Divide and Conquer & The Master Theorem

Algorithms and Datastructures, F25, Lecture 3

Andreas Holck Høeg-Petersen

Department of Computer Science
Aalborg University

January 21, 2025

AALBORG UNIVERSITET

Opdateringer

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

Outline

Outline

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

AALBORG UNIVERSITET

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

AALBORG UNIVERSITET

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

AALBORG UNIVERSITET

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???

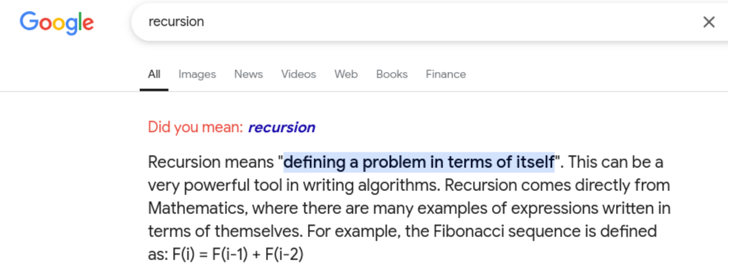

Figure: Google søgning på 'recursion'

Divide and Conquer

Rekursion???

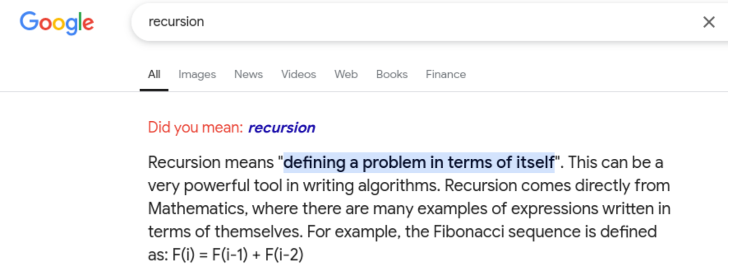

Figure: Google søgning på 'recursion'

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 elementer 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

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)
if p ≥ r
        return
q = ⌊(p + r)/2⌋
Merge-Sort(A, p, q)
Merge-Sort(A, q + 1, r)
Merge(A, p, q, r)
```

AALBORG UNIVERSITET

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)
if p ≥ r
        return
q = ⌊(p + r)/2⌋
Merge-Sort(A, p, q)
Merge-Sort(A, q + 1, r)
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

```
Merge-Sort(A, p, r)
1  if p ≥ r
2      return
3  q = ⌊(p + r)/2⌋
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

```
Merge-Sort(A, p, r)
1  if p ≥ r
2      return
3  q = ⌊(p + r)/2⌋
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 5 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)
n_L = q − p + 1          // length of A[p : q]
n_R = r − q              // length of A[q + 1 : r]
let L[0 : n_L − 1] and R[0 : n_R − 1] be new arrays
for i = 0 to n_L − 1     // copy A[p : q] into L[0 : n_L − 1]
    L[i] = A[p + i]
for j = 0 to n_R − 1     // copy A[q + 1 : r] into R[0 : n_R − 1]
    R[j] = A[q + j + 1]
i = 0                    // i indexes the smallest remaining element in L
j = 0                    // j indexes the smallest remaining element in R
k = p                    // k indexes the location in A to fill
// As long as each of the arrays L and R contains an unmerged element,
//      copy the smallest unmerged element back into A[p : r].
while i < n_L and j < n_R
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
    k = k + 1
// Having gone through one of L and R entirely, copy the
//      remainder of the other to the end of A[p : r].
while i < n_L
    A[k] = L[i]
    i = i + 1
    k = k + 1
while j < n_R
    A[k] = R[j]
    j = j + 1
    k = k + 1
```

Figure: Ew!!

Merge sort

Merge-operationen

En lidt mere venlig version kunne se sådan her ud:

```
Merge(A, p, q, r)
let B[0 : r − p] be a new array with 0-index
for i = 0 to r − p
    B[i] = A[i + p]
i = 0, j = (r − q)
for k = p to r
    if (i + p) > q
        A[k] = B[j]
        j = j + 1
    elseif (j + q) > r
        A[k] = B[i]
        i = i + 1
    elseif B[j] < B[i]
        A[k] = B[j]
        j = j + 1
    else
        A[k] = B[i]
        i = i + 1
```

Merge sort

Merge-operationen

Og her endda med forståelige navne:

```
Merge(A, low, mid, high)
let B[0 : high − low] be a new array with 0-index
for i = 0 to B.length
    B[i] = A[i + low]

i = 0, j = (high − mid)
for k = low to high
    if (i + low) > mid
        A[k] = B[j]
        j = j + 1
    elseif (j + mid) > high
        A[k] = B[i]
        i = i + 1
    elseif B[j] < B[i]
        A[k] = B[j]
        j = j + 1
    else
        A[k] = B[i]
        i = i + 1
```

Merge sort

Merge-operationen

```
Merge(A, low, mid, high)
  let B[0 : high − low] be
        a new array with 0-index
  for i = 0 to B.length
        B[i] = A[i + low]

  i = 0, j = (high − mid)
  for k = low to high
        if (i + low) > mid
              A[k] = B[j]
              j = j + 1
        elseif (j + mid) > high
              A[k] = B[i]
              i = i + 1
        elseif B[j] < B[i]
              A[k] = B[j]
              j = j + 1
        else
              A[k] = B[i]
              i = i + 1
```

- Vi lader $low = p$, $mid = q$ og $high = r$

Merge sort

Merge-operationen

```
Merge(A, low, mid, high)
  let B[0 : high − low] be
        a new array with 0-index
  for i = 0 to B.length
        B[i] = A[i + low]

  i = 0, j = (high − mid)
  for k = low to high
        if (i + low) > mid
              A[k] = B[j]
              j = j + 1
        elseif (j + mid) > high
              A[k] = B[i]
              i = i + 1
        elseif B[j] < B[i]
              A[k] = B[j]
              j = j + 1
        else
              A[k] = B[i]
              i = i + 1
```

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

AALBORG UNIVERSITET

Merge sort

Merge-operationen

```
Merge(A, low, mid, high)
let B[0 : high − low] be
        a new array with 0-index
for i = 0 to B.length
        B[i] = A[i + low]

i = 0, j = (high − mid)
for k = low to high
        if (i + low) > mid
                A[k] = B[j]
                j = j + 1
        elseif (j + mid) > high
                A[k] = B[i]
                i = i + 1
        elseif B[j] < B[i]
                A[k] = B[j]
                j = j + 1
        else
                A[k] = B[i]
                i = i + 1
```

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

AALBORG UNIVERSITET

Merge sort

Merge-operationen

```
Merge(A, low, mid, high)
let B[0 : high − low] be
        a new array with 0-index
for i = 0 to B.length
        B[i] = A[i + low]

i = 0, j = (high − mid)
for k = low to high
        if (i + low) > mid
                A[k] = B[j]
                j = j + 1
        elseif (j + mid) > high
                A[k] = B[i]
                i = i + 1
        elseif B[j] < B[i]
                A[k] = B[j]
                j = j + 1
        else
                A[k] = B[i]
                i = i + 1
```

- Vi lader $low = p, mid = q$ og $high = r$
- 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, low, mid, high)
  let B[0 : high − low] be
        a new array with 0-index
  for i = 0 to B.length
        B[i] = A[i + low]

  i = 0, j = (high − mid)
  for k = low to high
        if (i + low) > mid
              A[k] = B[j]
              j = j + 1
        elseif (j + mid) > high
              A[k] = B[i]
              i = i + 1
        elseif B[j] < B[i]
              A[k] = B[j]
              j = j + 1
        else
              A[k] = B[i]
              i = i + 1
```

- Vi lader $low = p$, $mid = q$ og $high = r$
- 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, low, mid, high)
    let B[0 : high − low] be
            a new array with 0-index
    for i = 0 to B.length
            B[i] = A[i + low]

    i = 0, j = (high − mid)
    for k = low to high
            if (i + low) > mid
                    A[k] = B[j]
                    j = j + 1
            elseif (j + mid) > high
                    A[k] = B[i]
                    i = i + 1
            elseif B[j] < B[i]
                    A[k] = B[j]
                    j = j + 1
            else
                    A[k] = B[i]
                    i = i + 1
```

- Vi lader $low = p, mid = q$ og $high = r$
- 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, low, mid, high)
  let B[0 : high − low] be
        a new array with 0-index
  for i = 0 to B.length
        B[i] = A[i + low]

  i = 0, j = (high − mid)
  for k = low to high
        if (i + low) > mid
              A[k] = B[j]
              j = j + 1
        elseif (j + mid) > high
              A[k] = B[i]
              i = i + 1
        elseif B[j] < B[i]
              A[k] = B[j]
              j = j + 1
        else
              A[k] = B[i]
              i = i + 1
```

- Vi lader $low = p, mid = q$ og $high = r$
- 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

AALBORG UNIVERSITET

Merge sort

Merge-operationen

```
Merge(A, low, mid, high)
let B[0 : high − low] be
        a new array with 0-index
for i = 0 to B.length
        B[i] = A[i + low]

i = 0, j = (high − mid)
for k = low to high
        if (i + low) > mid
                A[k] = B[j]
                j = j + 1
        elseif (j + mid) > high
                A[k] = B[i]
                i = i + 1
        elseif B[j] < B[i]
                A[k] = B[j]
                j = j + 1
        else
                A[k] = B[i]
                i = i + 1
```

- Vi lader $low = p$, $mid = q$ og $high = r$
- 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, low, mid, high)
let B[0 : high − low] be
        a new array with 0-index
for i = 0 to B.length
        B[i] = A[i + low]

i = 0, j = (high − mid)
for k = low to high
        if (i + low) > mid
                A[k] = B[j]
                j = j + 1
        elseif (j + mid) > high
                A[k] = B[i]
                i = i + 1
        elseif B[j] < B[i]
                A[k] = B[j]
                j = j + 1
        else
                A[k] = B[i]
                i = i + 1
```

- Vi lader $low = p, mid = q$ og $high = r$
- 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, low, mid, high)
let B[0 : high − low] be
        a new array with 0-index
for i = 0 to B.length
        B[i] = A[i + low]

i = 0, j = (high − mid)
for k = low to high
        if (i + low) > mid
                A[k] = B[j]
                j = j + 1
        elseif (j + mid) > high
                A[k] = B[i]
                i = i + 1
        elseif B[j] < B[i]
                A[k] = B[j]
                j = j + 1
        else
                A[k] = B[i]
                i = i + 1
```

- Vi lader $low = p, mid = q$ og $high = r$
- 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 $L[0 : n_L - 1]$ og $R[0 : n_R - 1]$ i et enkelt array B

Merge sort

Merge-operationen

```
Merge(A, low, mid, high)
  let B[0 : high − low] be
        a new array with 0-index
  for i = 0 to B.length
        B[i] = A[i + low]

  i = 0, j = (high − mid)
  for k = low to high
        if (i + low) > mid
              A[k] = B[j]
              j = j + 1
        elseif (j + mid) > high
              A[k] = B[i]
              i = i + 1
        elseif B[j] < B[i]
              A[k] = B[j]
              j = j + 1
        else
              A[k] = B[i]
              i = i + 1
```

- Vi lader $low = p, mid = q$ og $high = r$
- 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 $L[0 : n_L - 1]$ og $R[0 : n_R - 1]$ i et enkelt array B
- ...og at vi klarer resten i et enkelt loop (fremfor 3, eew!)

AALBORG UNIVERSITET

Merge sort

Example

Outline

Exercises!

Yay!

Outline

Outline

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!