Mail: lchauck@uio.no

Gruppe 8

Intro til IN2010

Mail: lchauck@uio.no

Fra å telle steg til store O

Vi ser på verste tilfelle, aka at n blir drit stor.

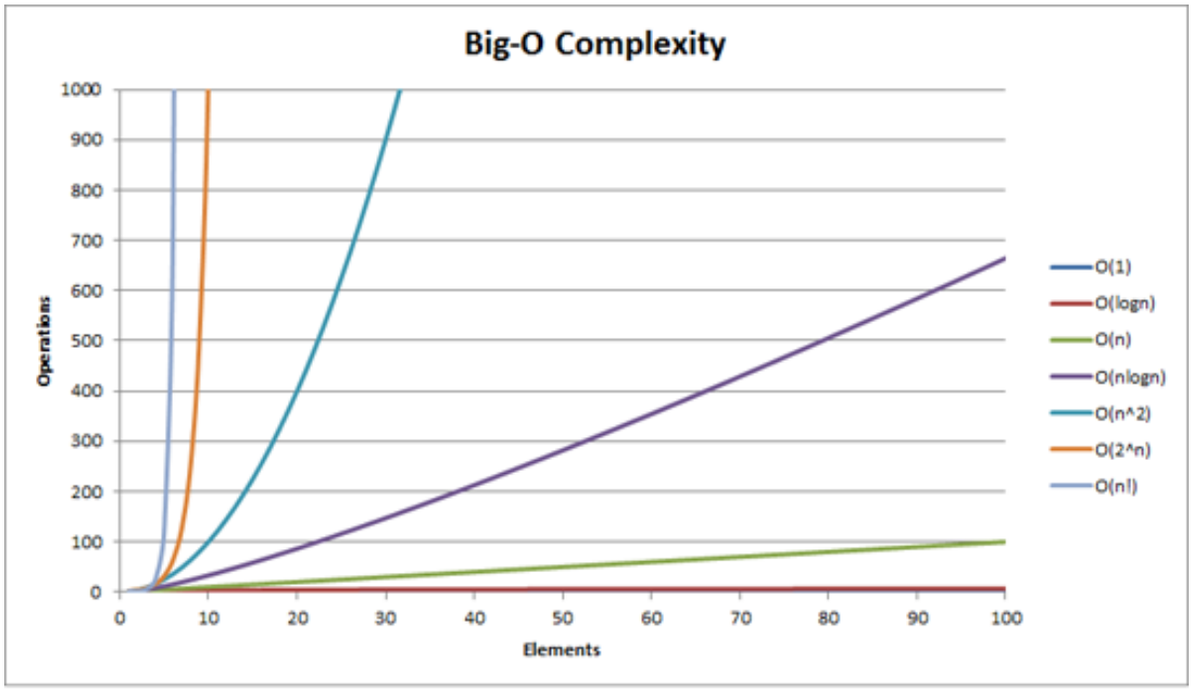
Konverterer addisjon (+) til maks (max).

Eks. Oppgave La oss gjøre dette om til store O:

O(6*log(n) + 3*n + 5)

O(log(n) + n + 1)

O(n)



Mail: lchauck@uio.no

Kjøretidskompleksitet

Lære å kjenne igjen de forskjellige.

Konstant tid: O(1)

Typiske kjennetegn:

Ingen iterasjoner: ingen for- eller while-løkker

Ingen rekursive kall: ingen funksjonskall på seg selv.

```
return 3·n
```

```
n ← 2
n · 3
return n
```

Logaritmisk tid: O(log(n))

Typiske kjennetegn:

Løkke som halverer i for hver iterasjon

```
i ← n
while i > 0 do
    n·3
    i ← ⌊i/2⌋
```

runder ned

$\lfloor \frac{3}{2} \rfloor = 1$

Mail: lchauck@uio.no

Lineær tid: O(n)

Typiske kjennetegn:

Enkle løkker.

```
for i←0 to n-1 do
    n·3
```

```
i←n
while i>0 do
    n·3
```

Kvadratisk tid: O(n^2)

Typiske kjennetegn:

En for/while-løkke med en annen for/while-løkke inni seg.

```
i←n
while i>0 do
    j←n
    while j>0 do
        n·3
```

```
for i←0 to n-1 do
    for j←0 to n-1 do
        n·3
```

O(n·m) $n \neq m$

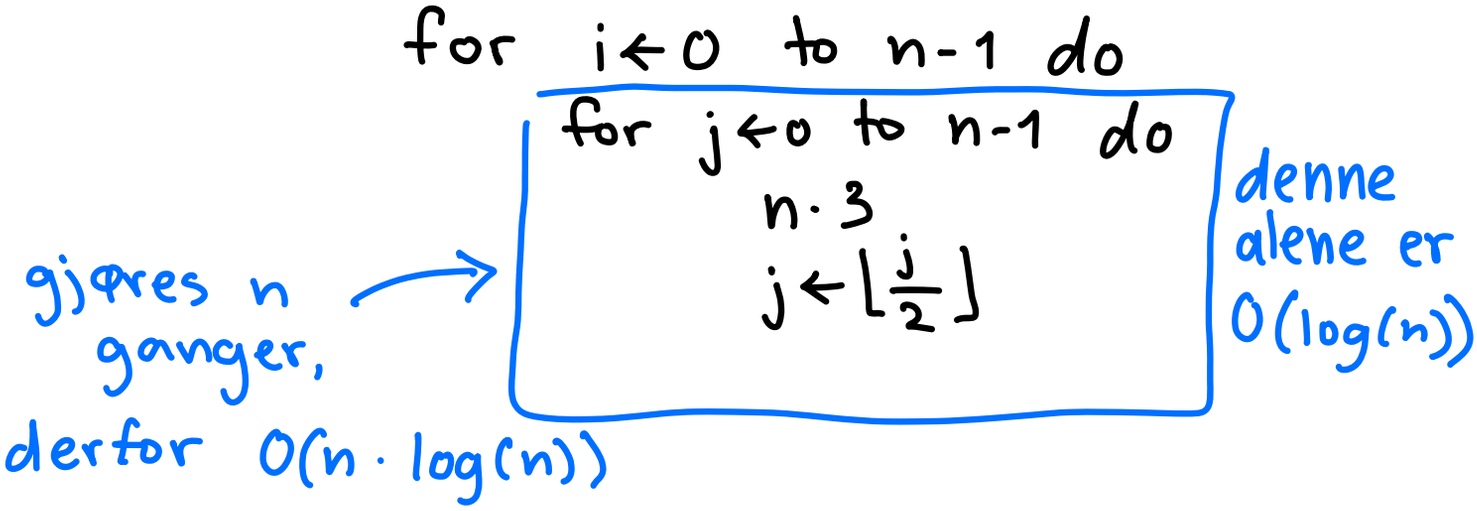
```
for i←0 to n-1 do
    for j←0 to m-1 do
        n·3
```

Mail: lchauck@uio.no

Lineæritmisk tid: O(n * log(n))

Typiske kjennetegn:

Ligner veldig på kvadratisk tid, bare at den indre løkken er logaritmisk, altså at den halverer i for hver iterasjon.

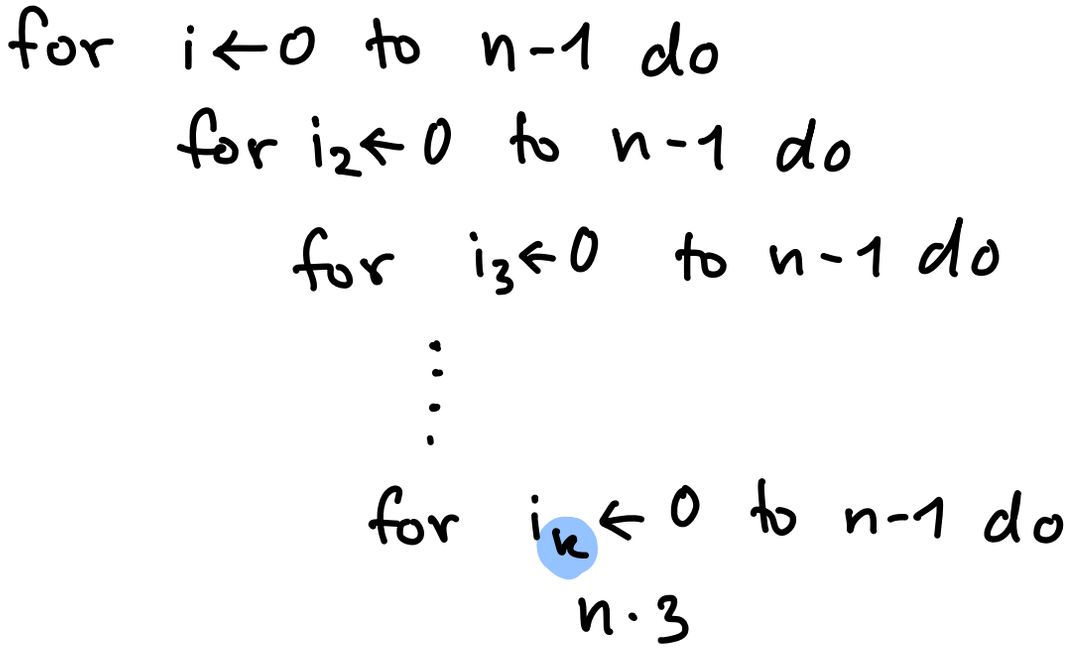


Polynomiell tid: O(n^k)

Typiske kjennetegn:

Samme som **kvadratisk** tid, bare med fler enn to løkker.

Dvs: løkke i løkke i løkke … osv



Mail: lchauck@uio.no

Eksponensiell tid: $O(2^n)$

$2 \cdot 2 \cdot 2 \ldots \cdot 2$ (n ganger)

Typiske kjennetegn:

Hvert rekursive kall resulterer i to nye rekursive kall.

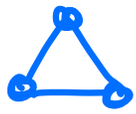
Aka. Algoritmen kaller på seg selv to ganger.

```
Procedure: Expo(n)
    if n=0 do
        return 1
    a ← Expo(n-1)
    b ← Expo(n-1)
    return a+b
```

2 rekursive kall

Mail: lchauck@uio.no

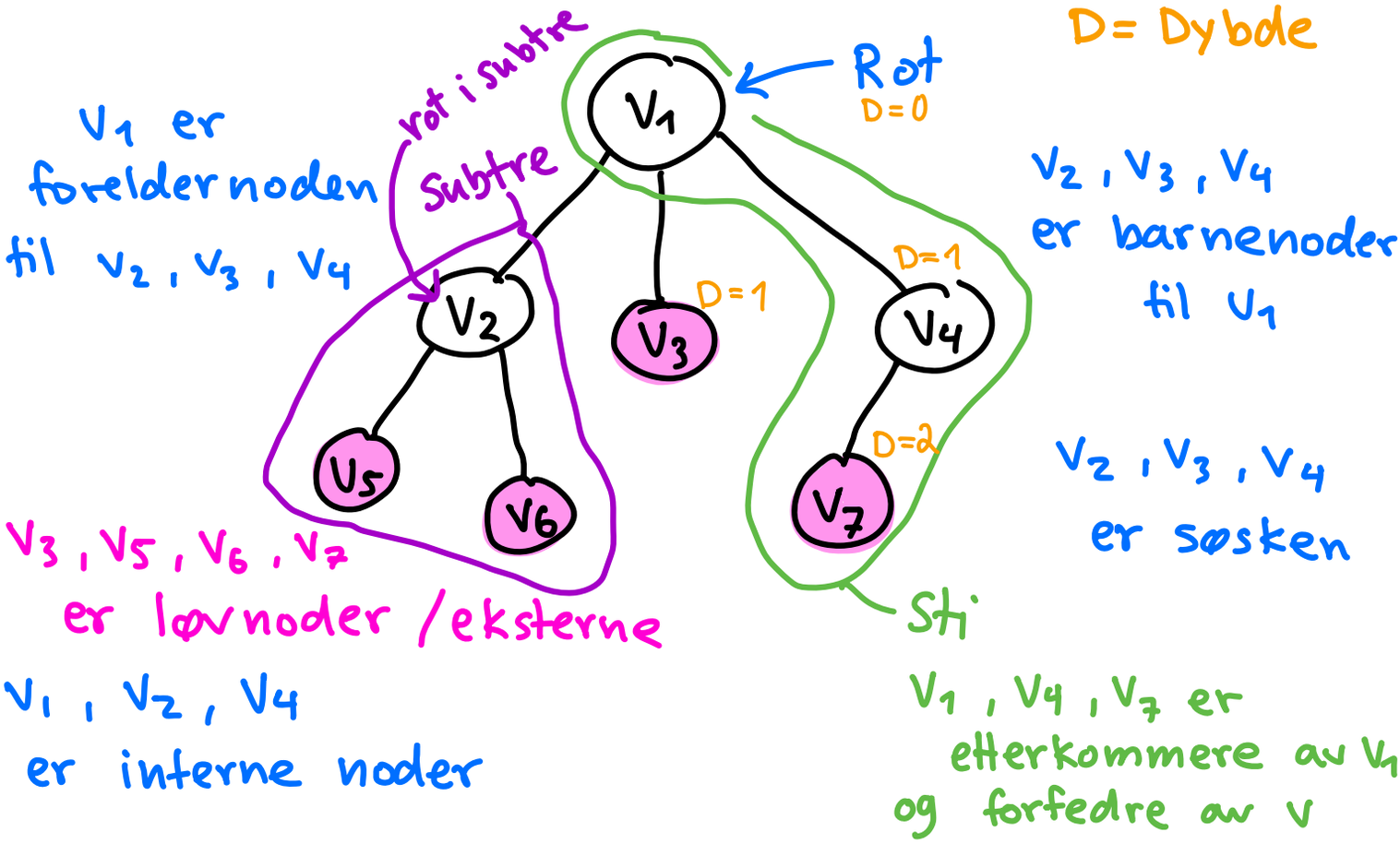
Trær

Ingen sykler. 

Hver node har 0 eller flere barnenoder.

Rot noden har ingen foreldernode.

Alle løvnoder (eller ekstern noder) har nøyaktig 1 foreldernode.



Dybde: rot=0, andre noder = foreldrens dybde+1, tomt tre = -1

Høyde: høyeste avstanden til en etterkommer, aka dybden til den dypeste løvnoden. Høyden her er 2

Traversere: systematisk gå gjennom treet

Preorder: utføre operasjonen på seg selv først, så barna

Postorder: utføre operasjonen på barna først, så seg selv.

Mail: lchauck@uio.no

Binære søketrær

Hver node har maksimalt to barn: *venstre* og *høyre* barn.

Alle elementer i venstre subtre er mindre.

Alle elementer i høyre subtre er større (eller lik).

Innsetting - Binært søketre

ALGORITHM: INNSETTING I ET BINÆRT SØKETRE

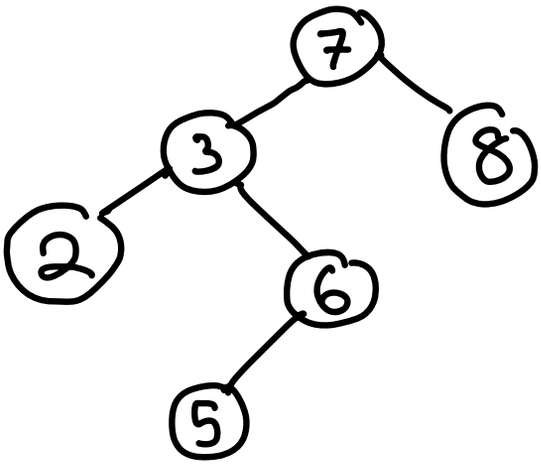
Input: En node v og et element x

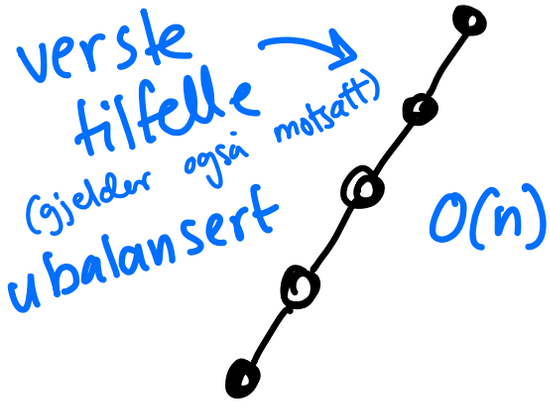
Output: En oppdatert node v der en node som inneholder x er en etterkommer av v

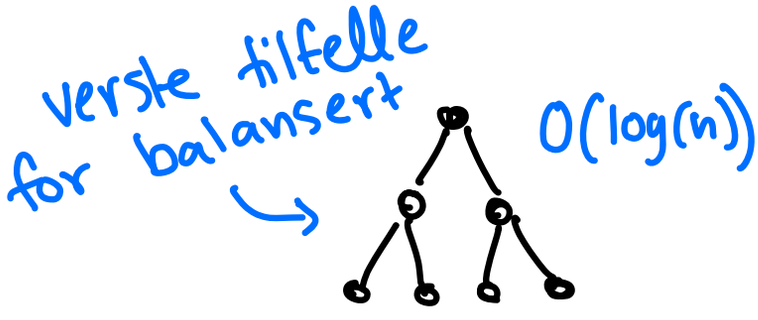
```
1 Procedure Insert(v, x)
2     if v = null then
3         v ← new Node(x)
4     else if x < v.element then
5         v.left ← Insert(v.left, x)
6     else if x > v.element then
7         v.right ← Insert(v.right, x)
8     return v
```

Kjøretidskompleksitet (ubalansert): Konstant tid O(n), der n=antall noder

Kjøretidskompleksitet (balansert): Konstant tid O(log(n))







Mail: lchauck@uio.no

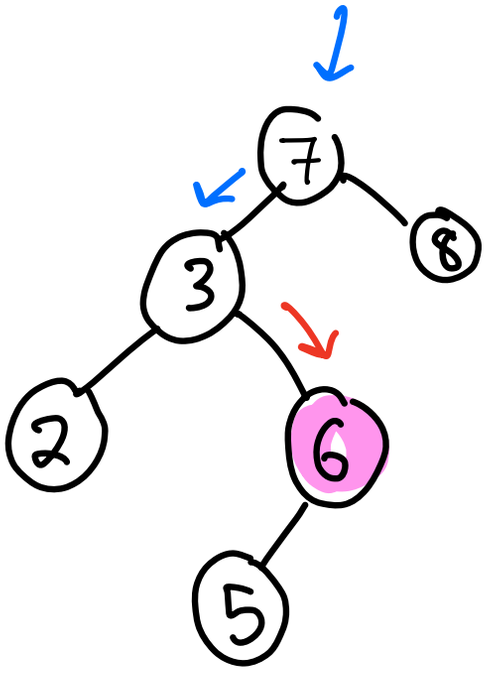
Oppslag - Binært søketre

ALGORITHM: OPPSLAG I ET BINÆRT SØKETRE

Input: En node v og et element x
Output: Dersom x forekommer i en node u som en etterkommer av v, returner u, ellers `null`.

```
1 Procedure Search(v, x)
2     if v = null then
3         return null
4     if v.element = x then
5         return v
6     if x < v.element then
7         return Search(v.left, x)
8     if x > v.element then
9         return Search(v.right, x)
```

Samme kompleksitet som innsetting.



Mail: lchauck@uio.no

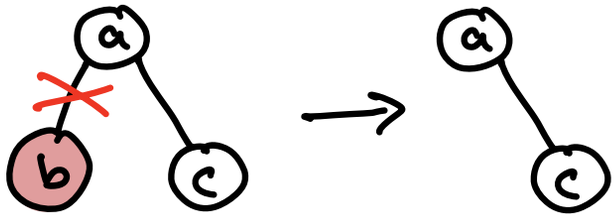
Sletting - Binært søketre

OBS: vi må tette eventuelle hull!

Vi har tre tilfeller:

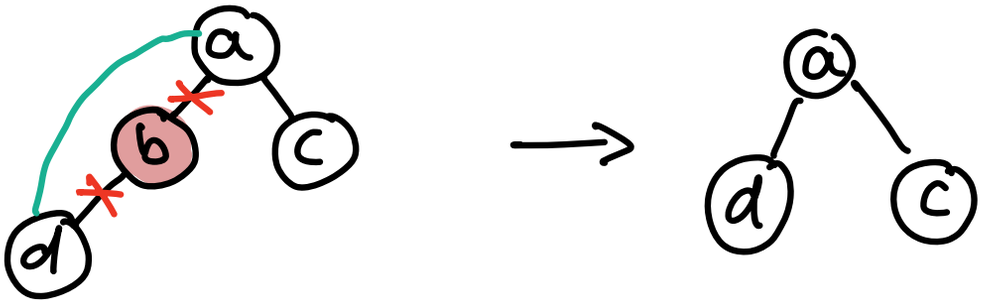
Noden vi sletter har ingen barn

Fjerner pekeren til x



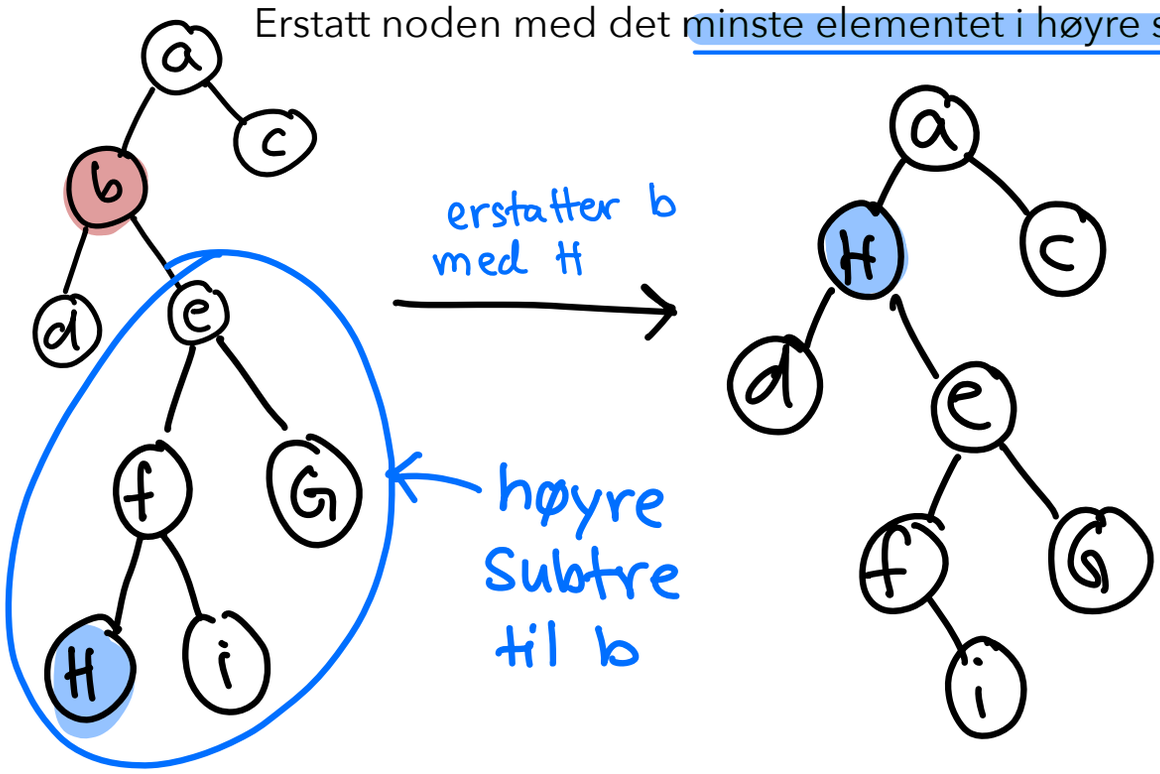
Noden vi sletter har ett barn

Erstatt noden med barnet.



Noden vi sletter har to barn

Erstatt noden med det minste elementet i høyre subtre.



Mail: lchauck@uio.no

AVL-trær

En type selvbalanserende binært søketre.

Treet balanseres med **rotasjoner** når høydeforskjellen (etter innsetting/sletting) mellom venstre og høyre subtre er **større enn 1**

Høyden på en node i AVL trær

0 hvis noden er null.

1 + Max(Height(v.left), Height(v.right))

Rotasjoner

Venstrerotasjon

ALGORITHM: VENSTREROTASJON AV ET BINÆRTRE

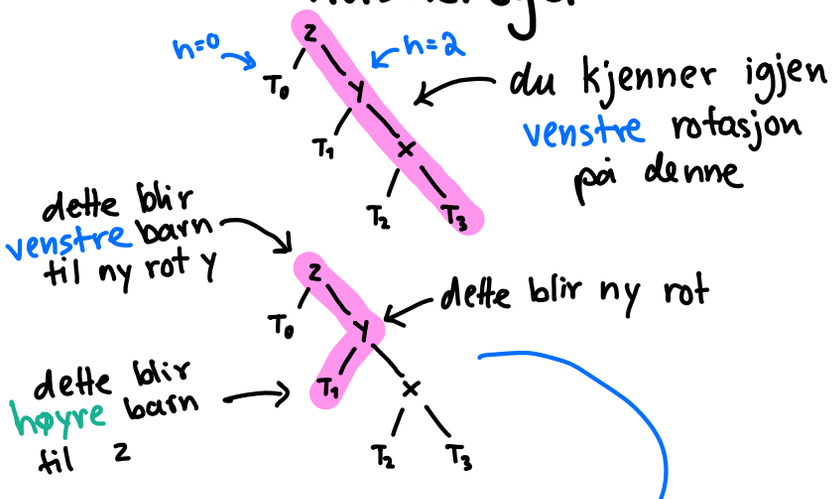
Input: En node z
Output: Roter treet til venstre, slik at y blir den nye roten

```
1 Procedure LeftRotate(z)
2     y ← z.right
3     T1 ← y.left
4     y.left ← z
5     z.right ← T1
6     SetHeight(z)
7     SetHeight(y)
8     return y
```

rotasjonen skjer her!

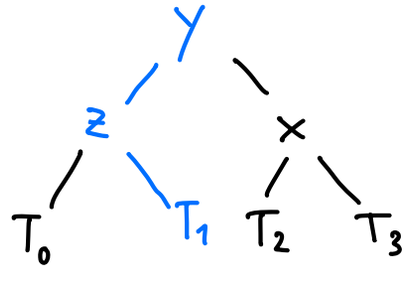
husk å oppdatere høyden

Huskeregel:



venstre rotasjon!

1. y blir vår nye rot
2. vår gamle rot (z) blir y sitt venstre barn
3. T_1, som er siste av de tre nodene markert i rosa, blir den gamle roten (z) sitt nye høyre barn
4. resten av nodene er urørte og fylles inn



Mail: lchauck@uio.no

Høyrerotasjon

ALGORITHM: HØYREROTASJON AV ET BINÆRTRE

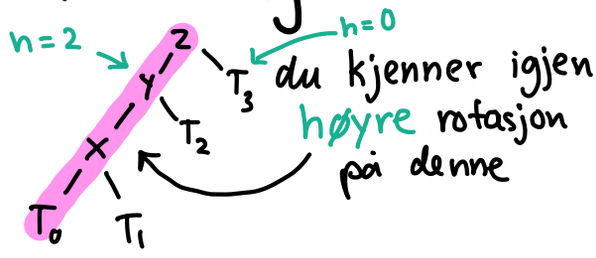
Input: En node z
Output: Roter treet til høyre, slik at y blir den nye roten

```
1 Procedure RightRotate(z)
2     y ← z.left
3     T2 ← y.right
4     y.right ← z
5     z.left ← T2
6     SetHeight(z)
7     SetHeight(y)
8     return y
```

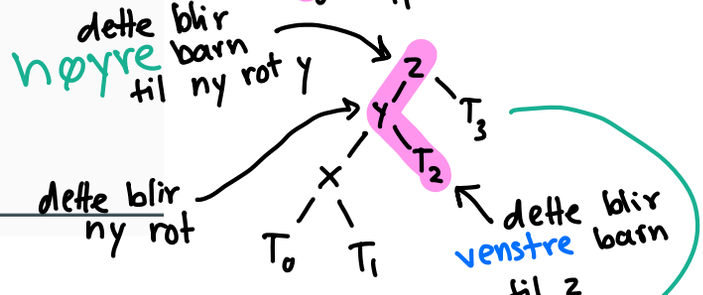
rotasjonen skjer her!

husk å oppdatere høyden

Huskeregel:

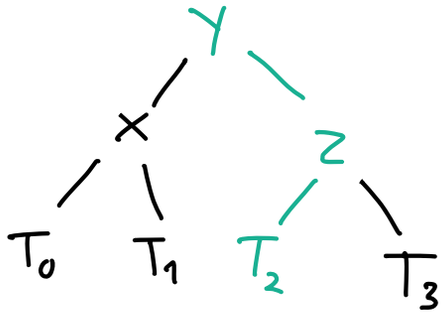


du kjenner igjen høyre rotasjon på denne



høyre rotasjon!

1. y blir vår nye rot
2. vår gamle rot (z) blir y sitt høyre barn
3. T_1, som er siste av de tre nodene markert i rosa, blir den gamle roten (z) sitt nye venstre barn
4. resten av nodene er urørte og fylles inn

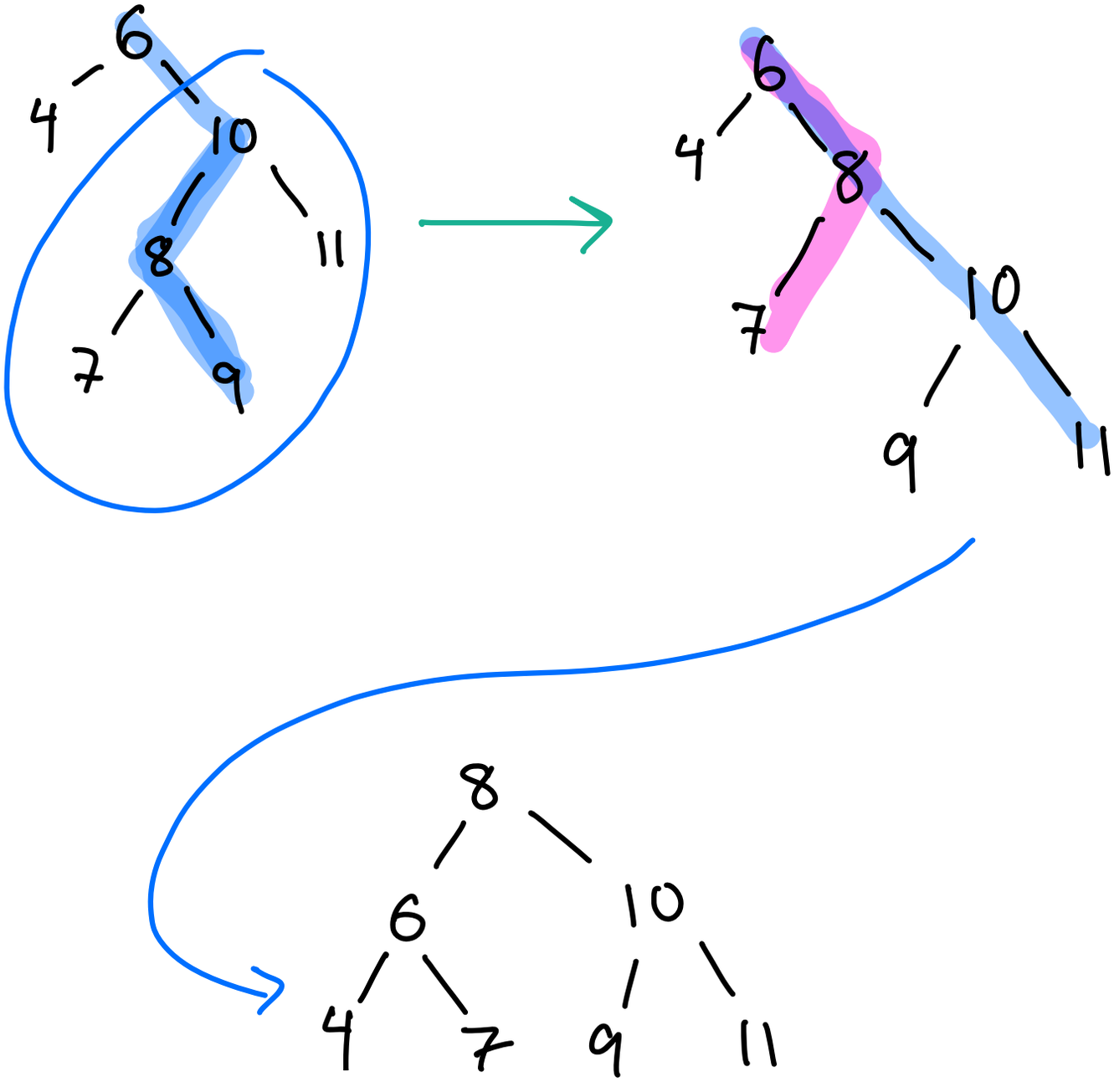


Mail: lchauck@uio.no

Doble rotasjoner på AVL trær

Høyretungt

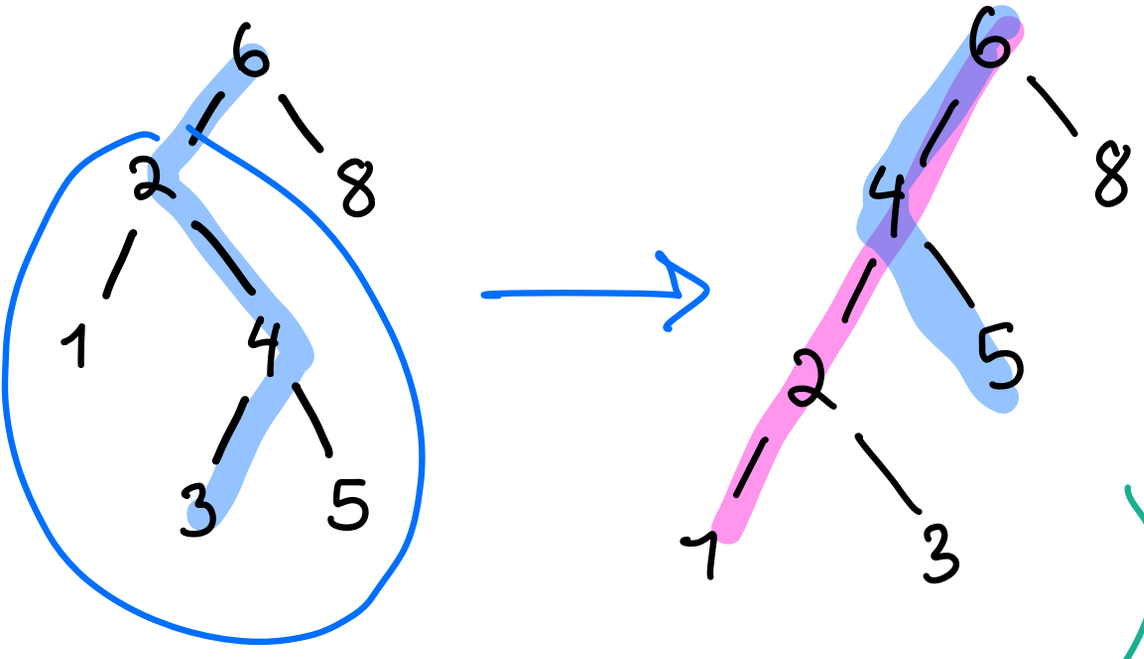
Starter med høyrerotasjon, så en venstrerotasjon

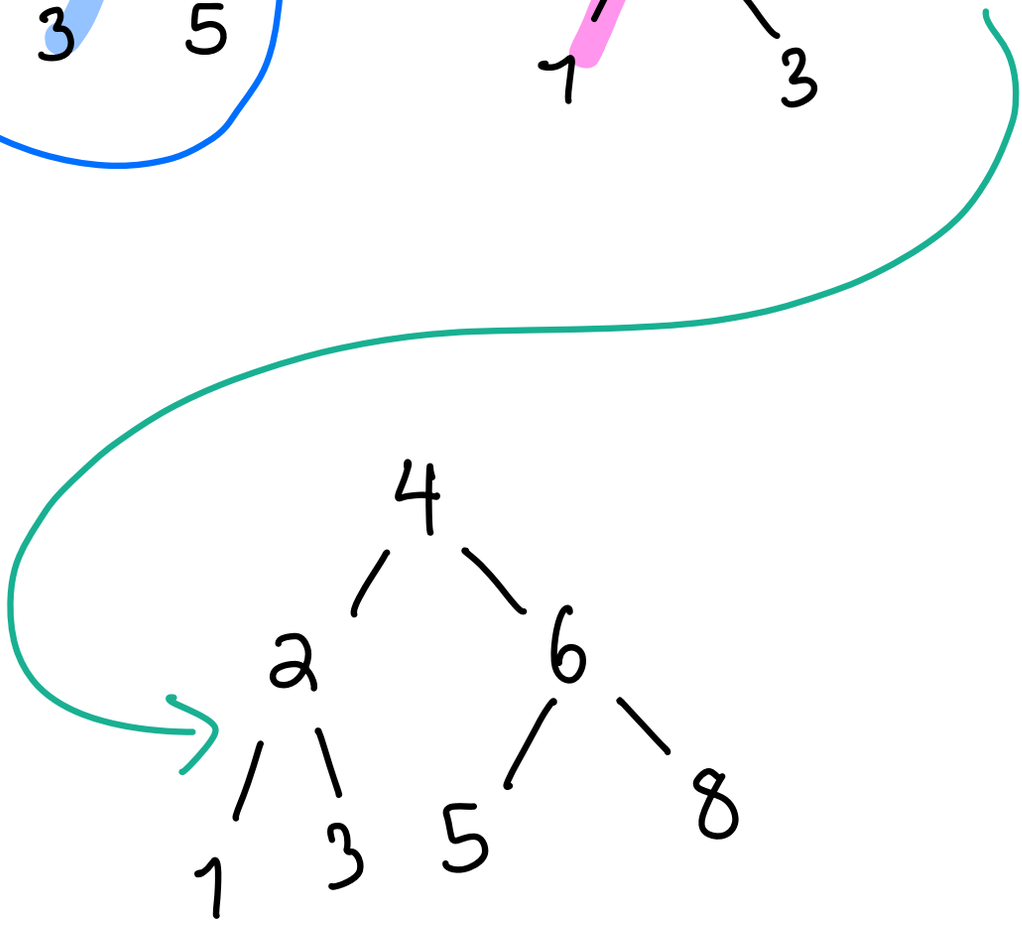


Mail: lchauck@uio.no

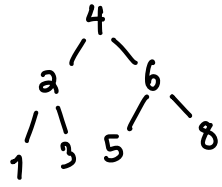
Venstretungt

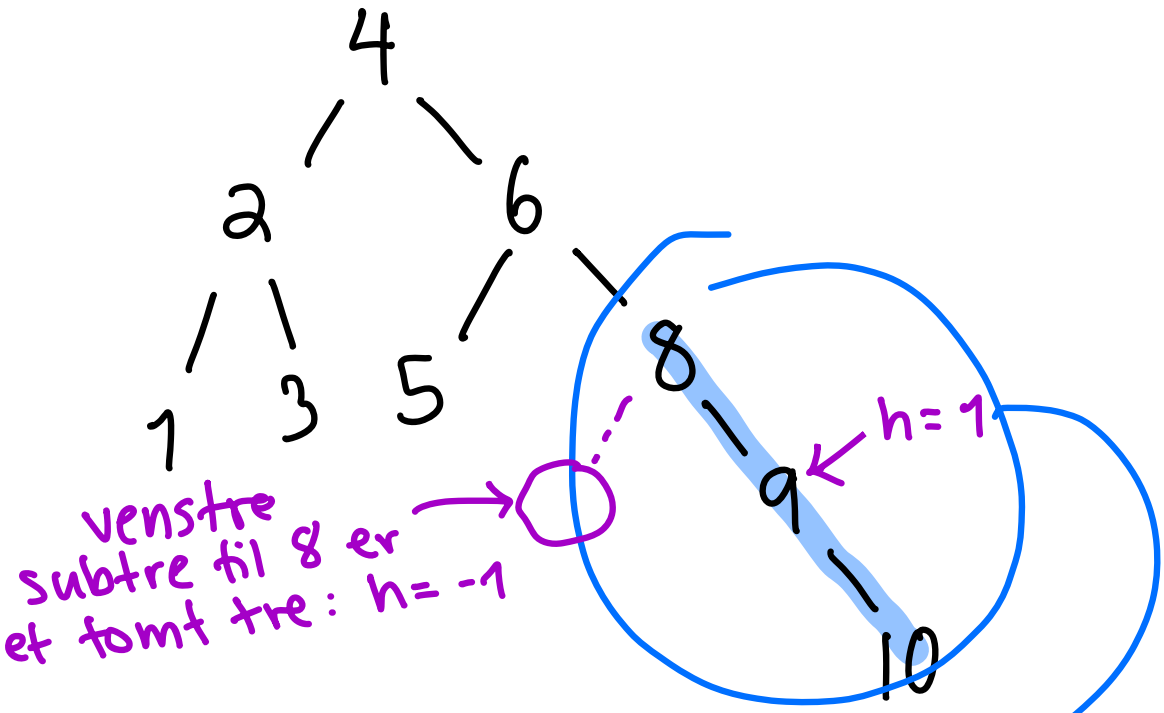
Starter med venstrerotasjon, så en høyrerotasjon.





Oppgave: Sett inn 9, 10 i treet





venstre rotasjon

