdot n \lg n,$$

så $T(n) = \Omega(n \boxed{\log} n)$.

Logaritme med uspecificeret base.
Tænk $\log = \lg = \log_2$.

Runde langsomme led op giver *øvre grænse* (overdrivelse):

$$T(n) = c \cdot n \lg n + c \cdot n \leq c \cdot n \lg n + c \cdot n \lg n = 2c \cdot n \lg n,$$

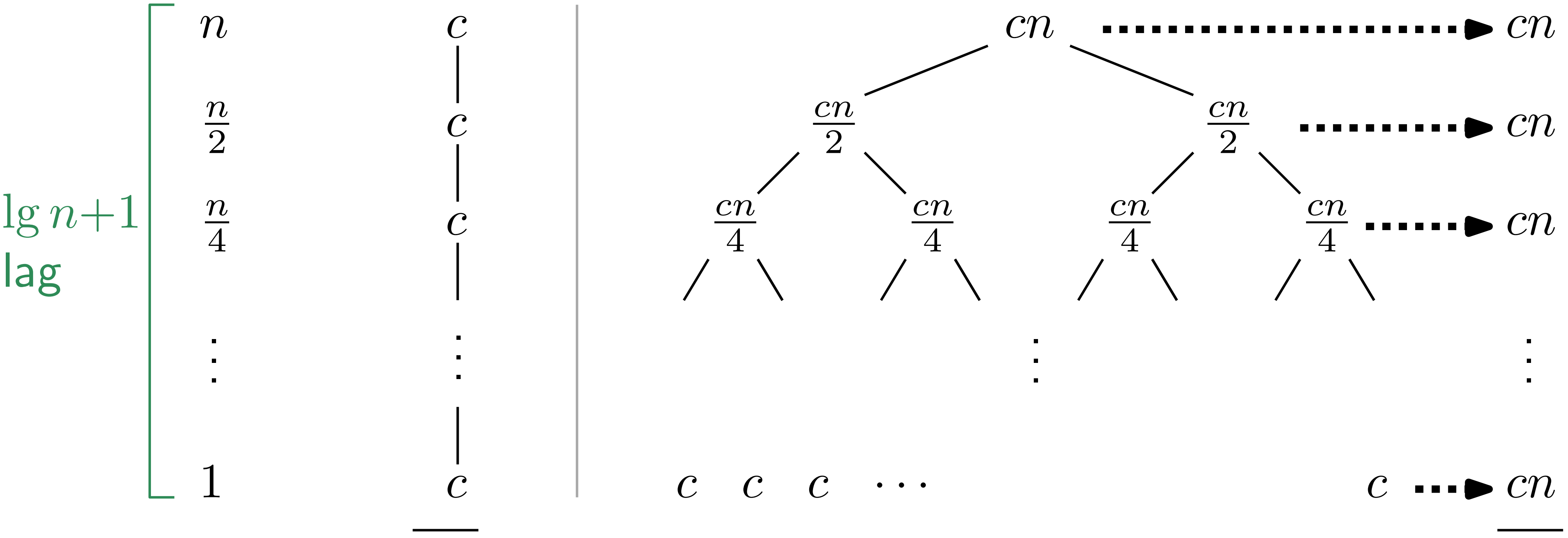
så $T(n) = O(n \log n)$.

Da $T(n) = \Omega(n \log n)$ og $T(n) = O(n \log n)$ har vi $T(n) = \Theta(n \log n)$.

Rekursivt definerede køretider

$$T_B(n) = \begin{cases} c, & n = 1 \\ T_B(n/2) + c, & n > 1 \end{cases} \qquad T_M(n) = \begin{cases} c, & n = 1 \\ 2T_M(n/2) + cn, & n > 1 \end{cases}$$

array-
str.



lg n+1
lag

n — c

$\frac{n}{2}$ — c

$\frac{n}{4}$ — c

$\vdots$ — $\vdots$

1 — c

$cn \rightarrow cn$

$\frac{cn}{2}$ $\frac{cn}{2}$ $\rightarrow cn$

$\frac{cn}{4}$ $\frac{cn}{4}$ $\frac{cn}{4}$ $\frac{cn}{4}$ $\rightarrow cn$

$\vdots$ $\vdots$

c c c $\cdots$ $c \rightarrow cn$

total: $T_B(n) = c \lg n + c$ $\qquad$ $T_M(n) = cn \lg n + cn$

Nestede vs. sekventielle løkker

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

Nestede vs. sekventielle løkker

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

skridt

c_1

c_2

c_3

c_4

Nestede vs. sekventielle løkker

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

skridt	max. gange
c_1	1
c_2	n
c_3	n^2
c_4	1

Nestede vs. sekventielle løkker

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

skridt	max. gange
c_1	1
c_2	n
c_3	n^2
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n^2 + c_4 = \Theta(n^2)$$

Nestede vs. sekventielle løkker

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

skridt	max. gange
c_1	1
c_2	n
c_3	n^2
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n^2 + c_4 = \boxed{\Theta(n^2)}$$

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

Nestede vs. sekventielle løkker

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

skridt	max. gange
c_1	1
c_2	n
c_3	n^2
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n^2 + c_4 = \boxed{\Theta(n^2)}$$

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

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

Nestede vs. sekventielle løkker

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

skridt	max. gange
c_1	1
c_2	n
c_3	n^2
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n^2 + c_4 = \boxed{\Theta(n^2)}$$

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

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

skridt	max. gange
c_1	1
c_2	n
c_3	n
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n + c_4 = \Theta(n)$$

Nestede vs. sekventielle løkker

```
nestedLoops($n$)
    $x = 0$
    for $i = 0$ to $n - 1$
        $x = x + 1$
        for $j = 0$ to $\xcancel{n - 1}$ $\sqrt{n} - 1$
            $x = x + 1$
    return $x$
```

skridt	max. gange
c_1	1
c_2	n
c_3	n^2
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n^2 + c_4 = \boxed{\Theta(n^2)}$$

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

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

skridt	max. gange
c_1	1
c_2	n
c_3	n
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n + c_4 = \Theta(n)$$

Nestede vs. sekventielle løkker

```
nestedLoops($n$)
    $x = 0$
    for $i = 0$ to $n - 1$
        $x = x + 1$    $\sqrt{n} - 1$
        for $j = 0$ to $\cancel{n - 1}$
            $x = x + 1$
    return $x$
```

skridt	max. gange
c_1	1
c_2	n
c_3	$\cancel{n^2}$ $n\sqrt{n}$
c_4	1

$$T(n) = \cancel{c_1 + c_2 n + c_3 n^2 + c_4 = \boxed{\Theta(n^2)}} \; \Theta(n\sqrt{n})$$

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

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

skridt	max. gange
c_1	1
c_2	n
c_3	n
c_4	1

$$T(n) = c_1 + c_2 n + c_3 n + c_4 = \Theta(n)$$

Nestede vs. sekventielle løkker

```
nestedLoops($n$)
    $x = 0$
    for $i = 0$ to $n - 1$
        $x = x + 1$   $\sqrt{n} - 1$
        for $j = 0$ to $\cancel{n - 1}$
            $x = x + 1$
    return $x$
```

skridt	max. gange
c_1	1
c_2	n
c_3	$\cancel{n^2}$ $n\sqrt{n}$
c_4	1

$$T(n) = \cancel{c_1 + c_2 n + c_3 n^2 + c_4 = \boxed{\Theta(n^2)}}\ \Theta(n\sqrt{n})$$

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

```
seqLoops($n$)
    $x = 0$
    for $i = 0$ to $n - 1$
        $x = x + 1$   $\sqrt{n} - 1$
    for $j = 0$ to $\cancel{n - 1}$
        $x = x + 1$
    return $x$
```

skridt	max. gange
c_1	1
c_2	n
c_3	$\cancel{n}$ $\sqrt{n}$
c_4	1

$$T(n) = c_1 + c_2 n + c_3 \underset{\sqrt{n}}{\cancel{n}} + c_4 = \Theta(n)$$

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i + 1] = A[i]$
      $i = i - 1$
    $A[i + 1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

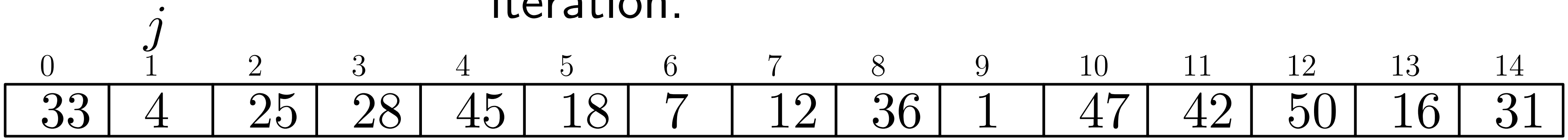
Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i + 1] = A[i]$
      $i = i - 1$
    $A[i + 1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
   for $j = 1$ to $n - 1$
      $key = A[j]$
      $i = j - 1$
      while $i \geq 0$ and $A[i] > key$
         $A[i + 1] = A[i]$
         $i = i - 1$
      $A[i + 1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	18	25	28	33	45	7	12	36	1	47	42	50	16	31

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

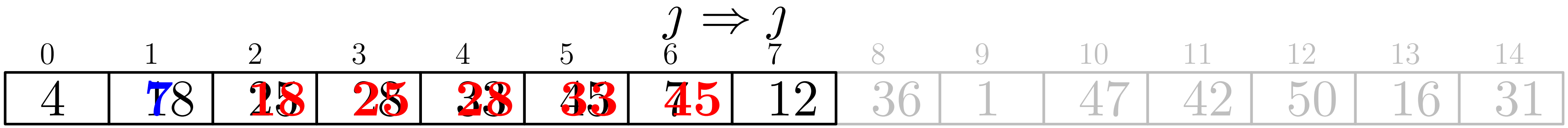
```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i + 1] = A[i]$
      $i = i - 1$
    $A[i + 1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

		j													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.



Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

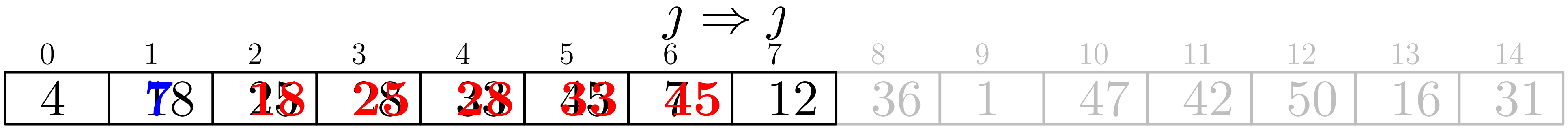
```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i + 1] = A[i]$
      $i = i - 1$
    $A[i + 1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

0	1 (j)	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.



Terminering: LI passer når løkken terminerer $\Longrightarrow$ Algoritmen virker

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	j
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50	

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i+1] = A[i]$
      $i = i - 1$
    $A[i+1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

	j													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.

						$j \Rightarrow$	j							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	18 **7**	25 **18**	28 **25**	33 **28**	45 **33**	7 **45**	12	36	1	47	42	50	16	31

Terminering: LI passer når løkken terminerer $\Longrightarrow$ Algoritmen virker

															j
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50	

	j													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i+1] = A[i]$
      $i = i - 1$
    $A[i+1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.

$j \Rightarrow j$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	18 **7**	25 **18**	28 **25**	33 **28**	45 **33**	7 **45**	12	36	1	47	42	50	16	31

Terminering: LI passer når løkken terminerer $\Longrightarrow$ Algoritmen virker

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

$j \Rightarrow j$

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

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i+1] = A[i]$
      $i = i - 1$
    $A[i+1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.

$j \Rightarrow j$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	78	18	25	28	33	45	12	36	1	47	42	50	16	31

Terminering: LI passer når løkken terminerer $\Longrightarrow$ Algoritmen virker

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

$j \Rightarrow j \Rightarrow j$

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

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i+1] = A[i]$
      $i = i - 1$
    $A[i+1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

j (over 1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.

$j \Rightarrow j$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	18 **7**	25 **18**	28 **25**	33 **28**	45 **33**	7 **45**	12	36	1	47	42	50	16	31

Terminering: LI passer når løkken terminerer $\Longrightarrow$ Algoritmen virker

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

$j \Rightarrow j \Rightarrow j \Rightarrow j$

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

Løkke-invarianter

Generelt: Egenskaber for variable og arrays i begyndelsen af hver iteration af en løkke, som bruges til at argumentere for korrekthed.

```
Insertion-Sort($A, n$)
  for $j = 1$ to $n - 1$
    $key = A[j]$
    $i = j - 1$
    while $i \geq 0$ and $A[i] > key$
      $A[i+1] = A[i]$
      $i = i - 1$
    $A[i+1] = key$
```

Løkke-invariant (LI): I begyndelsen af hver iteration af for-løkken indeholder $A[0 \ldots j-1]$ de samme tal som oprindeligt var i $A[0 \ldots j-1]$, men nu i sorteret rækkefølge.

Initialisering: LI passer i begyndelsen af første iteration.

j

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

Vedligeholdelse: LI passer i begyndelsen af én iteration $\Longrightarrow$ LI passer også i begyndelsen af den næste.

$j \Rightarrow j$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	18 7	25 18	28 25	33 28	45 33	7 45	12	36	1	47	42	50	16	31

Terminering: LI passer når løkken terminerer $\Longrightarrow$ Algoritmen virker

j

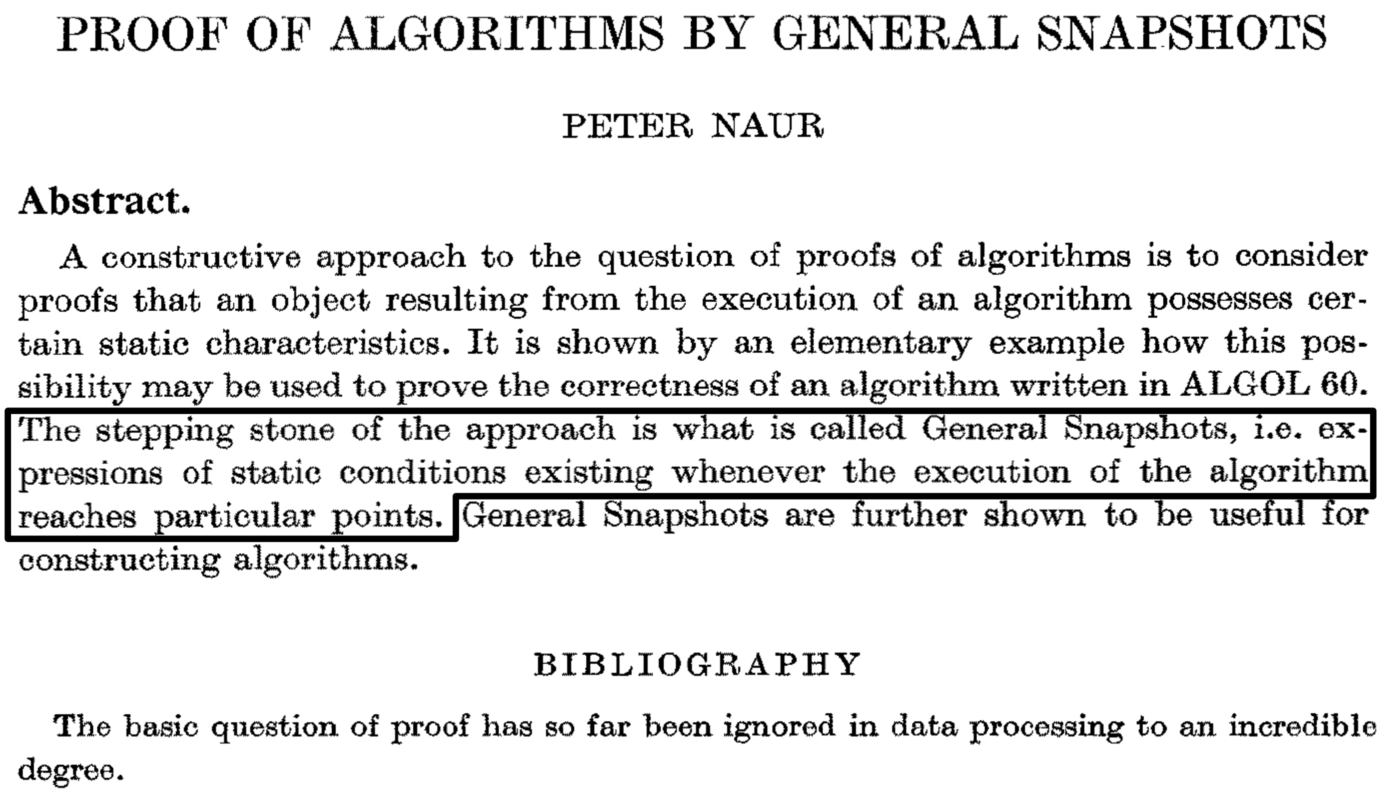
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

$j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j \Rightarrow j$

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

Bonus-info

Brugen af løkke-invarianter til at bevise korrekthed for algoritmer blev først beskrevet af Peter Naur (1928–2016) i en artikel fra 1966.



PROOF OF ALGORITHMS BY GENERAL SNAPSHOTS

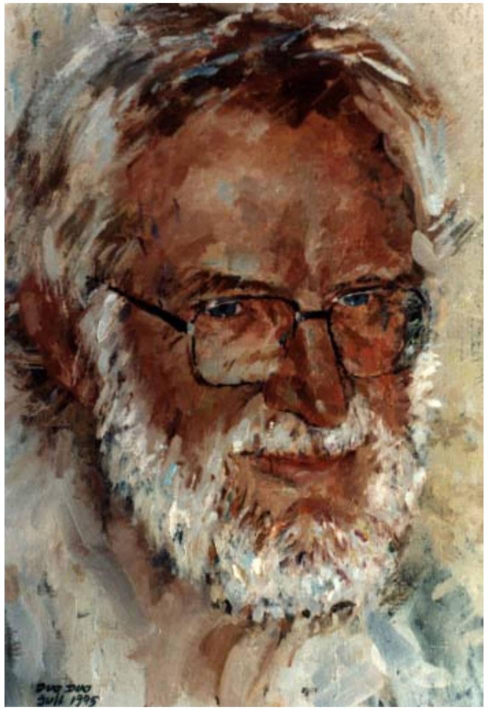
PETER NAUR

Abstract.

A constructive approach to the question of proofs of algorithms is to consider proofs that an object resulting from the execution of an algorithm possesses certain static characteristics. It is shown by an elementary example how this possibility may be used to prove the correctness of an algorithm written in ALGOL 60. The stepping stone of the approach is what is called General Snapshots, i.e. expressions of static conditions existing whenever the execution of the algorithm reaches particular points. General Snapshots are further shown to be useful for constructing algorithms.

BIBLIOGRAPHY

The basic question of proof has so far been ignored in data processing to an incredible degree.



Desuden opfandt han ordet *datalogi*, grundlagde DIKU i 1970 og modtog Turing-medaljen i 2005 (datalogiens “Nobel-pris”) for sit arbejde med at udvikle de første moderne/strukturerede programmingssprog.

Hvad har vi lært?

Pseudokode:

- Variable
- Arrays
- Funktionskald
- Indrykning
- if-else
- for- og while-løkker
- Rekursive funktioner

Hvad har vi lært?

Pseudokode:

- Variable
- Arrays
- Funktionskald
- Indrykning
- if-else
- for- og while-løkker
- Rekursive funktioner

Køretider
Tælle skridt
RAM-model
Asymptotisk notation
Rekursivt def. køretider
(Løkkeinvarianter)

Hvad har vi lært?

Pseudokode:

- Variable
- Arrays
- Funktionskald
- Indrykning
- if-else
- for- og while-løkker
- Rekursive funktioner

Køretider
Tælle skridt
RAM-model
Asymptotisk notation
Rekursivt def. køretider
(Løkkeinvarianter)

Algoritmer:

- Maksimum, $\Theta(n)$ tid
- Toppunkt, $\Theta(n)$ vs. $\Theta(\log n)$ tid
- Linear/sekventiel søgning, $\Theta(n)$ tid
- Binær søgning (iterativ og rekursiv), $\Theta(\log n)$ tid
- Insertion sort, $\Theta(n^2)$ tid, konstant ekstra plads
- Merge sort, $\Theta(n \log n)$ tid, $\Theta(n)$ ekstra plads
- Forskellige algoritmer fra øvelsestimer