

Divide and Conquer & The Master Theorem

Algorithms and Datastructures, F25, Lecture 3

Andreas Holck Høeg-Petersen

Department of Computer Science
Aalborg University

January 29, 2025

Opdateringer

- Løsninger på exercises kommer på et eller andet tidspunkt
- Fra evaluering:
 - ▶ Grupper?
 - ▶ Andet?

Outline

- 1 Divide and Conquer
- 2 Merge sort
- 3 Exercises
- 4 Quick sort
- 5 The Master Theorem



Outline

1 Divide and Conquer

2 Merge sort

3 Exercises

4 Quick sort

5 The Master Theorem



Divide and Conquer

Algoritmiske teknikker

Divide-and-conquer er en effektiv teknik til at designe effektive algoritmer til at løse komplekse problemer ved at bryde dem ned i mindre dele. Ofte giver det en asymptotiske køretid i $\Theta(n \log n)$.



Divide and Conquer

Algoritmiske teknikker

Divide-and-conquer er en effektiv teknik til at designe effektive algoritmer til at løse komplekse problemer ved at bryde dem ned i mindre dele. Ofte giver det en asymptotiske køretid i $\Theta(n \log n)$.

Metoden har overordnet set 3 skridt:

Divide and Conquer

Algoritmiske teknikker

Divide-and-conquer er en effektiv teknik til at designe effektive algoritmer til at løse komplekse problemer ved at bryde dem ned i mindre dele. Ofte giver det en asymptotiske køretid i $\Theta(n \log n)$.

Metoden har overordnet set 3 skridt:

Divide Del problemet op i et eller flere sub-problemer, der er mindre instanser af det samme problem

Divide and Conquer

Algoritmiske teknikker

Divide-and-conquer er en effektiv teknik til at designe effektive algoritmer til at løse komplekse problemer ved at bryde dem ned i mindre dele. Ofte giver det en asymptotiske køretid i $\Theta(n \log n)$.

Metoden har overordnet set 3 skridt:

Divide Del problemet op i et eller flere sub-problemer, der er mindre instanser af det samme problem

Conquer Løs sub-problemerne rekursivt

Divide and Conquer

Algoritmiske teknikker

Divide-and-conquer er en effektiv teknik til at designe effektive algoritmer til at løse komplekse problemer ved at bryde dem ned i mindre dele. Ofte giver det en asymptotiske køretid i $\Theta(n \log n)$.

Metoden har overordnet set 3 skridt:

Divide Del problemet op i et eller flere sub-problemer, der er mindre instanser af det samme problem

Conquer Løs sub-problemerne rekursivt

Combine Kombiner løsningerne på sub-problemerne til en løsning på det oprindelige problem

Divide and Conquer

Algoritmiske teknikker

Divide-and-conquer er en effektiv teknik til at designe effektive algoritmer til at løse komplekse problemer ved at bryde dem ned i mindre dele. Ofte giver det en asymptotiske køretid i $\Theta(n \log n)$.

Metoden har overordnet set 3 skridt:

Divide Del problemet op i et eller flere sub-problemer, der er mindre instanser af det samme problem

Conquer Løs sub-problemerne rekursivt

Combine Kombiner løsningerne på sub-problemerne til en løsning på det oprindelige problem

Divide and Conquer

Algoritmiske teknikker

Divide-and-conquer er en effektiv teknik til at designe effektive algoritmer til at løse komplekse problemer ved at bryde dem ned i mindre dele. Ofte giver det en asymptotiske køretid i $\Theta(n \log n)$.

Metoden har overordnet set 3 skridt:

Divide Del problemet op i et eller flere sub-problemer, der er mindre instanser af det samme problem

Conquer Løs sub-problemerne rekursivt

Combine Kombiner løsningerne på sub-problemerne til en løsning på det oprindelige problem

Hvis problemet er småt nok (**base case**), løses det uden videre. Ellers (**recursive case**) fortsætter man rekursionen.

Divide and Conquer

Rekursion???

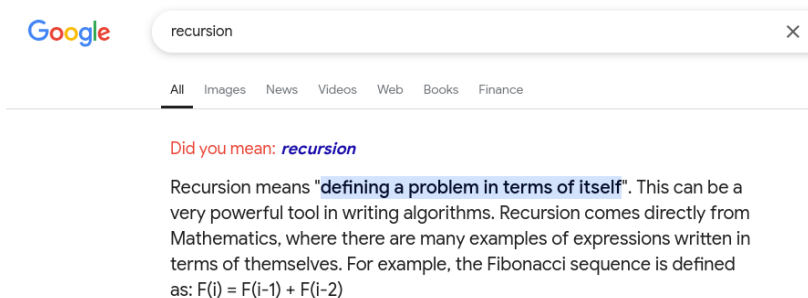


Figure: Google søgning på 'recursion'

Divide and Conquer

Rekursion???

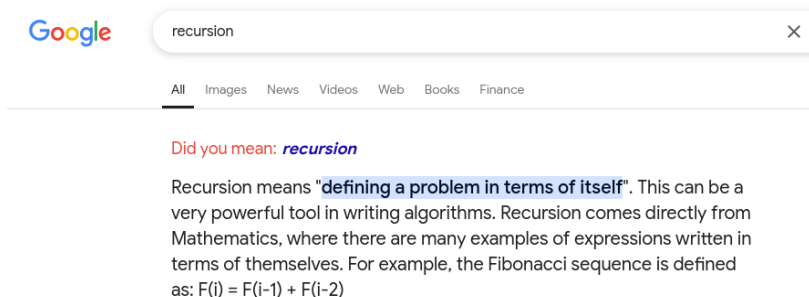


Figure: Google søgning på 'recursion'

Example (Fibonacci-sekvensen)

Det næste tal i Fibonacci-sekvensen er givet ved at summere de to foregående elementer. Den starter med 1, 1, 2, 3, 5, 8, 13, Men den kan dermed også defineres rekursivt, således at det i 'ende element er givet ved $F(i) = F(i - 1) + F(i - 2)$.

Divide and Conquer

Algoritmisk rekursion

I algoritmisk forstand forstår vi en rekursion $T(n)$ således, at der for en tilpas stor konstant n_0 skal gælde følgende:

- 1 For alle $n < n_0$ har vi at $T(n) = \Theta(1)$ — dvs. $T(n)$ er konstant
- 2 For alle $n \geq n_0$ må alle stier af rekursionen ende i en defineret base case inden for et **endeligt** antal rekursive kald.

Divide and Conquer

Algoritmisk rekursion

I algoritmisk forstand forstår vi en rekursion $T(n)$ således, at der for en tilpas stor konstant n_0 skal gælde følgende:

- 1 For alle $n < n_0$ har vi at $T(n) = \Theta(1)$ — dvs. $T(n)$ er konstant
- 2 For alle $n \geq n_0$ må alle stier af rekursionen ende i en defineret base case inden for et **endeligt** antal rekursive kald.

I kurset her gælder det for alle rekursioner, vi ser på, men det er værd at have in mente, hvis I selv designer algoritmer, som gør brug af rekursion.

Divide and Conquer

Eksempler

I dag skal vi se på to eksempler på divide-and-conquer-algoritmer:

- Merge sort
- Quicksort

Outline

- 1 Divide and Conquer
- 2 Merge sort
- 3 Exercises
- 4 Quick sort
- 5 The Master Theorem



Merge sort

Den kender I jo!

- En af de mest berømte og benyttede sorteringsalgoritmer — og en af de første til at blive implementeret i en computer (ca. 1945 af John von Neumann)
- Ide:
 - Divide** Opdel sekvensen i to lige store sub-sekvenser og kald algoritmen rekursivt
 - Conquer** Når algoritmen modtager en sekvens med kun et element, returner det trivielt sorterede element
 - Combine** Kombiner de sorterede sub-sekvenser, så sorteringsrækkefølgen overholdes

Merge sort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$

Merge-Sort(A, p, r)

```
1  if  $p \geq r$ 
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$ 
4  Merge-Sort( $A, p, q$ )
5  Merge-Sort( $A, q + 1, r$ )
6  Merge( $A, p, q, r$ )
```

Merge sort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså Merge-Sort($A, 1, n$)

Merge-Sort(A, p, r)

```
1  if  $p \geq r$ 
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$ 
4  Merge-Sort( $A, p, q$ )
5  Merge-Sort( $A, q + 1, r$ )
6  Merge( $A, p, q, r$ )
```

Merge sort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså $\text{Merge-Sort}(A, 1, n)$
- I linie 3 finder vi midtpunktet mellem p og r

Merge-Sort(A, p, r)

```
1  if  $p \geq r$ 
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$ 
4  Merge-Sort( $A, p, q$ )
5  Merge-Sort( $A, q + 1, r$ )
6  Merge( $A, p, q, r$ )
```

Merge sort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså $\text{Merge-Sort}(A, 1, n)$
- I linie 3 finder vi midtpunktet mellem p og r
- I linie 4 og 5 kalder vi rekursivt for den ene og anden halvdel af sekvensen

Merge-Sort(A, p, r)

```
1  if  $p \geq r$ 
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$ 
4  Merge-Sort( $A, p, q$ )
5  Merge-Sort( $A, q + 1, r$ )
6  Merge( $A, p, q, r$ )
```

Merge sort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså Merge-Sort($A, 1, n$)
- I linie 3 finder vi midtpunktet mellem p og r
- I linie 4 og 5 kalder vi rekursivt for den ene og anden halvdel af sekvensen
- I linie 6 kombinerer ('merger') vi de to halvdele sammen

Merge-Sort(A, p, r)

```
1  if  $p \geq r$ 
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$ 
4  Merge-Sort( $A, p, q$ )
5  Merge-Sort( $A, q + 1, r$ )
6  Merge( $A, p, q, r$ )
```

Merge sort

Eksempel



Merge sort

Merge-operationen

Merge-operationen er et rædselsfuldt monster i CLRS...!

```
MERGE( $A, p, q, r$ )
1   $n_L = q - p + 1$            // length of  $A[p : q]$ 
2   $n_R = r - q$                // length of  $A[q + 1 : r]$ 
3  let  $L[0 : n_L - 1]$  and  $R[0 : n_R - 1]$  be new arrays
4  for  $i = 0$  to  $n_L - 1$        // copy  $A[p : q]$  into  $L[0 : n_L - 1]$ 
5       $L[i] = A[p + i]$ 
6  for  $j = 0$  to  $n_R - 1$        // copy  $A[q + 1 : r]$  into  $R[0 : n_R - 1]$ 
7       $R[j] = A[q + j + 1]$ 
8   $i = 0$                        //  $i$  indexes the smallest remaining element in  $L$ 
9   $j = 0$                        //  $j$  indexes the smallest remaining element in  $R$ 
10  $k = p$                        //  $k$  indexes the location in  $A$  to fill
11 // As long as each of the arrays  $L$  and  $R$  contains an unmerged element,
12 //   copy the smallest unmerged element back into  $A[p : r]$ .
13 while  $i < n_L$  and  $j < n_R$ 
14     if  $L[i] \leq R[j]$ 
15          $A[k] = L[i]$ 
16          $i = i + 1$ 
17     else  $A[k] = R[j]$ 
18          $j = j + 1$ 
19          $k = k + 1$ 
20 // Having gone through one of  $L$  and  $R$  entirely, copy the
21 //   remainder of the other to the end of  $A[p : r]$ .
22 while  $i < n_L$ 
23      $A[k] = L[i]$ 
24      $i = i + 1$ 
25      $k = k + 1$ 
26 while  $j < n_R$ 
27      $A[k] = R[j]$ 
28      $j = j + 1$ 
29      $k = k + 1$ 
```

Figure: Ew!!

Merge sort

Merge-operationen

En lidt mere venlig version kunne se sådan her ud:

```
Merge( $A, p, q, r$ )
1  let  $B[0 : r - p]$  be a new array with 0-index
2  for  $i = 0$  to  $r - p$ 
3       $B[i] = A[i + p]$ 
4   $i = 0, j = (r - q)$ 
5  for  $k = p$  to  $r$ 
6      if  $(i + p) > q$ 
7           $A[k] = B[j]$ 
8           $j = j + 1$ 
9      elseif  $(j + q) > r$ 
10          $A[k] = B[i]$ 
11          $i = i + 1$ 
12     elseif  $B[j] < B[i]$ 
13          $A[k] = B[j]$ 
14          $j = j + 1$ 
15     else
16          $A[k] = B[i]$ 
17          $i = i + 1$ 
```



Merge sort

Merge-operationen

Og her endda med forståelige navne:

```
Merge(A, p, q, r)
1  low = p, mid = q + 1, high = r + 1
2  let B[0 : high - low] be a new array with 0-index
3  for i = 0 to B.length
4      B[i] = A[i + low]

5  i = 0, j = (mid - low)
6  for k = low to high
7      if (i + low) ≥ mid
8          A[k] = B[j]
9          j = j + 1
10     elseif (j + low) ≥ high
11         A[k] = B[i]
12         i = i + 1
13     elseif B[j] < B[i]
14         A[k] = B[j]
15         j = j + 1
16     else
17         A[k] = B[i]
18         i = i + 1
```



Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1  low = p, mid = q + 1, high = r + 1
2  let  $B[0 : \textit{high} - \textit{low}]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.\textit{length}$ 
4       $B[i] = A[i + \textit{low}]$ 

5   $i = 0, j = (\textit{mid} - \textit{low})$ 
6  for  $k = \textit{low}$  to  $\textit{high}$ 
7      if  $(i + \textit{low}) \geq \textit{mid}$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + \textit{low}) \geq \textit{high}$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $\textit{low} = p, \textit{mid} = q + 1$ og $\textit{high} = r + 1$

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    $\text{a new array with 0-index}$ 
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$
- Hvis i er forbi midten, tag næste element fra $B[j : r - p]$ og inkrementer j

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$
- Hvis i er forbi midten, tag næste element fra $B[j : r - p]$ og inkrementer j
- Hvis j er forbi slutningen, tag næste element fra $B[0 : q - p]$ og inkrementer i

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1  low = p, mid = q + 1, high = r + 1
2  let B[0 : high - low] be
   a new array with 0-index
3  for i = 0 to B.length
4      B[i] = A[i + low]

5  i = 0, j = (mid - low)
6  for k = low to high
7      if (i + low) ≥ mid
8          A[k] = B[j]
9          j = j + 1
10     elseif (j + low) ≥ high
11         A[k] = B[i]
12         i = i + 1
13     elseif B[j] < B[i]
14         A[k] = B[j]
15         j = j + 1
16     else
17         A[k] = B[i]
18         i = i + 1
```

- Vi lader $low = p$, $mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$
- Hvis i er forbi midten, tag næste element fra $B[j : r - p]$ og inkrementer j
- Hvis j er forbi slutningen, tag næste element fra $B[0 : q - p]$ og inkrementer i
- Hvis $B[j]$ er lavere end $B[i]$, sæt $A[k]$ til $B[j]$ og inkrementer j

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$
- Hvis i er forbi midten, tag næste element fra $B[j : r - p]$ og inkrementer j
- Hvis j er forbi slutningen, tag næste element fra $B[0 : q - p]$ og inkrementer i
- Hvis $B[j]$ er lavere end $B[i]$, sæt $A[k]$ til $B[j]$ og inkrementer j
- Ellers, sæt $A[k]$ til $B[i]$ og inkrementer i

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$
- Hvis i er forbi midten, tag næste element fra $B[j : r - p]$ og inkrementer j
- Hvis j er forbi slutningen, tag næste element fra $B[0 : q - p]$ og inkrementer i
- Hvis $B[j]$ er lavere end $B[i]$, sæt $A[k]$ til $B[j]$ og inkrementer j
- Ellers, sæt $A[k]$ til $B[i]$ og inkrementer i
- Vi har nu lagt elementerne fra B tilbage i A i sorteret rækkefølge!

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$
- Hvis i er forbi midten, tag næste element fra $B[j : r - p]$ og inkrementer j
- Hvis j er forbi slutningen, tag næste element fra $B[0 : q - p]$ og inkrementer i
- Hvis $B[j]$ er lavere end $B[i]$, sæt $A[k]$ til $B[j]$ og inkrementer j
- Ellers, sæt $A[k]$ til $B[i]$ og inkrementer i
- Vi har nu lagt elementerne fra B tilbage i A i sorteret rækkefølge!
- Forskellen fra CLRS er, at vi samler L og R i et enkelt array B

Merge sort

Merge-operationen

Merge(A, p, q, r)

```
1   $low = p, mid = q + 1, high = r + 1$ 
2  let  $B[0 : high - low]$  be
    a new array with 0-index
3  for  $i = 0$  to  $B.length$ 
4       $B[i] = A[i + low]$ 

5   $i = 0, j = (mid - low)$ 
6  for  $k = low$  to  $high$ 
7      if  $(i + low) \geq mid$ 
8           $A[k] = B[j]$ 
9           $j = j + 1$ 
10     elseif  $(j + low) \geq high$ 
11          $A[k] = B[i]$ 
12          $i = i + 1$ 
13     elseif  $B[j] < B[i]$ 
14          $A[k] = B[j]$ 
15          $j = j + 1$ 
16     else
17          $A[k] = B[i]$ 
18          $i = i + 1$ 
```

- Vi lader $low = p, mid = q + 1$ og $high = r + 1$
- Kopier $A[p : r]$ til $B[0 : r - p]$
- i og j peger på første og anden del af B
- k løber igennem alle indicies i $A[p : r]$
- Hvis i er forbi midten, tag næste element fra $B[j : r - p]$ og inkrementer j
- Hvis j er forbi slutningen, tag næste element fra $B[0 : q - p]$ og inkrementer i
- Hvis $B[j]$ er lavere end $B[i]$, sæt $A[k]$ til $B[j]$ og inkrementer j
- Ellers, sæt $A[k]$ til $B[i]$ og inkrementer i
- Vi har nu lagt elementerne fra B tilbage i A i sorteret rækkefølge!
- Forskellen fra CLRS er, at vi samler L og R i et enkelt array B
- ... og at vi klarer resten i et enkelt loop (fremfor 3, eew!)

Merge sort

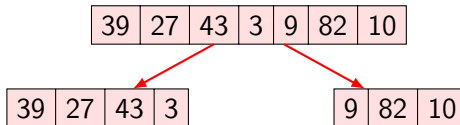
Example

39	27	43	3	9	82	10
----	----	----	---	---	----	----



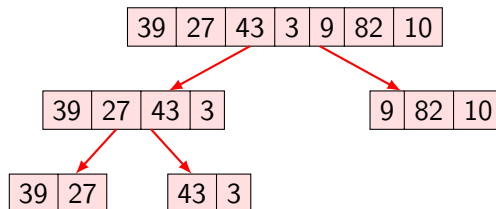
Merge sort

Example



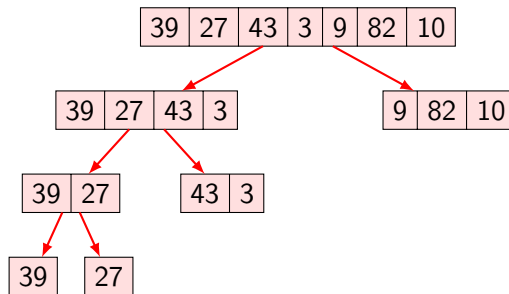
Merge sort

Example



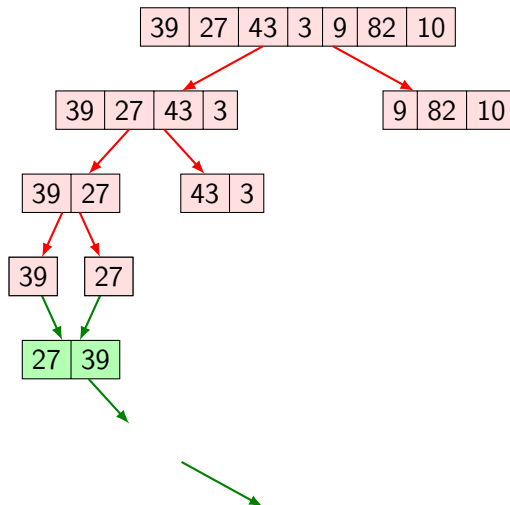
Merge sort

Example



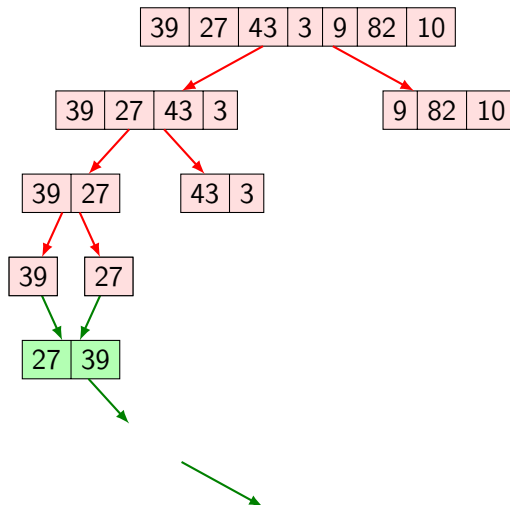
Merge sort

Example



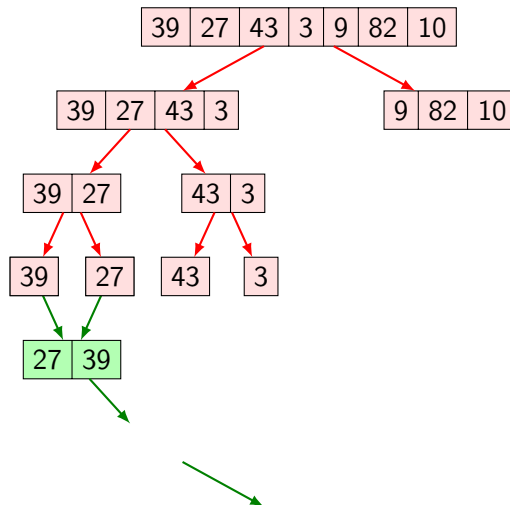
Merge sort

Example



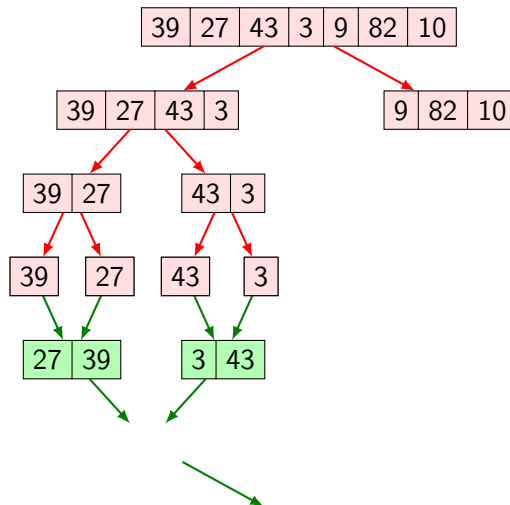
Merge sort

Example



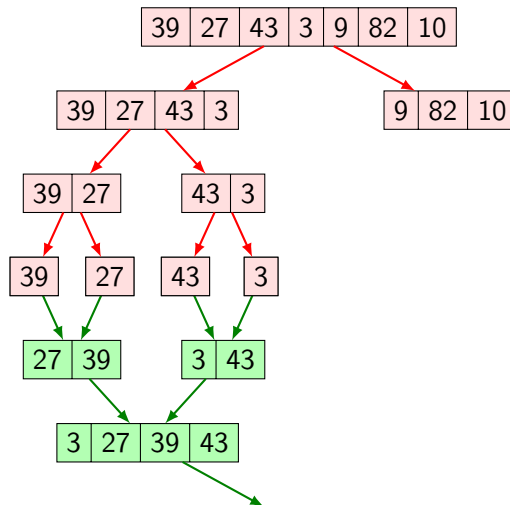
Merge sort

Example



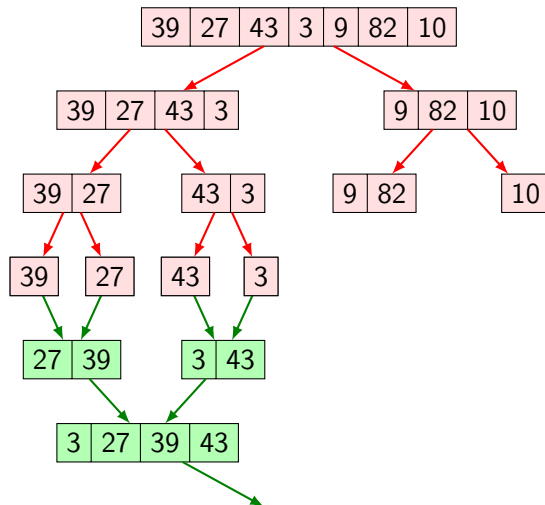
Merge sort

Example



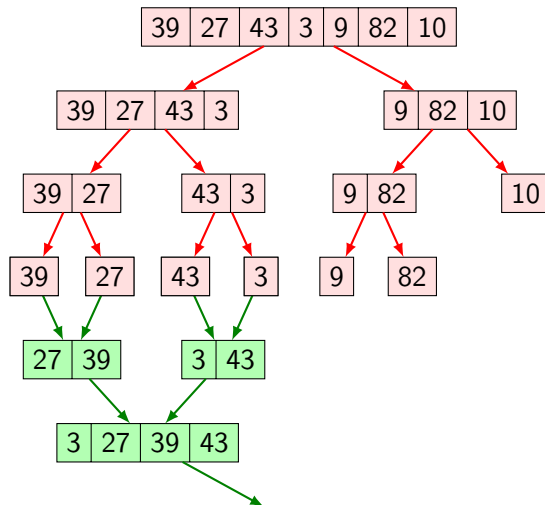
Merge sort

Example



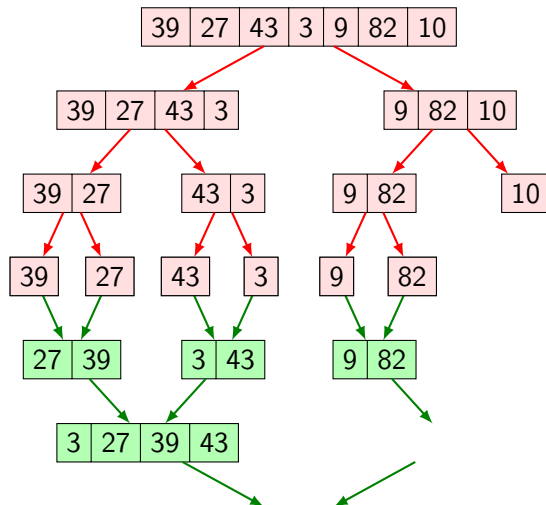
Merge sort

Example



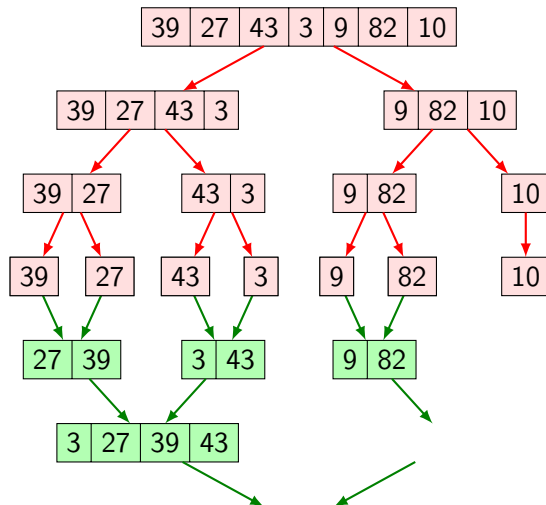
Merge sort

Example



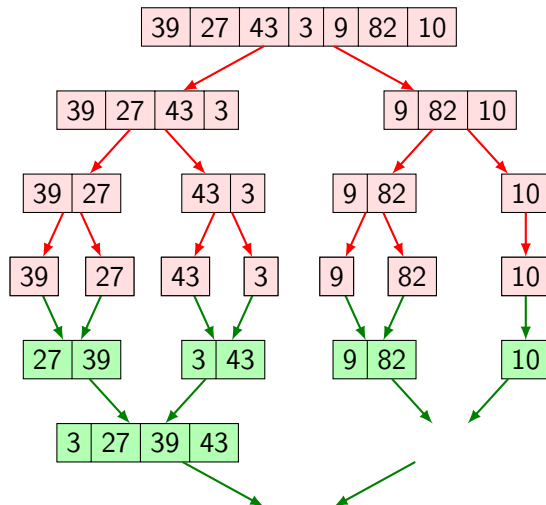
Merge sort

Example



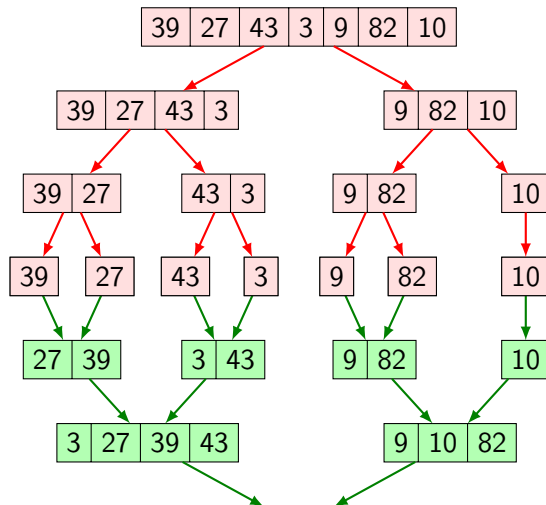
Merge sort

Example



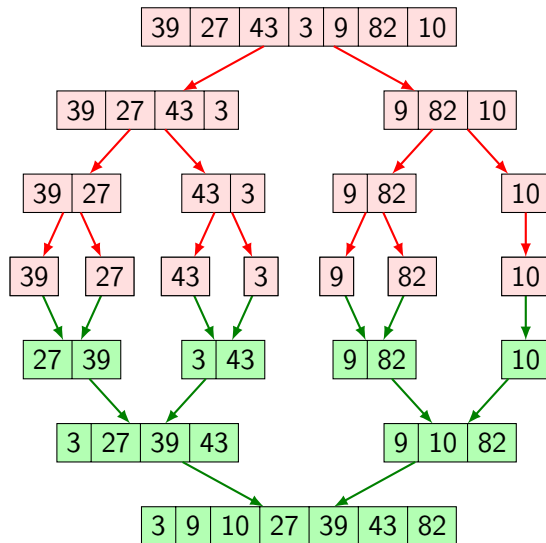
Merge sort

Example



Merge sort

Example



Quicksort

Endnu en klassiker

Quicksort er en anden meget populær sorteringsalgoritme, der ligeledes følger divide-and-conquer-metoden. I worst-case er dens køretid $\Theta(n^2)$, men i praksis er den typisk hurtigere end de fleste andre alternativer — og med en simpel modifikation, kan man (næsten) sikre sig, at køretiden er $\Theta(n \log n)$. Derudover er dens pladsforbrug mindre end for merge sort, og implementationen er noget simplere.

Quicksort

Divide-and-conquer

De tre dele af divide-and-conquer-metoden for quicksort er:



Quicksort

Divide-and-conquer

De tre dele af divide-and-conquer-metoden for quicksort er:

Divide Vælg et **pivot element** p , og del input sekvensen op i en 'lav' del og en 'høj' del således, alt i den lave del er mindre end p og alt i den høje er større end eller lig med p . Indsæt p , så den skiller de to.



Quicksort

Divide-and-conquer

De tre dele af divide-and-conquer-metoden for quicksort er:

Divide Vælg et **pivot element** p , og del input sekvensen op i en 'lav' del og en 'høj' del således, alt i den lave del er mindre end p og alt i den høje er større end eller lig med p . Indsæt p , så den skiller de to.

Conquer Kald quicksort rekursivt på de to halvdele.



Quicksort

Divide-and-conquer

De tre dele af divide-and-conquer-metoden for quicksort er:

Divide Vælg et **pivot element** p , og del input sekvensen op i en 'lav' del og en 'høj' del således, alt i den lave del er mindre end p og alt i den høje er større end eller lig med p . Indsæt p , så den skiller de to.

Conquer Kald quicksort rekursivt på de to halvdele.

Combine Her behøver vi ikke gøre noget, for når vi når til bunds i rekursionen, så er begge sub-arrays sorterede.

Quicksort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r
hvor $1 \leq p \leq r \leq n$

QuickSort(A, p, r)

1 if $p < r$

2 $q = \text{Partition}(p, r)$

3 QuickSort($A, p, q - 1$)

4 QuickSort($A, q + 1, r$)

Quicksort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså $\text{QuickSort}(A, 1, n)$

```
QuickSort( $A, p, r$ )  
1  if  $p < r$   
2       $q = \text{Partition}(p, r)$   
3      QuickSort( $A, p, q - 1$ )  
4      QuickSort( $A, q + 1, r$ )
```

Quicksort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså $\text{QuickSort}(A, 1, n)$
- I linie 2 kalder vi proceduren Partition, som deler A i to og returnerer indexet på pivot-elementet

```
QuickSort( $A, p, r$ )
```

```
1  if  $p < r$ 
```

```
2       $q = \text{Partition}(p, r)$ 
```

```
3      QuickSort( $A, p, q - 1$ )
```

```
4      QuickSort( $A, q + 1, r$ )
```

Quicksort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså $\text{QuickSort}(A, 1, n)$
- I linie 2 kalder vi proceduren Partition, som deler A i to og returnerer indexet på pivot-elementet
- I linie 3 og 4 kalder vi rekursivt for den ene og anden del af sekvensen

```
QuickSort( $A, p, r$ )
```

```
1  if  $p < r$ 
```

```
2       $q = \text{Partition}(p, r)$ 
```

```
3      QuickSort( $A, p, q - 1$ )
```

```
4      QuickSort( $A, q + 1, r$ )
```


Quicksort

Pseudo-kode del 1

- Input: en sekvens $A[1 : n]$ og to **indicies** p, r hvor $1 \leq p \leq r \leq n$
- Ved første kald er $p = 1$ og $r = n$, altså $\text{QuickSort}(A, 1, n)$
- I linie 2 kalder vi proceduren Partition, som deler A i to og returnerer indexet på pivot-elementet
- I linie 3 og 4 kalder vi rekursivt for den ene og anden del af sekvensen
- Og så behøver vi ikke gøre mere!

```
QuickSort( $A, p, r$ )  
1  if  $p < r$   
2       $q = \text{Partition}(p, r)$   
3      QuickSort( $A, p, q - 1$ )  
4      QuickSort( $A, q + 1, r$ )
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r]$

Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r]$
- i er det sidste index i den lave del, j er det første index i den høje del

Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r]$
- i er det sidste index i den lave del, j er det første index i den høje del
- Vi løber igennem alle elementer, bortset fra pivot-elementet

Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```



Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r]$
- i er det sidste index i den lave del, j er det første index i den høje del
- Vi løber igennem alle elementer, bortset fra pivot-elementet
- I linie 4 tjekker vi, om $A[j]$ hører til i den lave del (er mindre end x)

Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r]$
- i er det sidste index i den lave del, j er det første index i den høje del
- Vi løber igennem alle elementer, bortset fra pivot-elementet
- I linie 4 tjekker vi, om $A[j]$ hører til i den lave del (er mindre end x)
- I linie 5 'gør vi plads' i den lave ende ved at inkrementere i (bemærk at på dette tidspunkt er $A[i] > x$)

Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r]$
- i er det sidste index i den lave del, j er det første index i den høje del
- Vi løber igennem alle elementer, bortset fra pivot-elementet
- I linie 4 tjekker vi, om $A[j]$ hører til i den lave del (er mindre end x)
- I linie 5 'gør vi plads' i den lave ende ved at inkrementere i (bemærk at på dette tidspunkt er $A[i] > x$)
- I linie 6 bytter vi $A[j]$ (som er mindre end x) ud med $A[i]$ (som er højere end x)

Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```



Quicksort

Pseudo-kode del 2

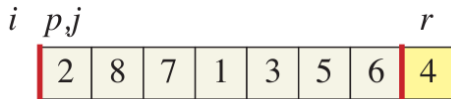
- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r]$
- i er det sidste index i den lave del, j er det første index i den høje del
- Vi løber igennem alle elementer, bortset fra pivot-elementet
- I linie 4 tjekker vi, om $A[j]$ hører til i den lave del (er mindre end x)
- I linie 5 'gør vi plads' i den lave ende ved at inkrementere i (bemærk at på dette tidspunkt er $A[i] > x$)
- I linie 6 bytter vi $A[j]$ (som er mindre end x) ud med $A[i]$ (som er højere end x)
- Efter loopet flytter vi x til den første plads i den høje del, $A[i + 1]$ — dermed er alt i $A[p : i] \leq x$ og alt i $A[i + 2 : r] \geq x$

Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```


Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 0$ og $j = 1$
- Vi sammenligner $A[j] = 2$ med **pivot-elementet** og ser, at det er mindre, så vi inkrementerer i og bytter $A[i]$ med $A[j]$ (men da $i = j = 1$ sker der ingenting lige nu). Når loopet fortsætter, inkrementeres j

 $\text{Partition}(A, p, r)$
$$1 \quad x = A[r]$$
$$2 \quad i = p - 1$$

```
3  for  $j = p$  to  $r - 1$ 
```

4 **if** $A[j] \leq x$

5 $i = i + 1$

```

6         exchange A[i] with A[j]

```

```

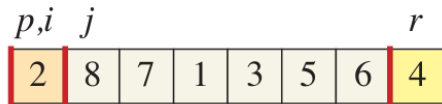
7  exchange  $A[i + 1]$  with  $x$ 

```

```
8 return  $i + 1$ 
```

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 1$ og $j = 2$
- Vi sammenligner $A[j] = 8$ med 4 og ser, at det er større, så vi gør intet andet end at inkrementere j


$$\text{Partition}(A, p, r)$$

```

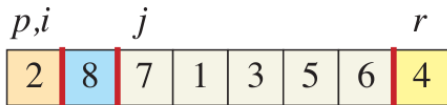
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 

```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 1$ og $j = 3$
- Igen, vi ser $A[j] = 7$ er større end 4 og inkrementerer blot j



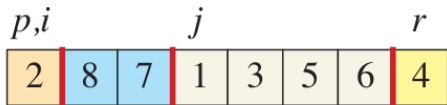
Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 1$ og $j = 4$
- Nu ser vi $A[j] = 1$ som er mindre end vores pivot-element, så vi inkrementerer i og bytter plads på $A[i]$ og $A[j]$ inden j inkrementeres



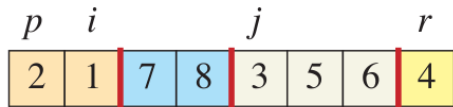
Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 2$ og $j = 5$
- $A[j] = 3$ er mindre end 4, så vi øger i med 1 og bytter 7 og 3 (altså $A[3]$ og $A[5]$)



Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 3$ og $j = 6$
- For $j = 6$ er $A[j] = 5$, hvilket er større end 4, så loopet kan fortsætte



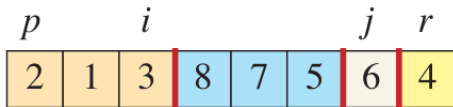
Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 3$ og $j = 7$
- ... og igen, 6 er større end 4, så vi gør ikke noget



Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 3$ og $j = 8$
- Til sidst bytter vi $A[i + 1] = 8$ med vores pivot-element, og dermed er alt i $A[p : i]$ mindre end (eller lig med) 4, mens alt i $A[i + 2 : r]$ er større end 4



Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```


Quicksort

Pseudo-kode del 2

- Vælg **pivot-elementet** x til at være det sidste i sekvensen, $A[r] = 4$
- Vi har $i = 3$ og $j = 8$
- Bum! Nu kan vi returnere indexet på pivot-elementet og gentage proceduren rekursivt for de to sub-arrays



Partition(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $x$ 
8  return  $i + 1$ 
```

Outline

- 1 Divide and Conquer
- 2 Merge sort
- 3 Exercises
- 4 Quick sort
- 5 The Master Theorem



Exercises!

Yay!



AALBORG
UNIVERSITET

Outline

- 1 Divide and Conquer
- 2 Merge sort
- 3 Exercises
- 4 Quick sort
- 5 The Master Theorem



Outline

- 1 Divide and Conquer
- 2 Merge sort
- 3 Exercises
- 4 Quick sort
- 5 The Master Theorem



Dagens temaer

Opsummering

- Vi har mødt vores første sorteringsalgoritme — Insertion-Sort!
 - ▶ Simpel at implementere og forstå
 - ▶ God til næsten sorterede sekvenser
 - ▶ Den asymptotiske worst case køretid er kvadratisk
- Loop invarianter og korrekthed
 - ▶ Initialization, maintenance og termination
- Asymptotisk analyse og notation
 - ▶ O, Ω, Θ



Tak for i dag!

Flere exercises..

Den bedste måde ikke at snyde sig selv på er lave exercises!

