Midtsemesterforelesning

Algdat 2020

Datastrukturer :3

HVa gjør en datastruktur?

- Representere data

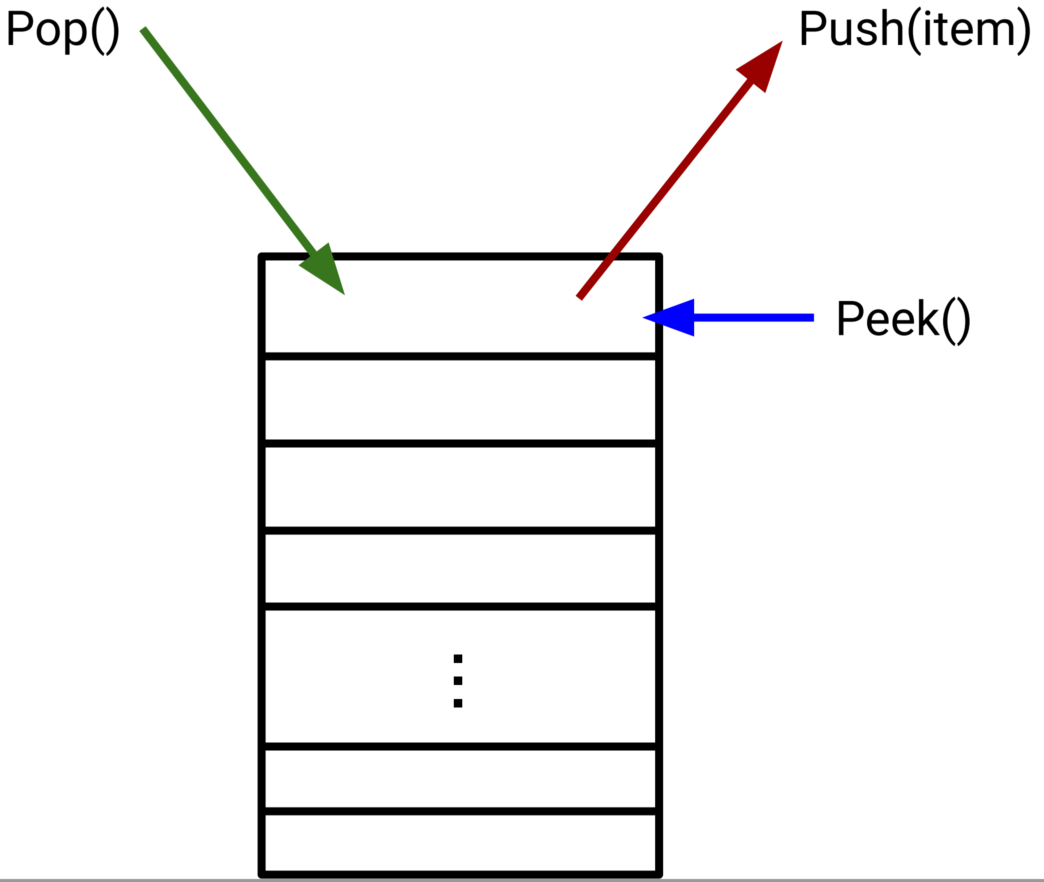
HVORDAN!?

I feks:

- Prioritetskøer, Trær, Lenkede lister, hash-maps… osv.

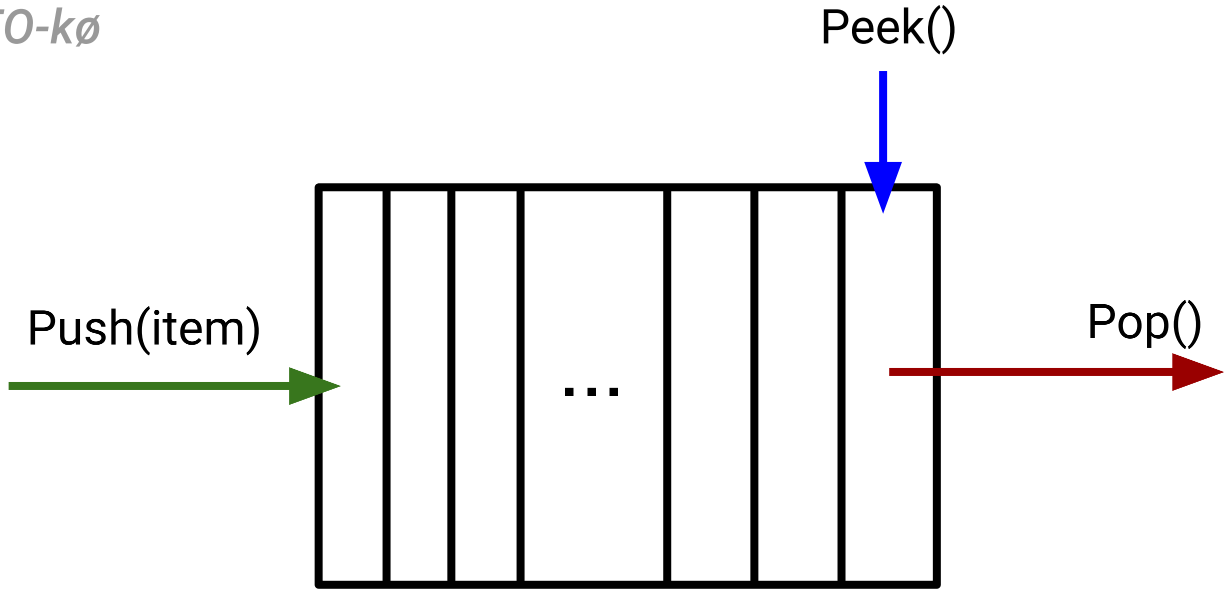
Stack

eller LIFO-kø

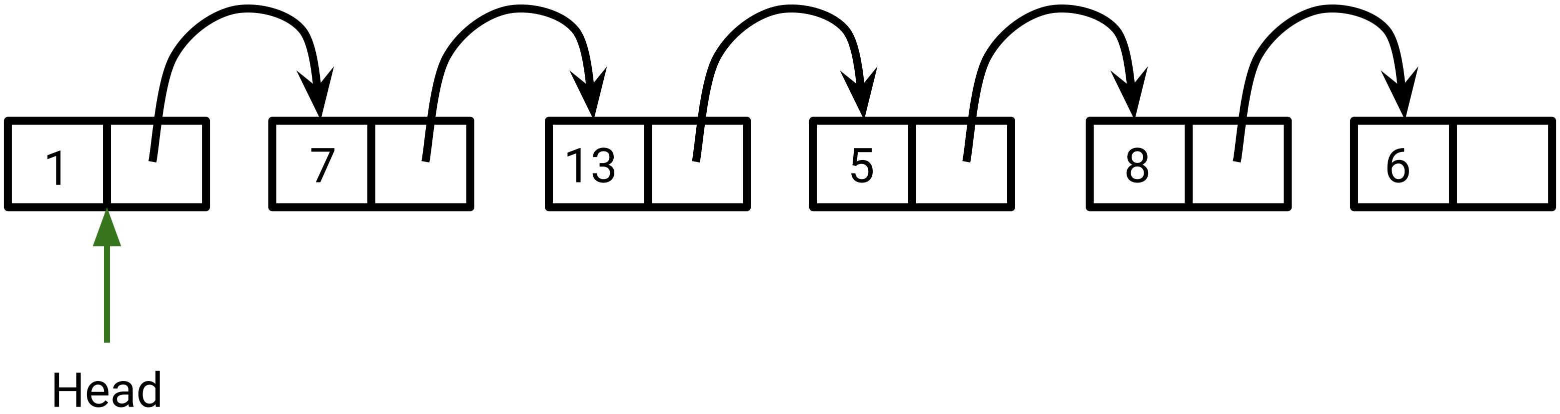


Queue

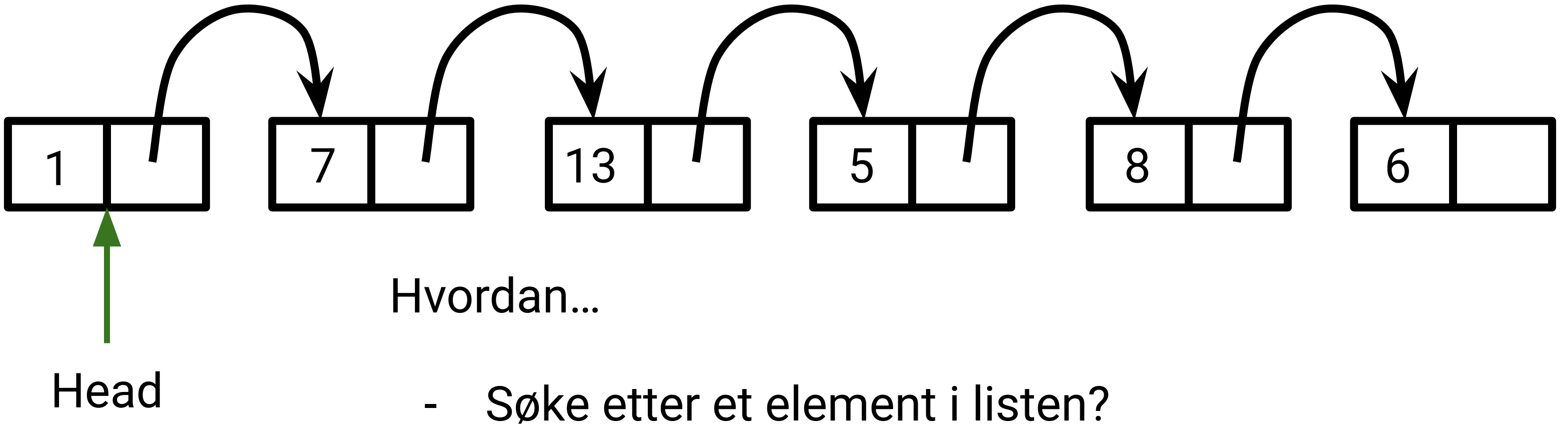
eller FIFO-kø



Linked list



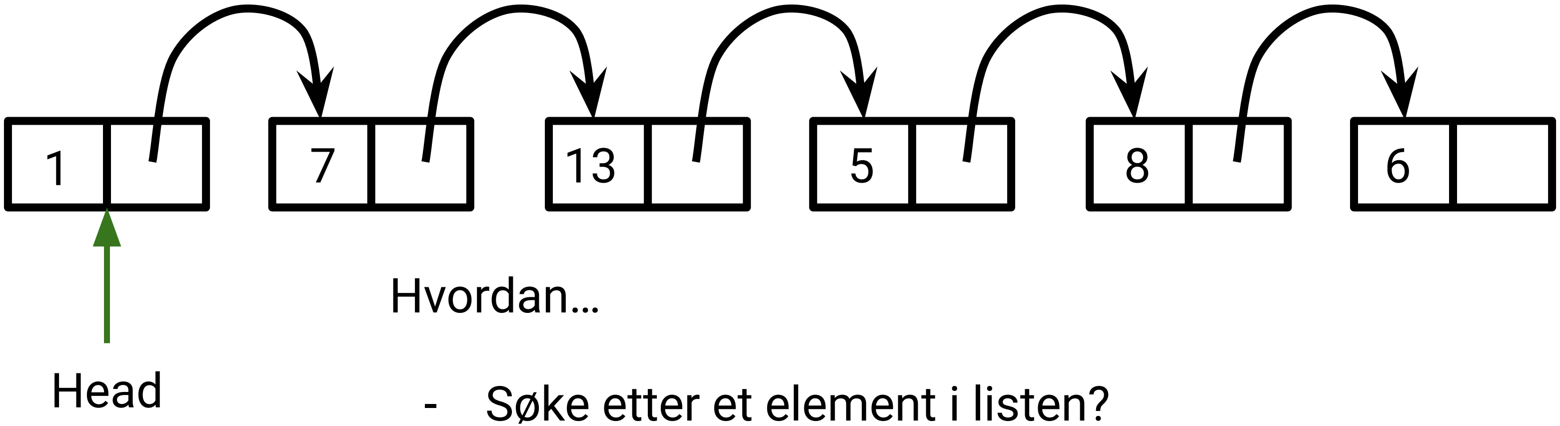
Linked list



Hvordan...

- Søke etter et element i listen?

Linked list

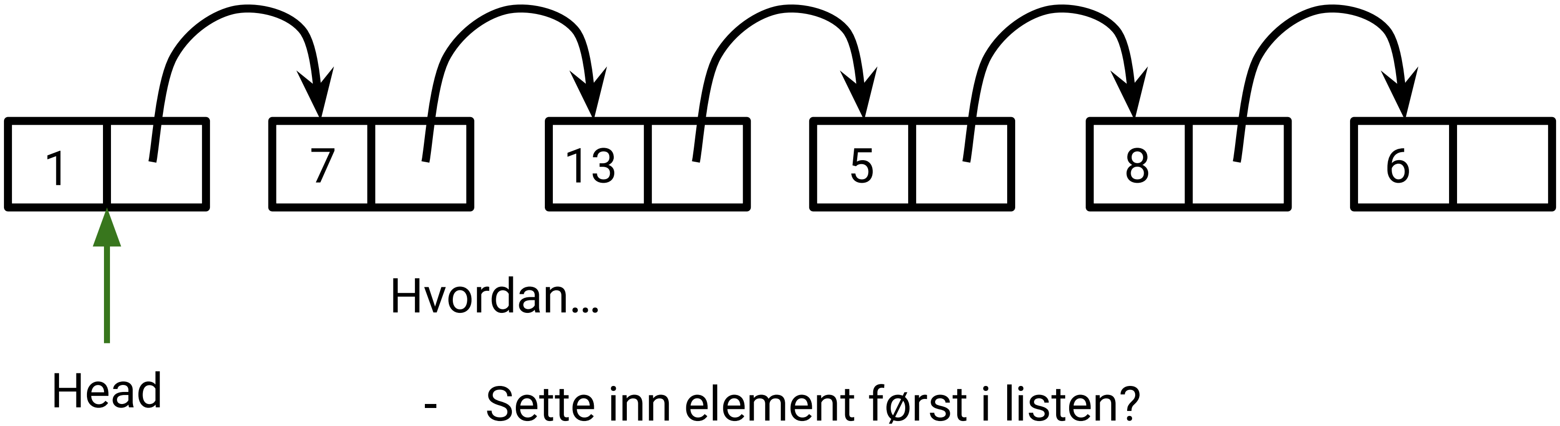


Hvordan…

- Søke etter et element i listen?

Begynne fra starten → lineært søk
Kjøretid: O(n)

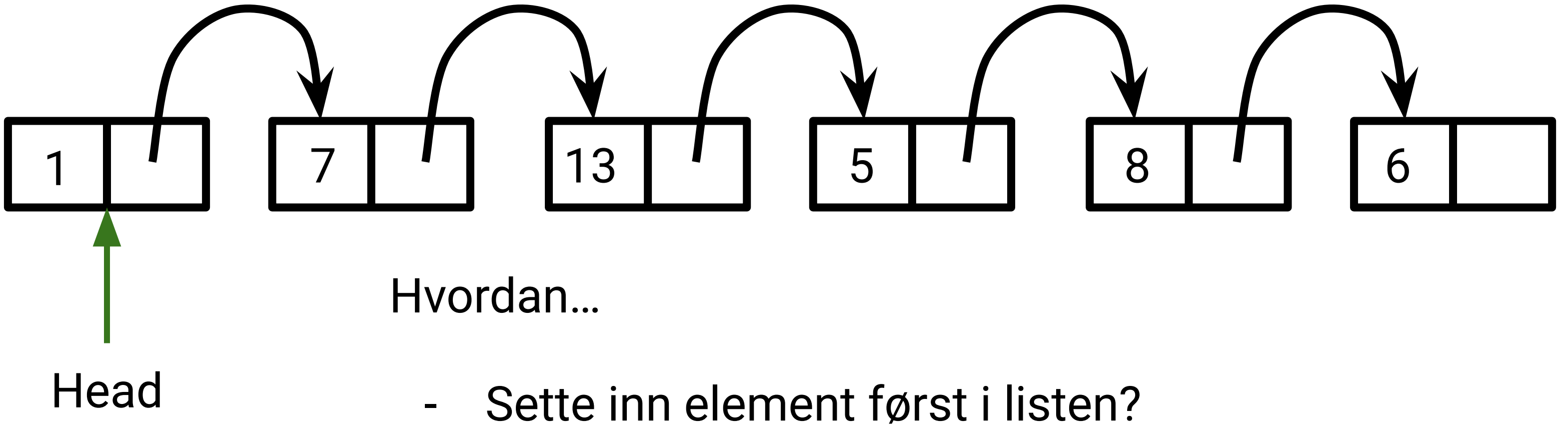
Linked list



Hvordan…

- Sette inn element først i listen?

Linked list

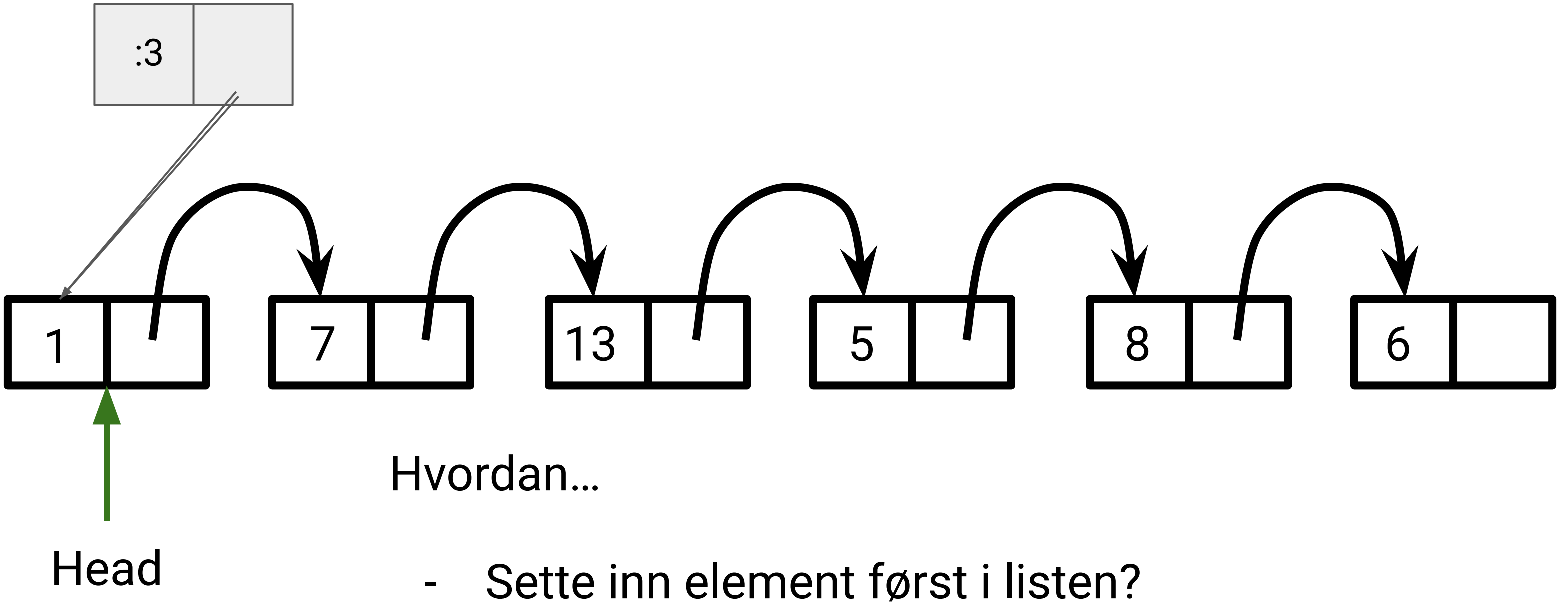


Hvordan…

- Sette inn element først i listen?

Sett inn element → Sett peker til neste element i listen
Kjøretid: O(1)

Linked list

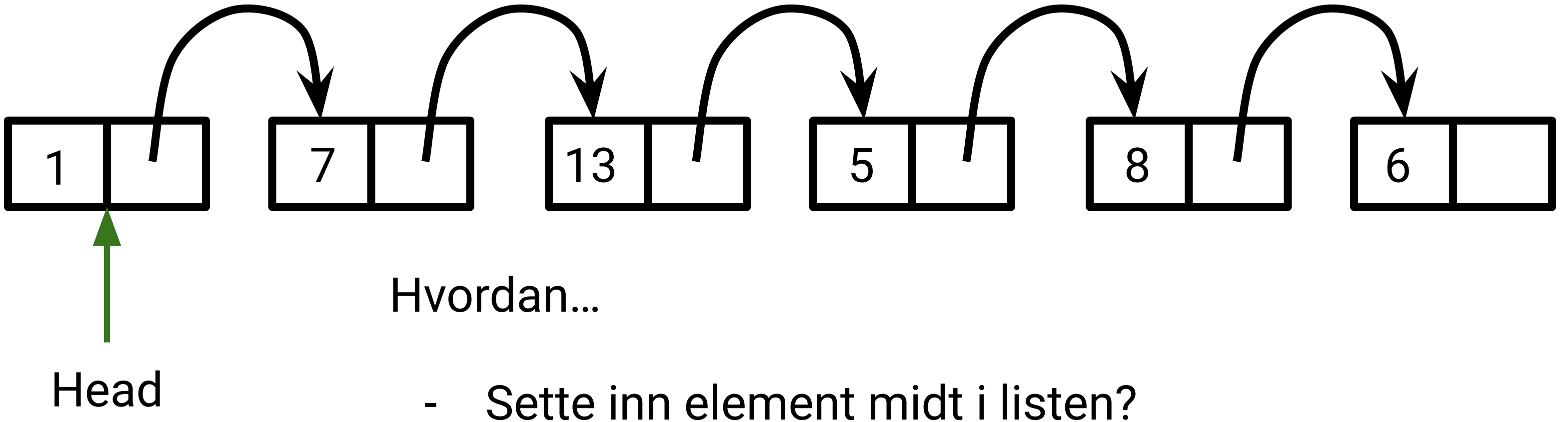


Hvordan…

- Sette inn element først i listen?

Sett inn element → Sett peker til neste element i listen
Kjøretid: O(1)

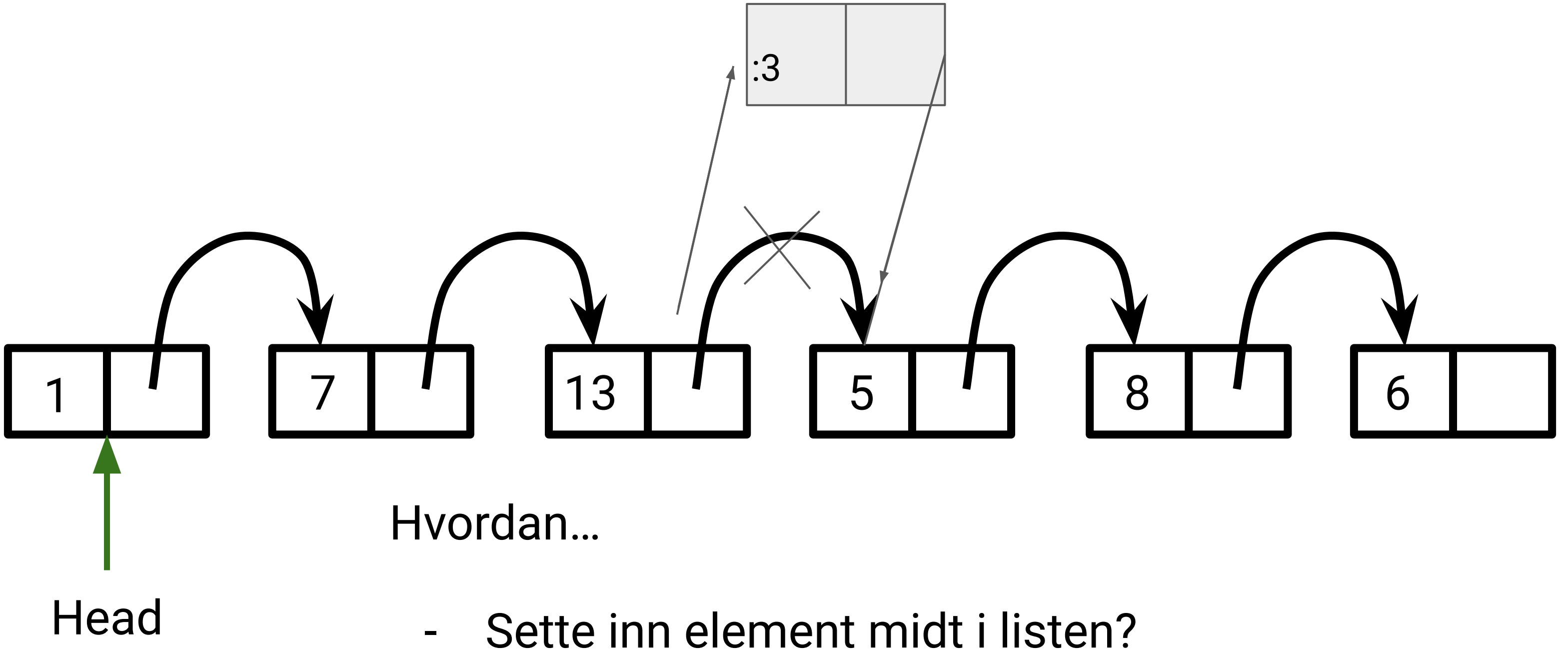
Linked list



Hvordan…

- Sette inn element midt i listen?

Linked list

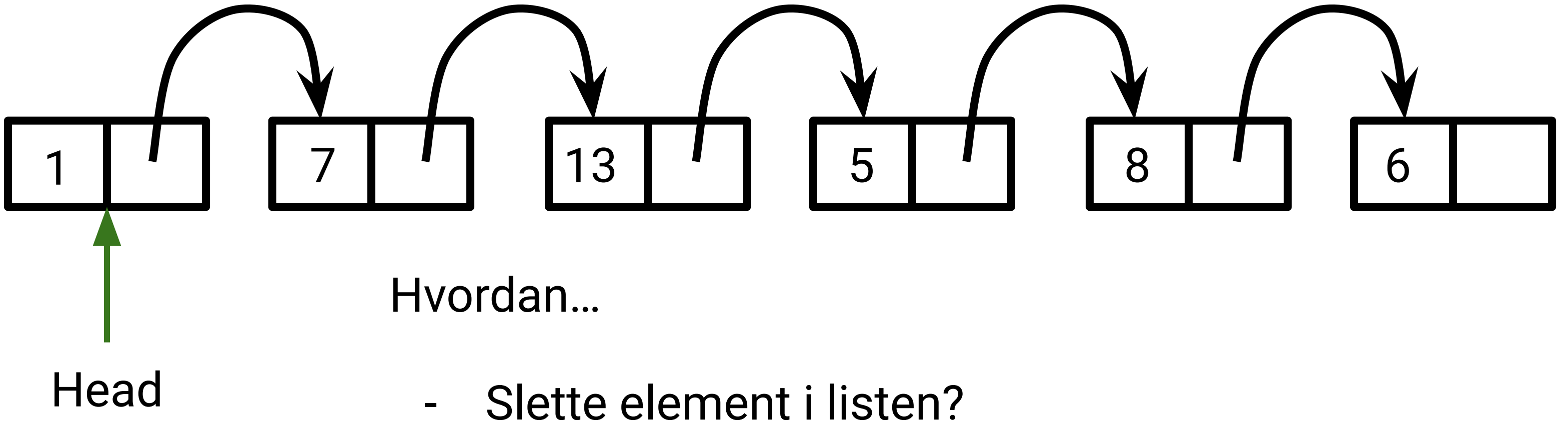


Hvordan…

- Sette inn element midt i listen?

Søke seg frem → oppdatere lenker → sette inn element
Kjøretid: O(n)

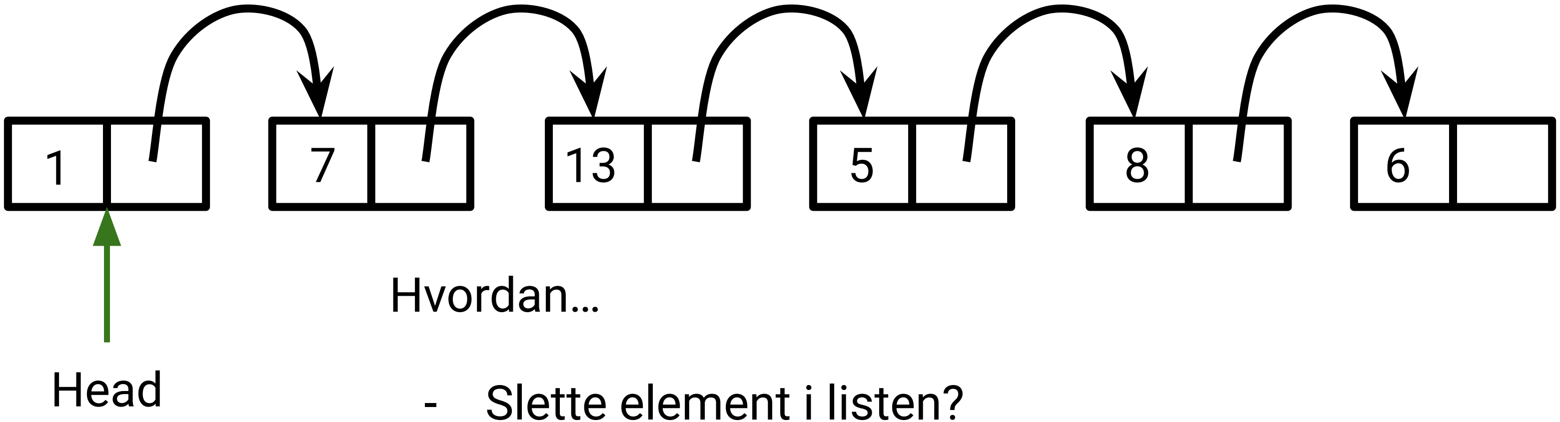
Linked list



Hvordan...

- Slette element i listen?

Linked list

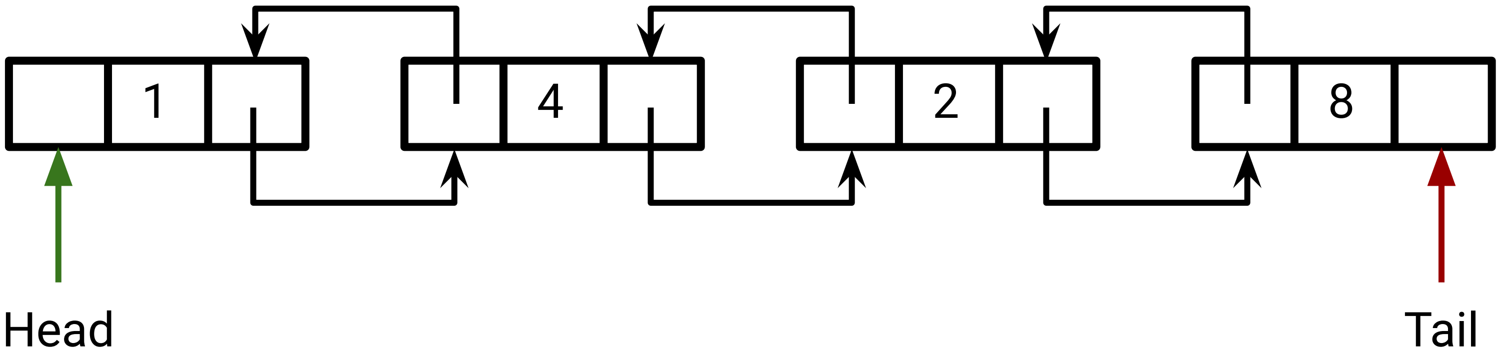


Hvordan…

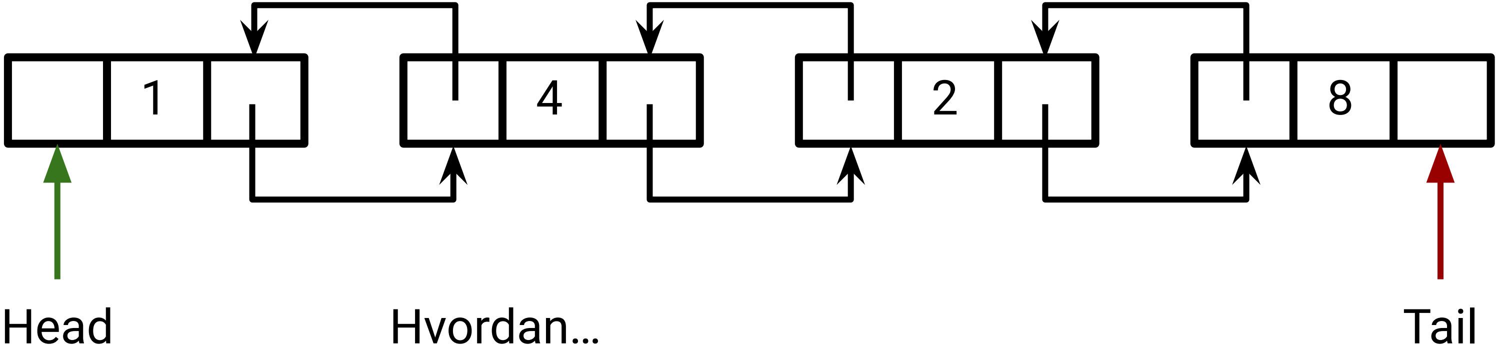
- Slette element i listen?

Søke seg frem → oppdatere lenke ved å hoppe over et element
Kjøretid: O(n)

Doubly linked list



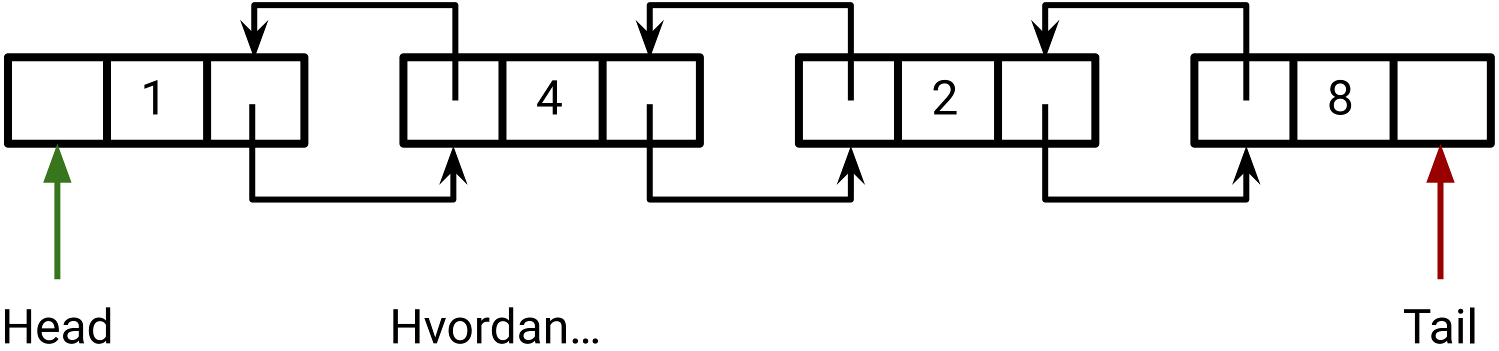
Doubly linked list



Hvordan...

- Slette element i listen?

Doubly linked list

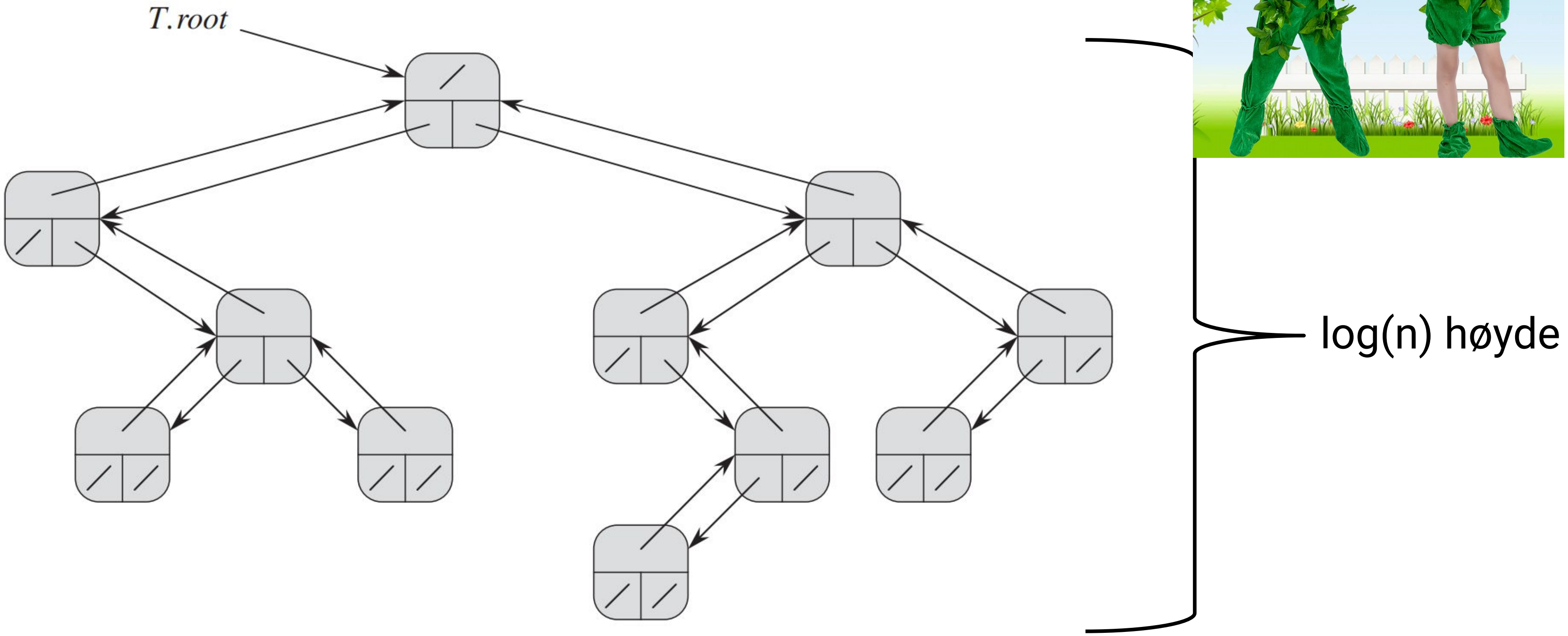


Hvordan…

- Slette element i listen?

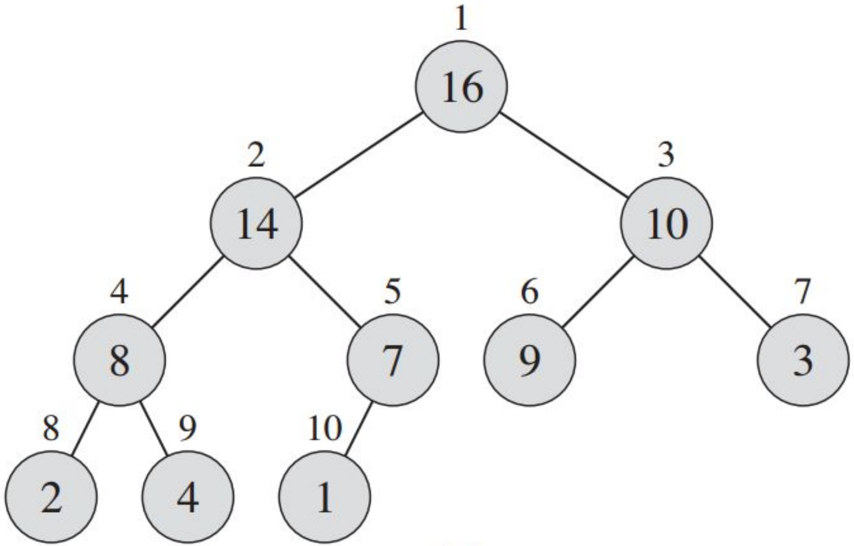
Oppdatere naboene ved å lenke dem til hverandre
Kjøretid: O(1)

Trær



https://www.youtube.com/watch?v=t0Cq6tVNRBA&t=365s

Heap



(a)

Insert = O(log n), O(h)

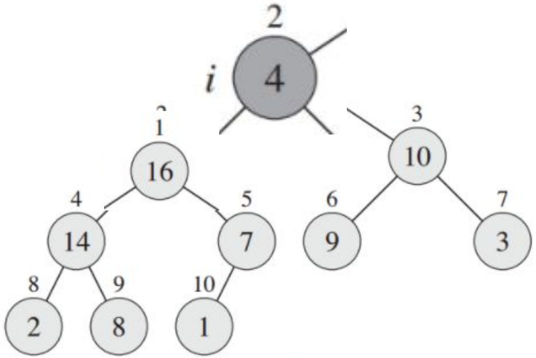
Delete = O(log n), O(h)

Build = O(n)

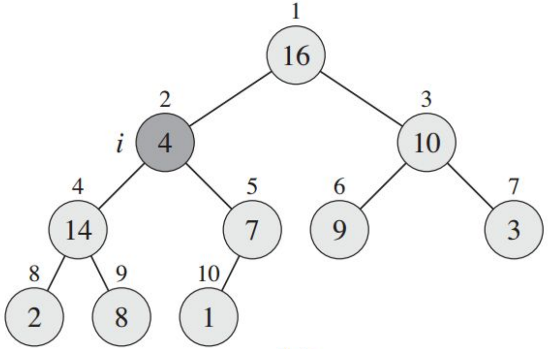
MAX-HEAPIFY

MAX-HEAPIFY = O(log

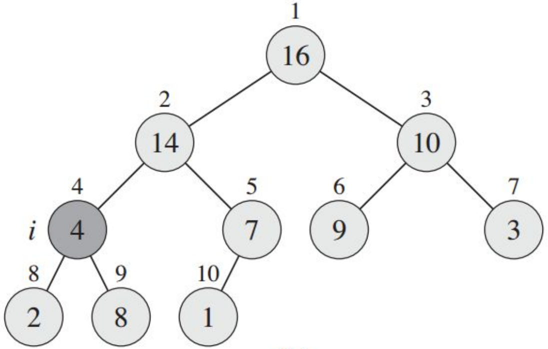
HEAP-EXTRACT-MAX = O(log



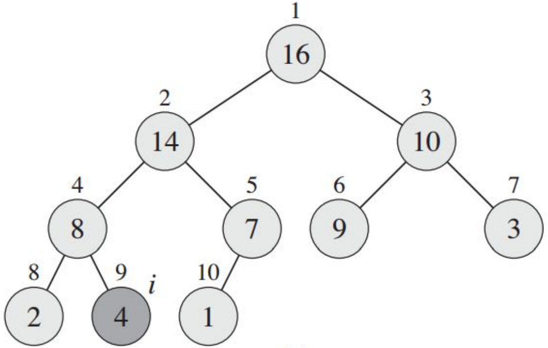
(a)



(a)



(b)

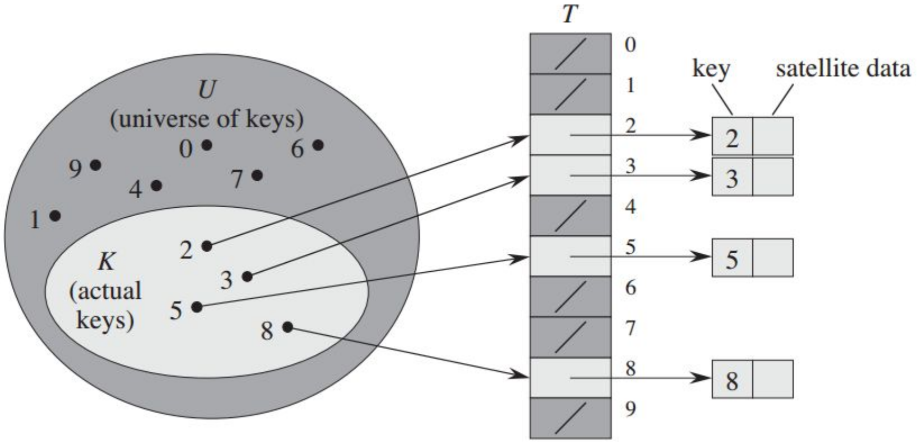


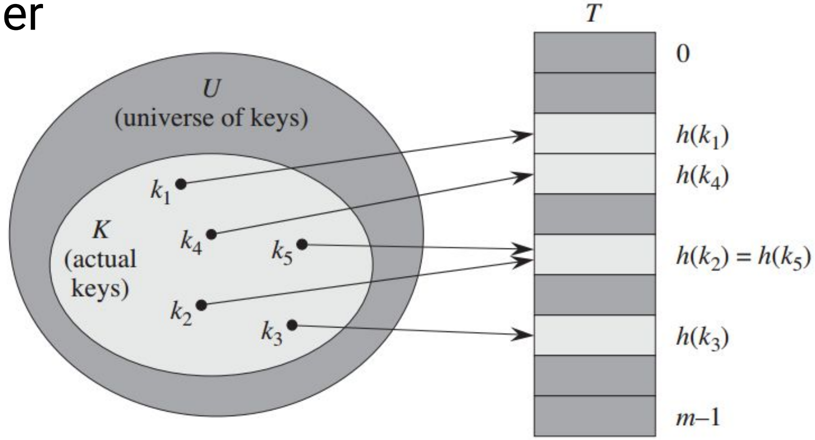
(c)

Hashing

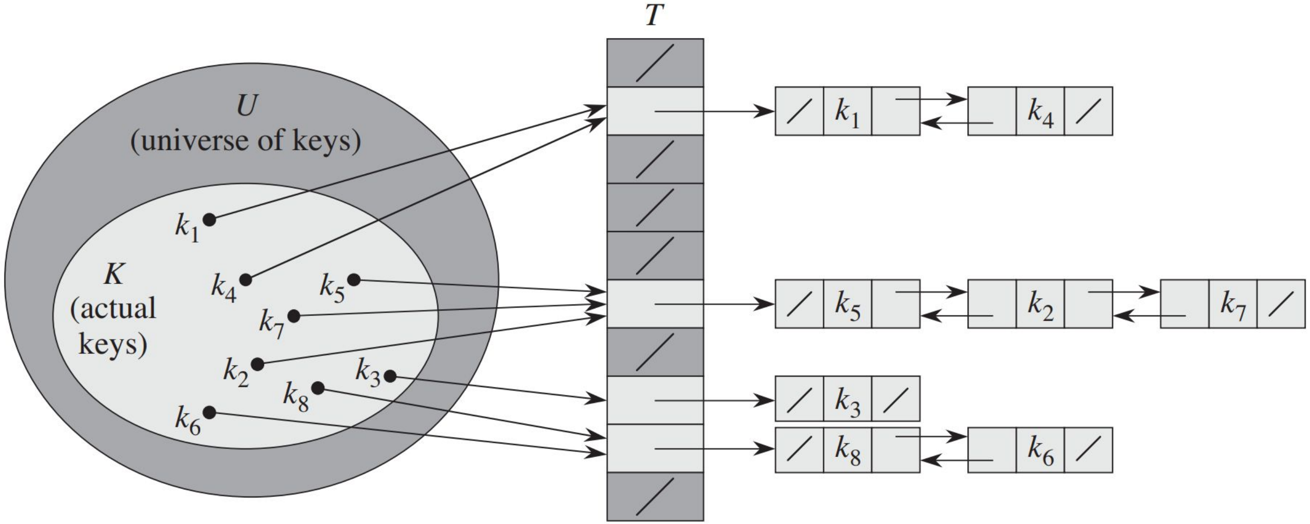
h(k) = index

- Mapping fra nøkler til indekser
- Må være deterministisk
- **Bør** fordele jevnt





Hash chaining



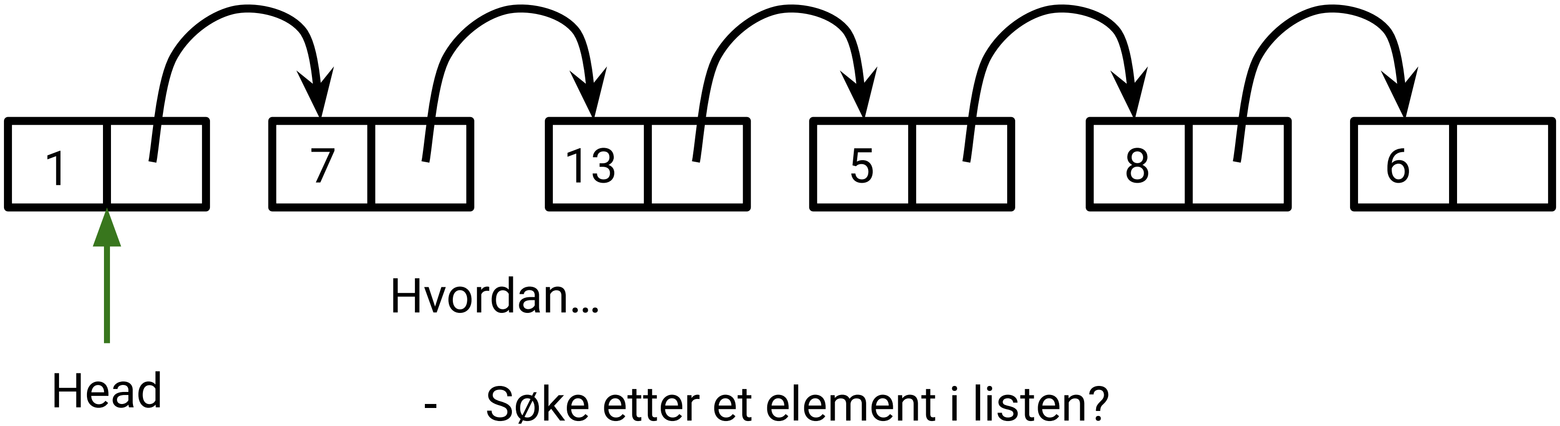
Hva er greia? Hvorfor vil vi ha datastrukturer?

Hva er greia? Hvorfor vil vi ha datastrukturer?

Fordi forskjellige Datastrukturer har forskjellige egenskaper vi kan benytte oss av!

Du har et lager med godteri og du vil ha et program som gjør at du raskt kan sjekke dataen om enkelte typer godteri. Hvilken datastruktur kan du trenge her?

Linked list



Hvordan…

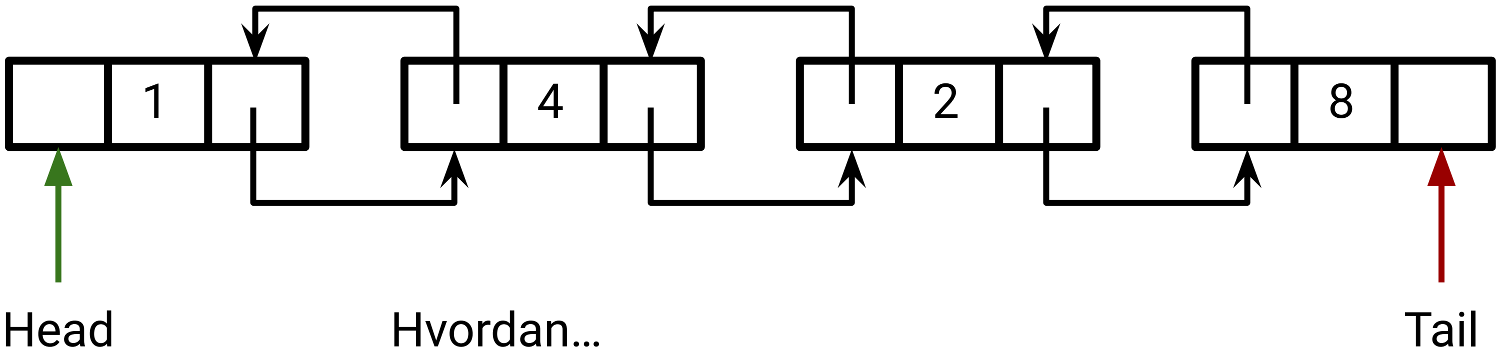
- Søke etter et element i listen?

Begynne fra starten → lineært søk
Kjøretid: O(n)

https://www.youtube.com/watch?v=pKO9UjSeLew

Eller hva om du vil raskt kunne sjekke hvilket godteri som veier mest men også sette inn nye typer godteri raskt?

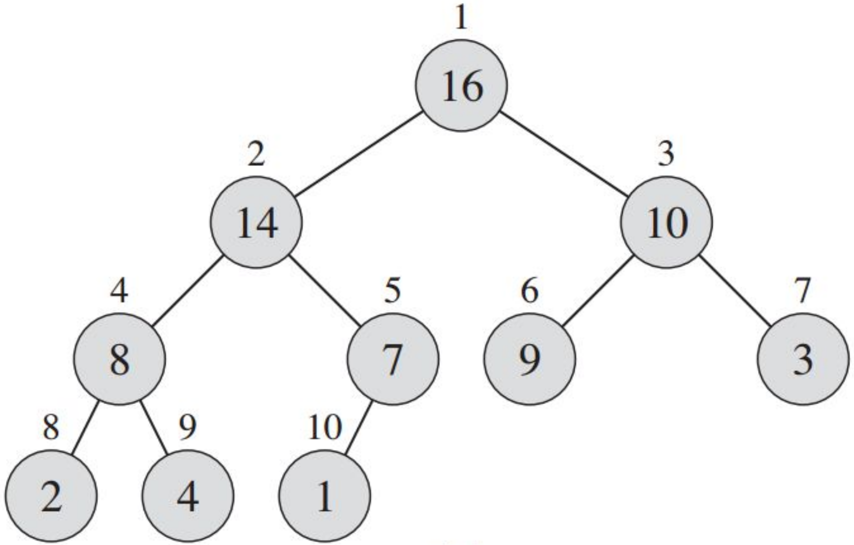
Doubly linked list



Hvordan…

- Slette element i listen?

Eller hva om du vil raskt kunne sjekke hvilket godteri som veier mest? KONSTANT TID!



(a)

Tips:
Algdat er både et puggefag og et forståelsesfag

- Lær dere hver datastruktur på detaljnivå
- Lær dere og forstå hvilke egenskaper de har og når de kan brukes!

Dynamisk programmering

Dynamisk programmering

Krav til DP:

- Optimal substruktur
- Overlappende subproblemer

Hvordan gjøre det i praksis?

- Memoisering
- Bottom-up problemløsing

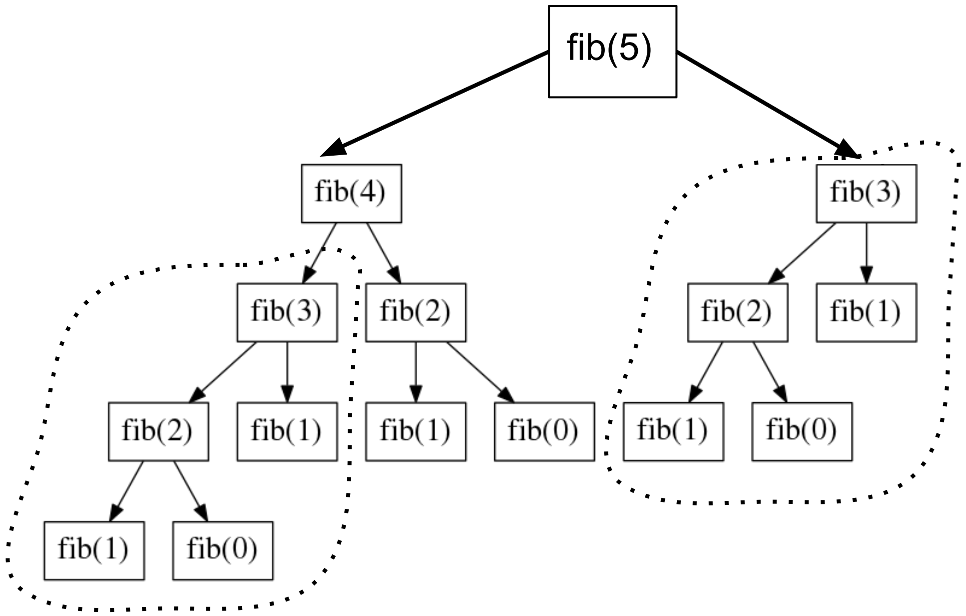
Dynamisk programmering - motivasjon

Fibonacci-tall

$$f_n = \begin{cases} 0 & \text{hvis } n = 0 \\ 1 & \text{hvis } n = 1 \\ f_{n-1} + f_{n-2} & \text{hvis } n > 1 \end{cases}$$

Fibonacci

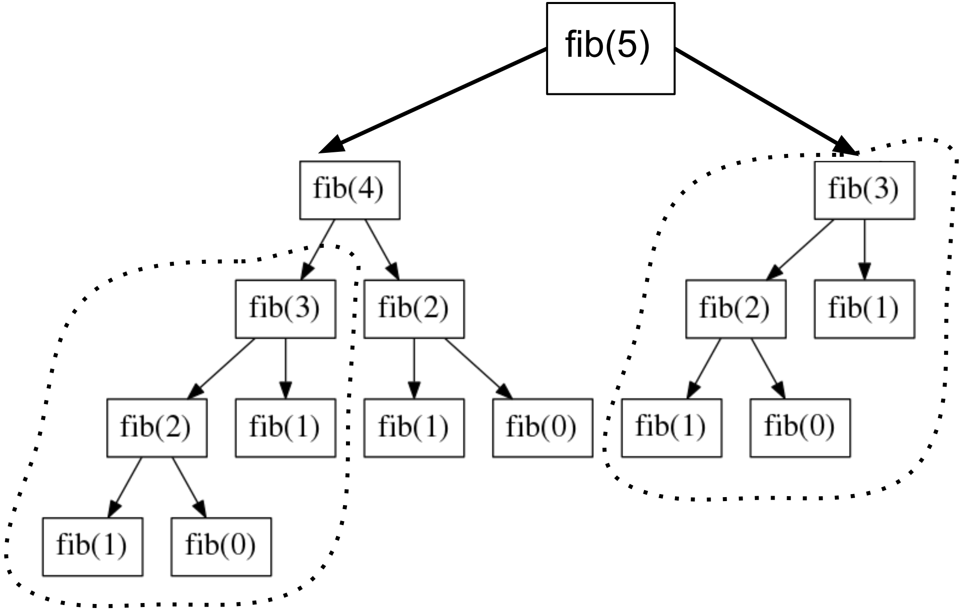
$$f_n = \begin{cases} 0 & \text{hvis } n = 0 \\ 1 & \text{hvis } n = 1 \\ f_{n-1} + f_{n-2} & \text{hvis } n > 1 \end{cases}$$



Fibonacci

$$f_n = \begin{cases} 0 & \text{hvis } n = 0 \\ 1 & \text{hvis } n = 1 \\ f_{n-1} + f_{n-2} & \text{hvis } n > 1 \end{cases}$$

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

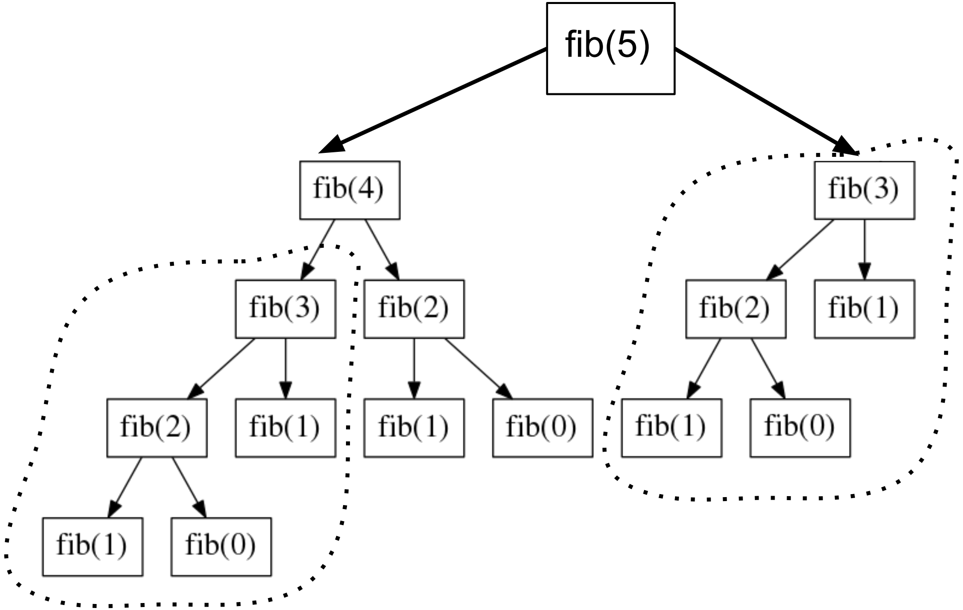


Fibonacci

$$f_n = \begin{cases} 0 & \text{hvis } n = 0 \\ 1 & \text{hvis } n = 1 \\ f_{n-1} + f_{n-2} & \text{hvis } n > 1 \end{cases}$$

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

La oss prøve å kjøre den på noen n'er!

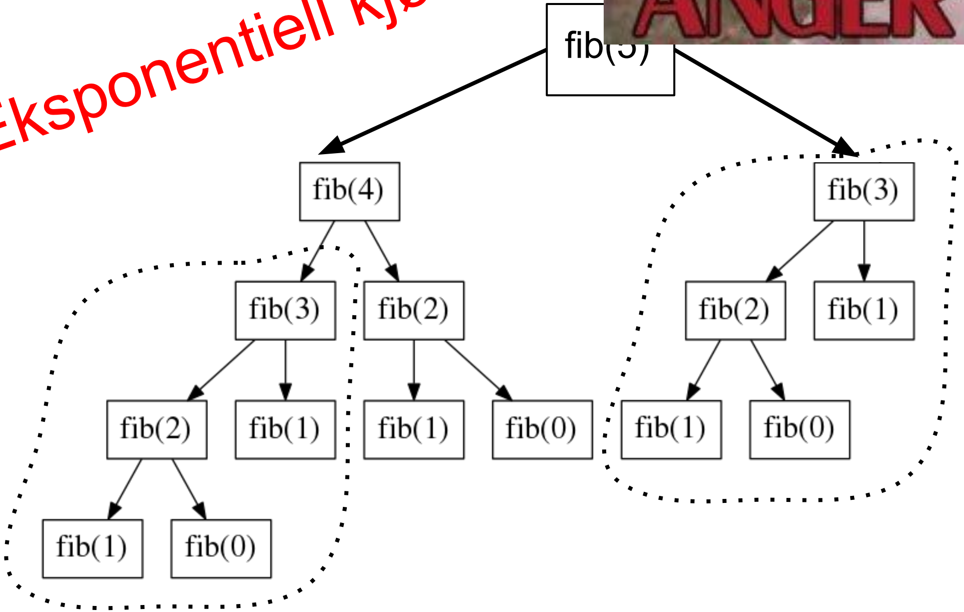


Fibonacci

$$f_n = \begin{cases} 0 & \text{hvis } n = 0 \\ 1 & \text{hvis } n = 1 \\ f_{n-1} + f_{n-2} & \text{hvis } n > 1 \end{cases}$$


```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Eksponentiell kjøretid !



La oss prøve å kjøre den på noen n’er!

Det er jo kjempetregt !

Forbedre Fibonacci

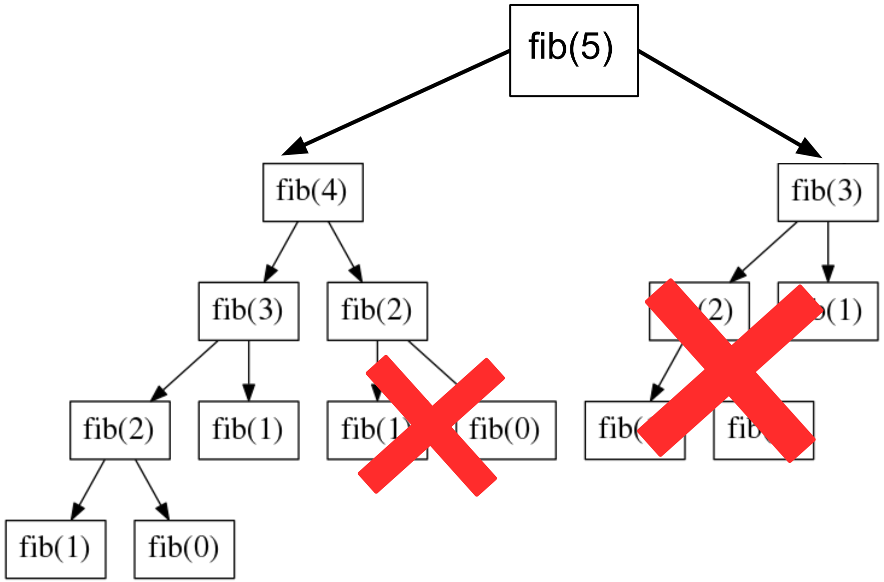
Vi må utnytte de overlappende delproblemene

Forbedre Fibonacci

Vi må utnytte de overlappende delproblemene

```python
memo = {}

def fib(n):
    if n in memo:
        return memo[n]

    f = 0

    if n == 0:
        f = 0
    elif n == 1:
        f = 1
    else:
        f = fib(n-1) + fib(n-2)

    memo[n] = f
    return f
```

La oss prøvekjøre vidunderet!



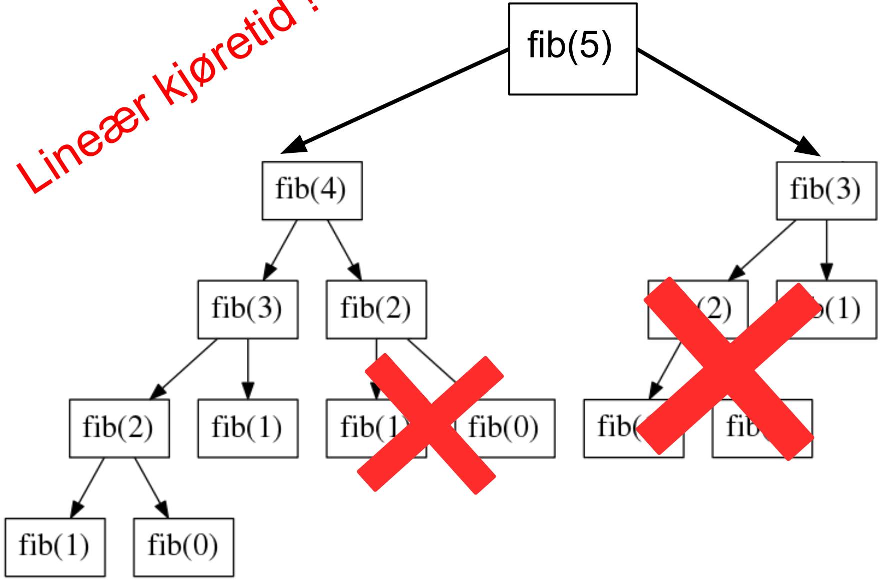
Forbedre Fibonacci

Vi må utnytte de overlappende delproblemene

```python
memo = {}

def fib(n):
    if n in memo:
        return memo[n]

    f = 0

    if n == 0:
        f = 0
    elif n == 1:
        f = 1
    else:
        f = fib(n-1) + fib(n-2)

    memo[n] = f
    return f
```

La oss prøvekjøre vidunderet!

Lineær kjøretid ! <3



Enda bedre?

0	1	1	2	3	5	8	13

```python
def fib(n):
    f = [None] * max(2,n+1)
    f[0] = 0
    f[1] = 1

    for i in xrange(2, n+1):
        f[i] = f[i-1] + f[i-2]

    return f[n]
```

La oss prøvekjøre denne

Enda bedre?

0	1	1	2	3	5	8	13

```python
def fib(n):
    f = [None] * max(2,n+1)
    f[0] = 0
    f[1] = 1

    for i in xrange(2, n+1):
        f[i] = f[i-1] + f[i-2]

    return f[n]
```

Også lineær kjøretid <3

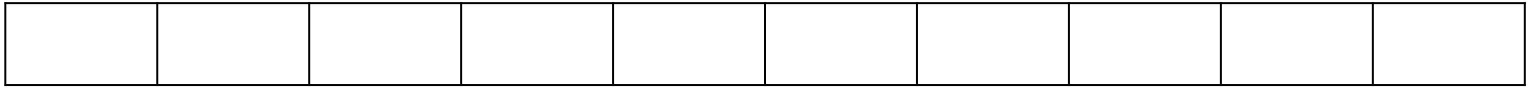
La oss prøvekjøre denne

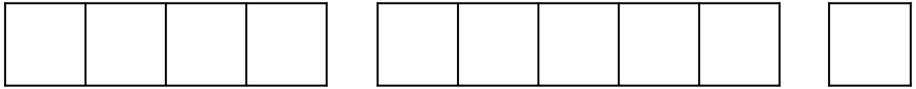
Dynamisk programmering

- Naiv (dum) løsning:
  - Lett, men eksponentiell tid
- Memoisering
  - Lineær tid
- Bottom up
  - Lineær tid

- Problemer som består av delproblemer
- Delproblemene overlapper (brukes flere ganger)
- Som regel er vi interessert i optimaliseringsproblemer
  - La oss se på et

Rod Cutting





9 + 10 + 1 = 20

Optimal substruktur

1	$1
2	$5
3	$8
4	$9
5	$10
6	$17
7	$17
8	$20
9	$24
10	$30

Rod Cutting

r_n: maks avkastning for stang av lengde n
p_n: prisen for en stang av lengde n

$$r_n = \max(p_n, r_1+r_{n-1}, r_2+r_{n-2}, \dots , r_{n-2}+r_2, r_{n-1}+r_1)$$

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

1	\$1
2	\$5
3	\$8
4	\$9
5	\$10
6	\$17
7	\$17
8	\$20
9	\$24
10	\$30

Rod Cutting

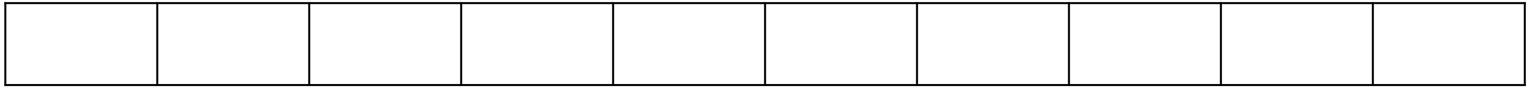
1	$1
2	$5
3	$8
4	$9
5	$10
6	$17
7	$17
8	$20
9	$24
10	$30

r_n: maks avkastning for stang av lengde n
p_n: prisen for en stang av lengde n

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

```
CUT-ROD(p,n)
        if n == 0
                return 0
        q = -∞
        for i = 1 to n
                q = max(q, p[i] + CUT-ROD(p, n-i))
        return q
```

Rod Cutting



r_n: maks avkastning for stang av lengde n
p_n: prisen for en stang av lengde n

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

```
CUT-ROD(p,n)
        if n == 0
                return 0
        q = -∞
        for i = 1 to n
                q = max(q, p[i] + CUT-ROD(p, n-i))
        return q
```

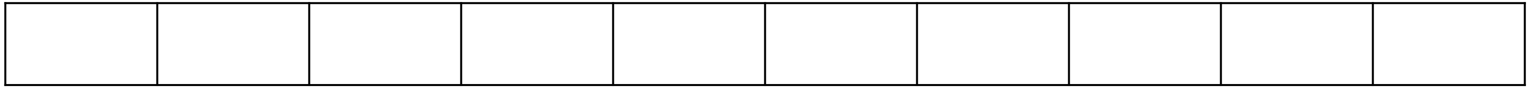
Eksponentiell kjøretid :-(

Men vi kan memoisere :-)

Hva blir kjøretiden da?

1	$1
2	$5
3	$8
4	$9
5	$10
6	$17
7	$17
8	$20
9	$24
10	$30

Rod Cutting



1	$1
2	$5
3	$8
4	$9
5	$10
6	$17
7	$17
8	$20
9	$24
10	$30

r_n: maks avkastning for stang av lengde n
p_n: prisen for en stang av lengde n

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

Hva hvis vi skal bygge løsningen nedenfra og opp?
La oss prøve å gjøre det

Rod Cutting

r_n: maks avkastning for stang av lengde n
p_n: prisen for en stang av lengde n

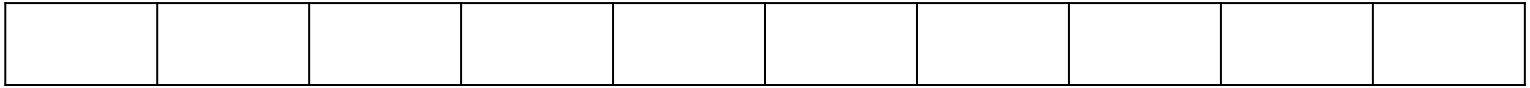
$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

Hva hvis vi skal bygge løsningen nedenfra og opp?
La oss prøve å gjøre det

i	0	1	2	3	4	5	6	7	8	9	10
r[i]	0	1	5	8	10	13	17	18	22	25	30

1	$1
2	$5
3	$8
4	$9
5	$10
6	$17
7	$17
8	$20
9	$24
10	$30

Rod Cutting



r_n: maks avkastning for stang av lengde n
p_n: prisen for en stang av lengde n

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

```
BOTTOM-UP-CUT-ROD(p, n)
    let r[0..n] be a new array
    r[0] = 0
    for j = 1 to n
        q = -∞
        for i = 1 to j
            q = max(q, p[i] + r[j - i])
        r[j] = q
    return r[n]
```

Hva blir kjøretiden?

1	\$1
2	\$5
3	\$8
4	\$9
5	\$10
6	\$17
7	\$17
8	\$20
9	\$24
10	\$30

Rod Cutting

r_n: maks avkastning for stang av lengde n
p_n: prisen for en stang av lengde n

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

```
BOTTOM-UP-CUT-ROD(p, n)
    let r[0..n] be a new array
    r[0] = 0
    for j = 1 to n
        q = -∞
        for i = 1 to j
            q = max(q, p[i] + r[j - i])
        r[j] = q
    return r[n]
```

Hva blir kjøretiden?

$\Theta(n^2)$
Akkurat som memoisering

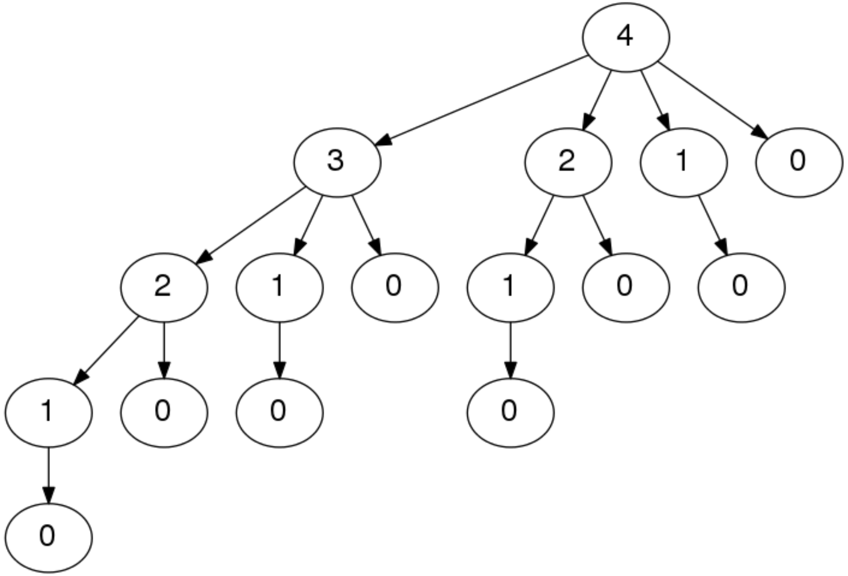
1	$1
2	$5
3	$8
4	$9
5	$10
6	$17
7	$17
8	$20
9	$24
10	$30

Hva skjedde nå?

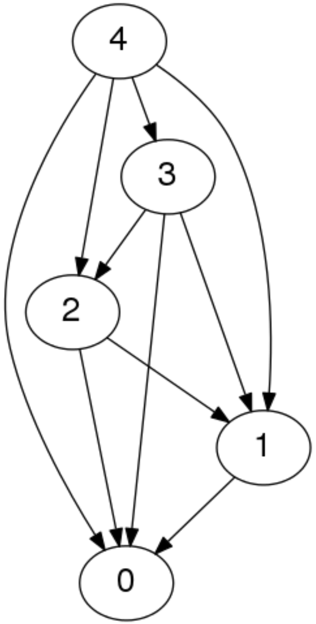
- Problemet vårt hadde optimal substruktur
- Problemet hadde overlappende delproblemer
- Vi sørget for å løse et delproblem kun én gang, og effektivt byttet lagringsplass mot bedre kjøretid

Hva skjedde nå?

- Problemet vårt hadde optimal substruktur
- Problemet hadde overlappende delproblemer
- Vi sørget for å løse et delproblem kun én gang, og effektivt byttet lagringsplass mot bedre kjøretid



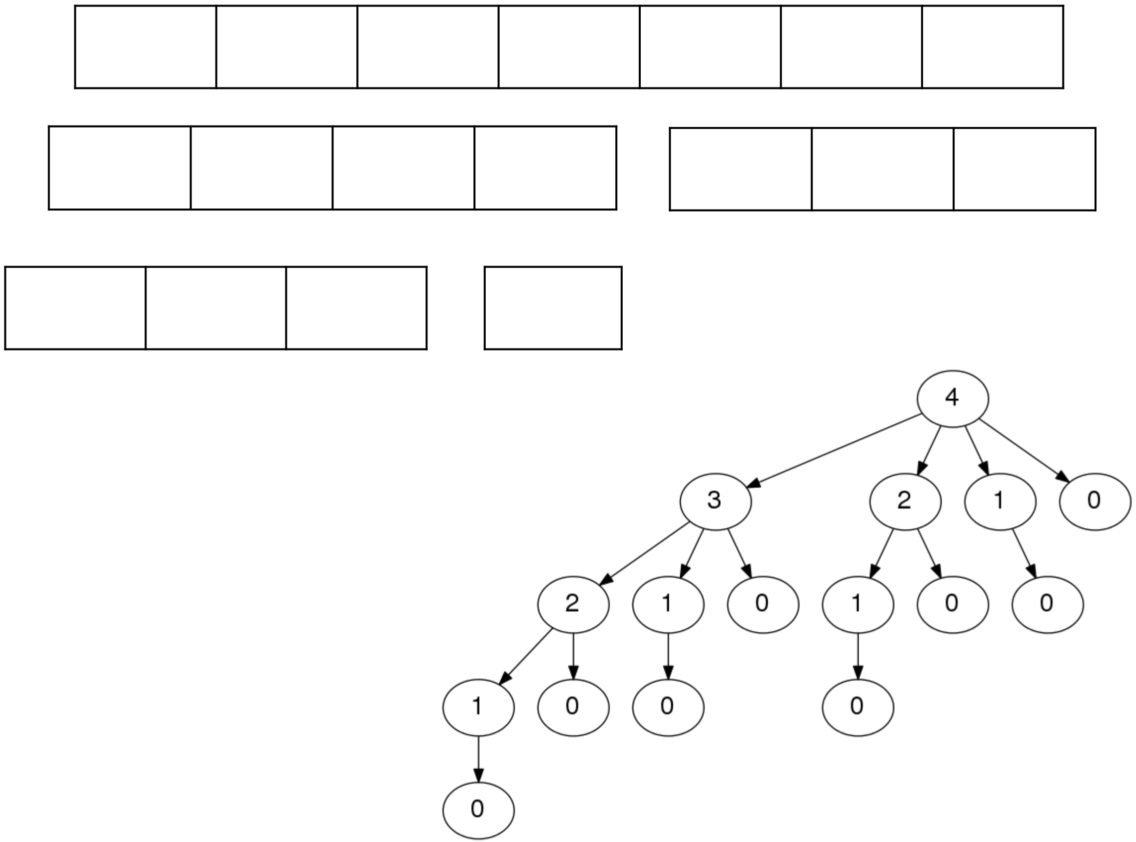
Delproblem-grafen



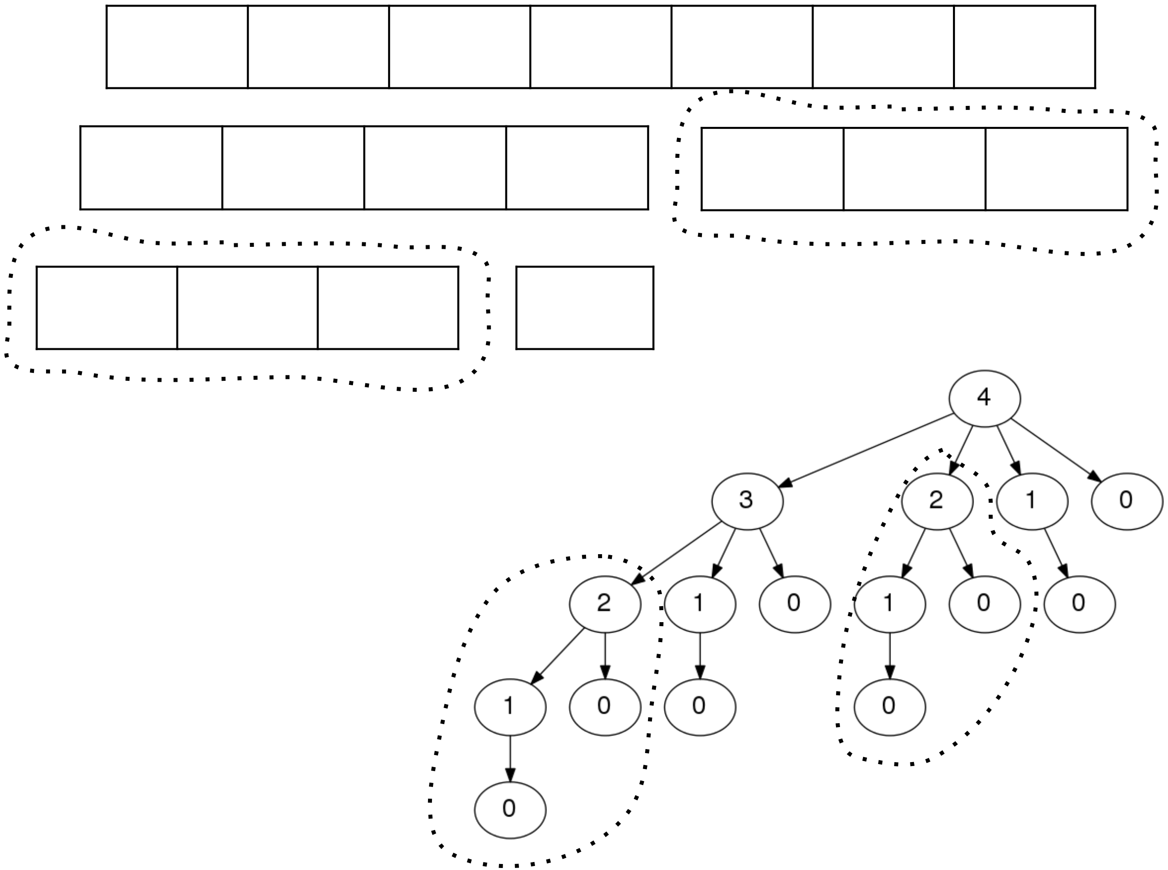
Abstraksjon

Vi må ha:

- Optimal substruktur
- Overlappende delproblemer



Abstraksjon



Vi må ha:

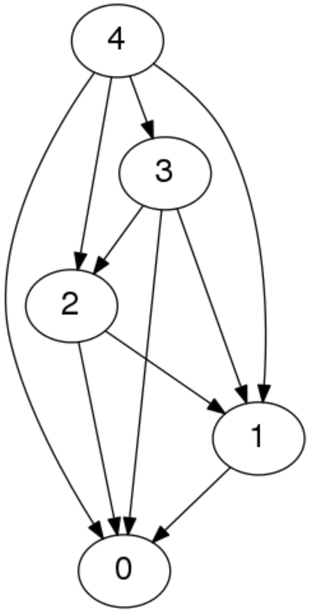
- Optimal substruktur
- Overlappende delproblemer

H2010, oppgave 1h

h) (6 %) Hvorfor er det ikke alltid nyttig å bruke memoisering i rekursive algoritmer?

Så hvordan går vi fram?

1. Beskriv/karakteriser strukturen til en optimal løsning (definer problem-parametere og finn delproblem-grafen) - tenk på hvilke **valg** som må gjøres
2. Definer rekursivt verdien til en optimal løsning
3. Regn ut verdien til en optimal løsning (og husk valgene du gjør)
4. Bygg opp en optimal løsning basert på beregnet informasjon



H2012, oppgave 1f

f) I ryggsekkproblemet (0-1 *knapsack*), la $c[i, w]$ være optimal verdi for de i første objektene, med en kapasitet på w. La v_i og w_i være henholsvis verdien og vekten til objekt i. Fyll ut rekurrensen for $c[i, w]$, hvis vi antar at $i > 0$ og $w_i \leq w$.

Svar (6%): $c[i, w] = \max\{ \qquad\qquad , \qquad\qquad \}$.

H2012, oppgave 1f

f) I ryggsekkproblemet (0-1 *knapsack*), la $c[i, w]$ være optimal verdi for de i første objektene, med en kapasitet på w. La v_i og w_i være henholsvis verdien og vekten til objekt i. Fyll ut rekurrensen for $c[i, w]$, hvis vi antar at $i > 0$ og $w_i \leq w$.

Svar (6%): $c[i, w] = \max\{$, $\}$.