

# Øvingsforelesning 3: Splitt og hersk

Daniel Solberg

# Plan for dagen

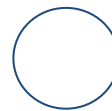
- Vi går raskt gjennom øving 2
- Splitt og hersk
- Algoritmer:
  - Mergesort
  - Quicksort
  - Binærsøk
- Rekurrenser, masse rekurrenser

# Splitt og hersk (Divide-and-Conquer)

- Et designparadigme for algoritmer
- Hovedidé: Løs et problem vha. å løse mindre versjoner av det samme problemet
- Går hånd i hånd med matematisk induksjon

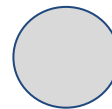
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine



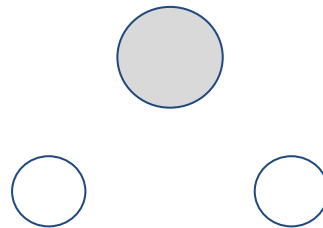
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine



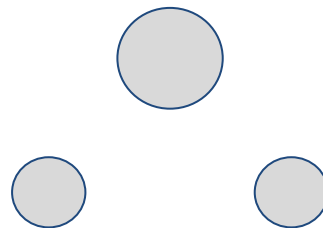
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine



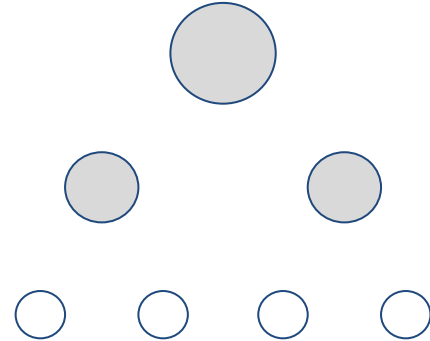
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine



# Splitt og hersk (Divide-and-Conquer)

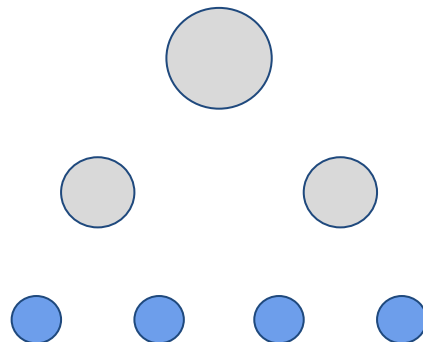
- Tre steg:
  - Divide
  - Conquer
  - Combine





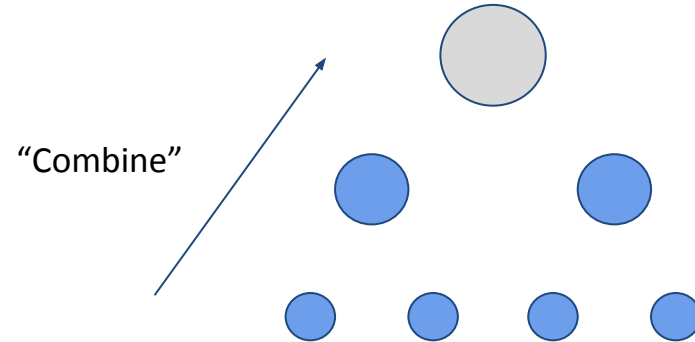
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine



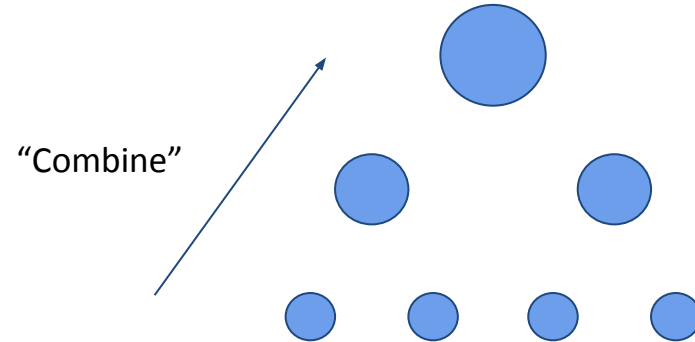
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine



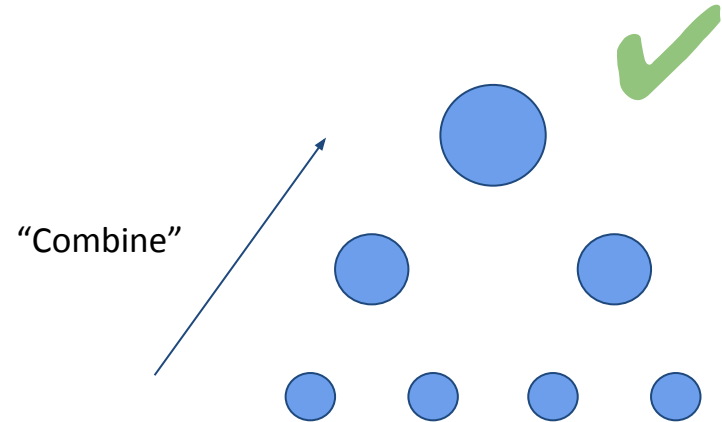
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine



# Splitt og hersk (Divide-and-Conquer)

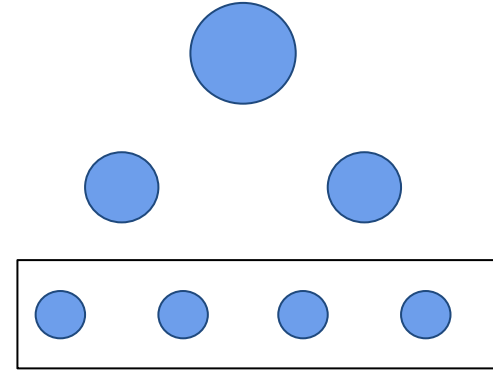
- Tre steg:
  - Divide
  - Conquer
  - Combine



# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine

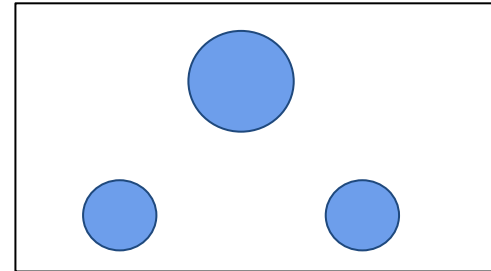
Basetilfeller



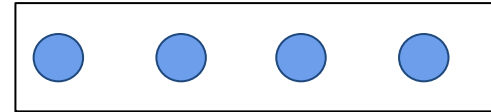
# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide
  - Conquer
  - Combine

Rekursive  
tilfeller



Basetilfeller



# Splitt og hersk (Divide-and-Conquer)

- Tre steg:
  - Divide: Dele opp i “subproblemer”
  - Conquer: Løse subproblemene
  - Combine: Kombinere løsningene til en en større løsning

# Algoritme: Binærsøk

- Står ikke i boka, men i pensumheftet!
- Et veldig enkelt eksempel på splitt og hersk!
  - Finner løsningen ved å løse ett mindre subproblem
  - Ingen combine-steg
- Kort fortalt: Lar oss søke i en **sortert** tabell etter et gitt element på  $O(\log n)$  tid



# Algoritme: Binærsøk

- BISECT( $A$ ,  $p$ ,  $r$ ,  $v$ )

$A = [1, 2, 3, 4, 5, 6, 7, 8]$

# Algoritme: Binærsøk

- `BISECT(A, 1, 8, 7)`

$A = [1, 2, 3, 4, 5, 6, 7, 8]$

# Algoritme: Binærsøk

- `BISECT(A, 1, 8, 7)`

$A = [1, 2, 3, 4, 5, 6, 7, 8]$

# Algoritme: Binærsøk

- BISECT(A, 5, 8, 7)

$A = [1, 2, 3, 4, 5, 6, 7, 8]$

# Algoritme: Binærsøk

- BISECT(A, 5, 8, 7)

$A = [1, 2, 3, 4, 5, 6, 7, 8]$

# Algoritme: Binærsøk

- BISECT(A, 7, 8, 7)

A = [1, 2, 3, 4, 5, 6, 7, 8]

# Algoritme: Binærsøk

- BISECT(A, 7, 8, 7)

A = [1, 2, 3, 4, 5, 6, **7**, 8]

# Algoritme: Binærsøk

- BISECT(A, 7, 8, 7)

A = [1, 2, 3, 4, 5, 6, **7**, 8]

Returner i = 7



# Algoritme: Mergesort

- En ganske god sorteringsalgoritme
  - Kjøretid  $\Theta(n \log n)$
  - Mye bedre enn insertionsort, bubblesort og andre  $O(n^2)$ -algoritmer for store input

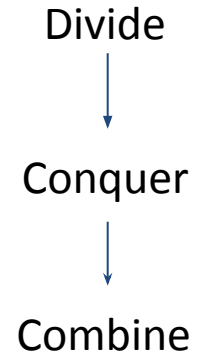
# Algoritme: Mergesort

- Mergesort:
  1. Del tabellen i to
  2. Sorter hver halvdel rekursivt med Mergesort
  3. Flett sammen halvdelene til én sortert liste

# Algoritme: Mergesort

- Mergesort:

1. Del tabellen i to
2. Sorter hver halvdel rekursivt med Mergesort
3. Flett sammen halvdelene til én sortert liste



# Algoritme: Mergesort

MERGE-SORT( $A, p, r$ )

```
1  if  $p < r$   
2       $q = \lfloor (p + r) / 2 \rfloor$   
3      MERGE-SORT( $A, p, q$ )  
4      MERGE-SORT( $A, q + 1, r$ )  
5      MERGE( $A, p, q, r$ )
```

# Algoritme: Mergesort

MERGE-SORT( $A, p, r$ )

```
1  if  $p < r$ 
2       $q = \lfloor (p + r) / 2 \rfloor$ 
3      MERGE-SORT( $A, p, q$ )
4      MERGE-SORT( $A, q + 1, r$ )
5      MERGE( $A, p, q, r$ )
```

**Idé:** Å flette sammen to sorterte lister er ganske enkelt,  $\Theta(n)$  tid

# Algoritme: Mergesort

- Tavle-eksempel

Sorter med Mergesort listen

$A = [8, 3, 6, 2, 5, 7, 4, 1]$

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([8, 3, 6, 2])



# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([8, 3, 6, 2])

Mergesort([8, 3])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([8, 3, 6, 2])

Mergesort([8, 3])

Mergesort([8])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([8, 3, 6, 2])

Mergesort([8, 3])

Mergesort([3])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([8, 3, 6, 2])

Mergesort([8, 3])

- Merge([8], [3])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([8, 3, 6, 2])

Mergesort([3, 8])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 6, 2])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 6, 2])

Mergesort([6, 2])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 6, 2])

Mergesort([6, 2])

Mergesort([6])



# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 6, 2])

Mergesort([6, 2])

Mergesort([2])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 6, 2])

Mergesort([6, 2])

-Merge([6], [2])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 6, 2])

Mergesort([2, 6])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 2, 6])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([3, 8, 2, 6])

-Merge([3, 8], [2, 6])

# Algoritme: Mergesort

Mergesort([8, 3, 6, 2, 5, 7, 4, 1])

Mergesort([2, 3, 6, 8])

# Algoritme: Mergesort

Mergesort([2, 3, 6, 8, 5, 7, 4, 1])

# Algoritme: Mergesort

Mergesort([2, 3, 6, 8, 5, 7, 4, 1])

■ ■ ■



# Algoritme: Mergesort

Mergesort([2, 3, 6, 8, 1, 4, 5, 7])

# Algoritme: Mergesort

Mergesort([2, 3, 6, 8, 1, 4, 5, 7])

-Merge([2, 3, 6, 8], [1, 4, 5, 7])

# Algoritme: Mergesort

Mergesort([1, 2, 3, 4, 5, 6, 7, 8])

# Algoritme: Mergesort

[1, 2, 3, 4, 5, 6, 7, 8]

# Algoritme: Quicksort

- *Også* en ganske god sorteringsalgoritme :)
  - Brukes i praksis, kan være raskere enn mergesort i virkeligheten
  - Average case kjøretid  $\Theta(n \log n)$ , men worst case  $O(n^2)$

# Algoritme: Quicksort

Quicksort:

1. Velg en *pivot* (splittelement). Velger alltid bakerste element.
2. Partisjoner tabellen:  $\leq$ -side og  $>$ -side
3. Sorter hver del med Quicksort

# Algoritme: Quicksort

QUICKSORT( $A, p, r$ )

1   **if**  $p < r$

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

3       QUICKSORT( $A, p, q - 1$ )

4       QUICKSORT( $A, q + 1, r$ )

# Algoritme: Quicksort

QUICKSORT( $A, p, r$ )

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

**Idé:** Å partisjonere en liste er ganske enkelt,  $\Theta(n)$  tid



# Algoritme: Quicksort

- Sortere med Quicksort:

$A = [8, 3, 6, 2, 1, 7, 4, 5]$

# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement

$A = [8, 3, 6, 2, 1, 7, 4, 5]$

# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement
  - 2) Del i mindre enn og større enn

$A = [8, 3, 6, 2, 1, 7, 4, 5]$

# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement
  - 2) Del i mindre enn og større enn

$A = [8, 3, 6, 2, 1, 7, 4, 5]$

# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement
  - 2) Del i mindre enn og større enn

$A = [8, 3, 6, 2, 1, 7, 4, 5]$

# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement
  - 2) Del i mindre enn og større enn

$A = [3, 2, 1, 4, 5, 8, 6, 7]$

# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement
  - 2) Del i mindre enn og større enn


$A = [3, 2, 1, 4, 5, 8, 6, 7]$

The diagram illustrates the partitioning step of the Quicksort algorithm. The array  $A = [3, 2, 1, 4, 5, 8, 6, 7]$  is shown. The pivot element is 5, which is highlighted in blue. The elements less than the pivot (3, 2, 1, 4) are grouped together under a bracket, and the elements greater than the pivot (8, 6, 7) are grouped together under another bracket.

# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement
  - 2) Del i mindre enn og større enn
  - 3) Sorter de to delene med Quicksort

$A = [3, 2, 1, 4, 5, 8, 6, 7]$



Quicksort                      Quicksort



# Algoritme: Quicksort

- Sortere med Quicksort:
  - 1) Velg splittelement
  - 2) Del i mindre enn og større enn
  - 3) Sorter de to delene med Quicksort

A = [1, 2, 3, 4, 5, 6, 7, 8]

Quicksort Quicksort

# Algoritme: Quicksort

- Ulempe: Noen ganger får vi worst case,  $O(n^2)$ 
  - Og dette skjer for en allerede sortert liste :-/

# Algoritme: Quicksort

- Ulempe: Noen ganger får vi worst case,  $O(n^2)$ 
  - Og dette skjer for en allerede sortert liste :-/

$A = [1, 2, 3, 4, 5, 6, 7, 8]$

# Algoritme: Quicksort

- Ulempe: Noen ganger får vi worst case,  $O(n^2)$ 
  - Og dette skjer for en allerede sortert liste :-/

$A = [1, 2, 3, 4, 5, 6, 7, 8]$



Quicksort

# Algoritme: Quicksort

- Ulempe: Noen ganger får vi worst case,  $O(n^2)$ 
  - Og dette skjer for en allerede sortert liste :-/
- Derfor: Randomized-Quicksort
  - Velger tilfeldig splittelement, i stedet for alltid bakerst

# Algoritme: Quicksort

- Ulempe: Noen ganger får vi worst case,  $O(n^2)$ 
  - Og dette skjer for en allerede sortert liste :-/
- Derfor: Randomized-Quicksort
  - Velger tilfeldig splittelement, i stedet for alltid bakerst

$A = [8, 3, 6, 2, 1, 7, 4, 5]$

# Algoritme: Quicksort

- Ulempe: Noen ganger får vi worst case,  $O(n^2)$ 
  - Og dette skjer for en allerede sortert liste :-/
- Derfor: Randomized-Quicksort
  - Velger tilfeldig splittelement, i stedet for alltid bakerst

$A = [8, 3, 6, 2, 1, 7, 4, 5]$

# Quicksort vs Randomized-Quicksort

- Spørsmål: Dersom input består av en tabell med helt tilfeldige tall, hvilken algoritme er best? Er det noen forskjell?



# Plan for dagen

- Vi går raskt gjennom øving 2
- Splitt og hersk
- Algoritmer:
  - Mergesort
  - Quicksort
  - Binærsøk
- Rekurrenser, masse rekurrenser

# Rekurrenser: Motivasjon

- Hittil har algoritmene sett noe sånn ut:

```
1 ▼ function sum(A)
2     s = 0
3 ▼   for i in 1:length(A)
4       s += A[i]
5   end
6   return s
7 end
```

# Rekurrenser: Motivasjon

- Hittil har algoritmene sett noe sånn ut:
  - Ganske enkelt å finne kjøretid
  - Gjentar  $n$  ganger  $\rightarrow \Theta(n)$  kjøretid

```
1 ▼ function sum(A)
2     s = 0
3 ▼   for i in 1:length(A)
4       s += A[i]
5   end
6   return s
7 end
```

# Rekurrenser: Motivasjon

- Men hva når de ser sånn ut?

```
function sum(A)
    if length(A) > 1
        return A[1] + sum(A[2:end])
    else
        return 0
    end
end
```

# Rekurrenser: Motivasjon

- Men hva når de ser sånn ut?
  - Trenger en måte å uttrykke og regne på kjøretiden

```
function sum(A)
    if length(A) > 1
        return A[1] + sum(A[2:end])
    else
        return 0
    end
end
```

# Rekurrenser: Motivasjon

- La  $T(n)$  betegne kjøretiden for `sum(A)` for en tabell med  $n$  tall

```
function sum(A)
    if length(A) > 1
        return A[1] + sum(A[2:end])
    else
        return 0
    end
end
```

# Rekurrenser: Motivasjon

- La  $T(n)$  betegne kjøretiden for `sum(A)` for en tabell med  $n$  tall
  - $T(n) = T(n-1) + 1$

```
function sum(A)
    if length(A) > 1
        return A[1] + sum(A[2:end])
    else
        return 0
    end
end
```

```
end
```

# Rekurrenser: Motivasjon

- La  $T(n)$  betegne kjøretiden for  $\text{sum}(A)$  for en tabell med  $n$  tall
  - $T(n) = T(n-1) + 1 \rightarrow T(n) = \Theta(n)$

```
function sum(A)
    if length(A) > 1
        return A[1] + sum(A[2:end])
    else
        return 0
    end
end
```



# Løse rekurrenser

- Vi bruker følgende metoder i dette faget
  - Master-teoremet
  - Regning med rekursjonstrær
  - Iterasjonsmetoden
  - Substitusjonsmetoden
- Og et triks vi må kunne
  - Variabelskifte

# Løse rekurrenser

- Vi bruker følgende metoder i dette faget
  - Master-teoremet
    - Gir en eksakt løsning
  - Regning med rekursjonstrær
    - Gir en god gjetning
  - Iterasjonsmetoden
    - Gir en god gjetning
  - Substitusjonsmetoden
    - Beviser en gjetning
- Og et triks vi må kunne
  - Variabelskifte
    - Gjør noen former for vanskelige rekurrenser om til enkle rekurrenser

# Løse rekurrenser

- Vi regner på tavlen