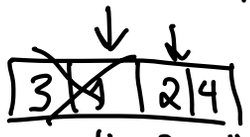
Gruppe 8

Binære heaps - Prioritetskøer, Heaps

Først litt kompleksitet:)

Søk

Binærsøk funker ikke på usortert array!



Intersection2 vil aldri være riktig fordi A ikke sorteres

Anta at vi har tre prosedyrer:

- `Search` – som gjør et rett frem søk på lineær tid
- `BinarySearch` – som gjør et binærsøk
- `Sort` – sorterer i $\mathcal{O}(n \cdot \log(n))$

Prosedyrene nedenfor tar et array A med heltall av størrelse n og et array B av størrelse k som input. Alle skal printe tallene som er i både A og B.

```
1 Procedure Intersection1(A, B)
2     for x in B do
3         if Search(A, x) then
4             print(x)
```

```
1 Procedure Intersection2(A, B)
2     for x in B do
3         if BinarySearch(A, x) then
4             print(x)
```

```
1 Procedure Intersection3(A, B)
2     Sort(A)
3     for x in B do
4         if BinarySearch(A, x) then
5             print(x)
```

```
1 Procedure Intersection4(A, B)
2     for x in B do
3         Sort(A)
4         if BinarySearch(A, x) then
5             print(x)
```

Tre av disse prosedyrene gir alltid riktig svar, og én kan gi feil svar. Vi lurer på når man burde bruke hvilken variant av `Intersection` avhengig av hvor stor k (altså lengden av B) er i forhold til n (altså lengden til A). Du trenger bare å oppgi ett riktig svar hvis flere prosedyrer gir opphav til samme kjøretidskompleksitet.

(a) Hvis B inneholder ett element, altså $k = 1$, hvilken prosedyre gir riktig svar med minimal kjøretidskompleksitet?

Intersection1: $O(k \cdot n) = O(1 \cdot n) = O(n)$

denne er størst

Intersection3: $O(n \cdot \log(n) + k \cdot \log(n)) = O(n \cdot \log(n) + 1 \cdot \log(n))$

O-notasjon konverterer fra addisjon (+) til max

$= O(n \cdot \log(n))$

Intersection4: Sorterer for hver iterasjon, noe som er overflødig

SVAR: Intersection1, fordi vi ser etter den som gjør riktig svar med minimal kjøretidskompleksitet, som betyr at vi ser etter den som bruker kortest tid.

(b) Hvis B inneholder $\log(n)$ elementer, altså $k = \log(n)$, hvilken prosedyre gir riktig svar med minimal kjøretidskompleksitet?

1: $O(k \cdot n) = O(\log(n) \cdot n)$

3: $O(n \cdot \log(n) + k \cdot \log(n)) = O(n \cdot \log(n) + \log(n) \cdot \log(n)$

$= O(n \log(n)$

SVAR: Intersection1 og Intersection3

(c) Hvis B inneholder n elementer, altså $k = n$, hvilken prosedyre gir riktig svar med minimal kjøretidskompleksitet?

1: $O(n \cdot n) = O(n^2)$ (k=n)

3: $O(n \cdot \log(n) + n \cdot \log(n)) = O(2n \log(n)) = O(n \cdot \log(n))$ (k=n)

SVAR: Intersection 3

(d) Hvis B inneholder n^2 elementer, altså $k = n^2$, hvilken prosedyre gir riktig svar med minimal kjøretidskompleksitet?

1: $O(k \cdot n) = O(n^2 \cdot n) = O(n^3)$

3: $O(n \cdot \log(n) + k \cdot \log(n)) = O(n \log(n) + n^2 \log(n))$ (bruker lengst tid)

$= O(n^2 \log(n))$

SVAR: Intersection 3

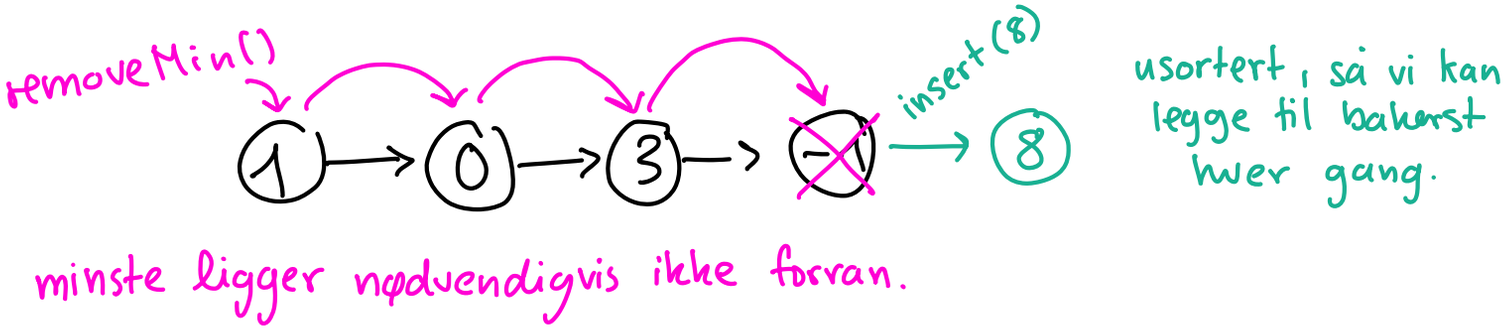
Prioritetskø

Operasjoner: insert(e), removeMin()

Underliggende datastrukturer:

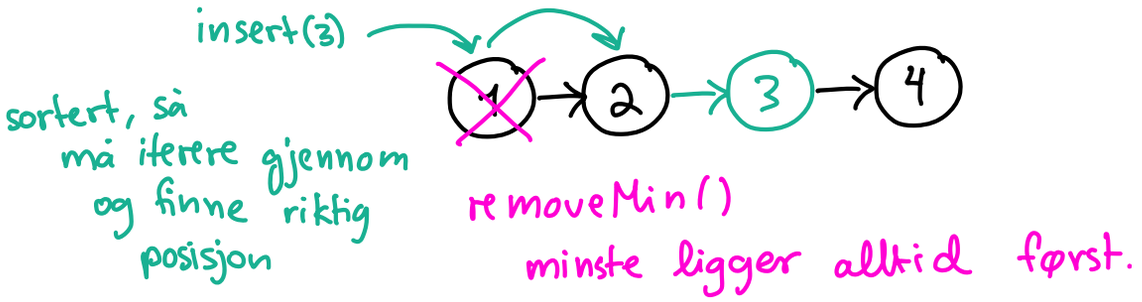
Usortert lenket liste

O(1) på insert, men O(n) på removeMin



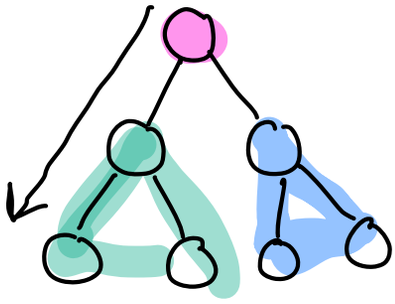
Sortert lenket liste

O(n) på insert, men O(1) på removeMin



Balansert binært søketre

O(log(n)) på insert og O(log(n)) på removeMin()



Heaps

Ukas pensum!

Heaps brukes til å implementere prioritetskøer.

Hva er en prioritetskø?

En kø der element har en «prioritet».
Elementer med høyere prioritet blir behandlet før elementer med lavere prioritet.

Binære heaps

Binære heaps er et binærtre som oppfyller følgende egenskaper:

1. Hver node v, er større enn foreldrenoden. Roten har ingen forelder, så dette gjelder altså ikke for roten.
2. Binærtreet må være **komplett.**

Hvorfor Binære Heaps?

Alltid balanserte!

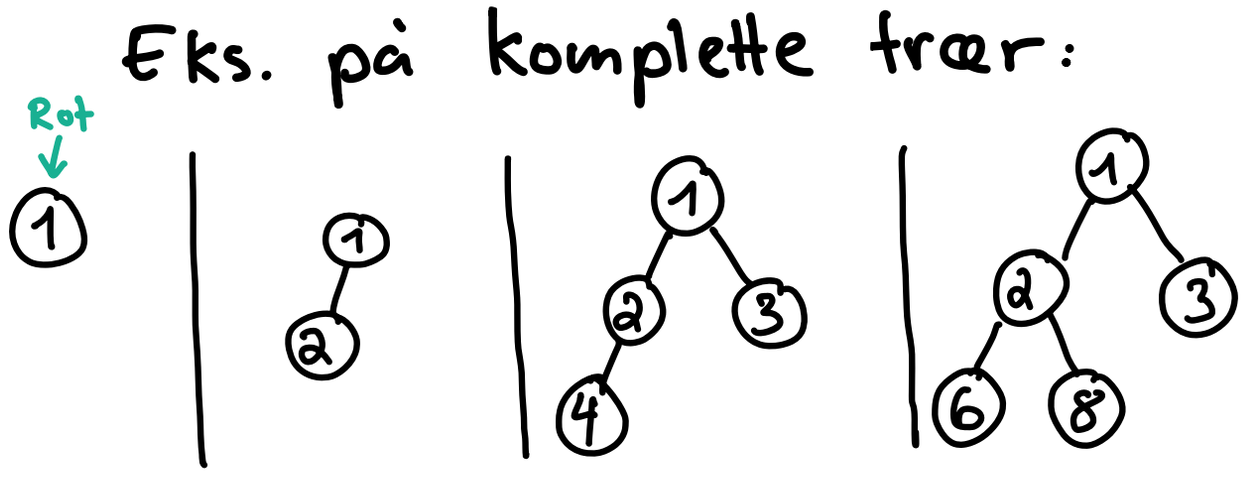
De er **komplette**, så de er alltid balanserte.
Vi trenger ingen rotasjoner.
Mer balanserte enn AVL!
Implementeres effektivt med arrayer.

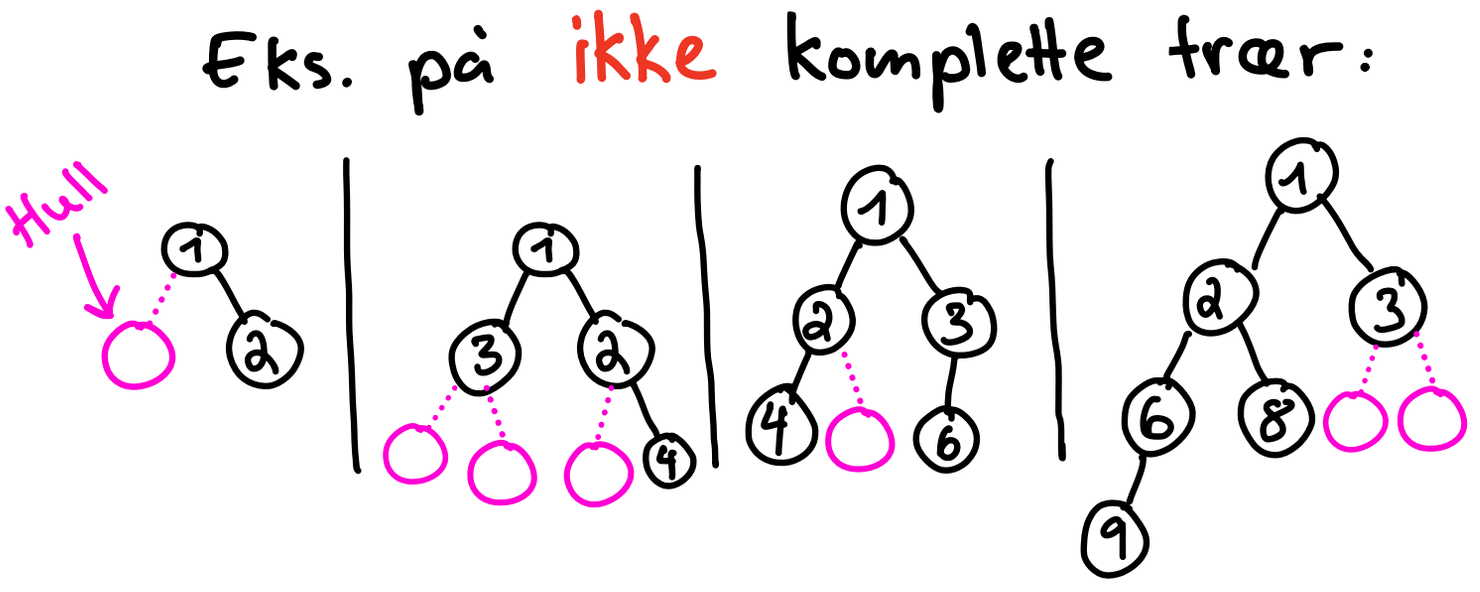
Hva er et komplett binær tre??

1. Alle nivåer utenom siste skal være fulle. 2^n noder i nivå n.

Obs: starter på 0, så rotnivå er nivå 0.

2. Siste nivå er fylt fra venstre uten hull.



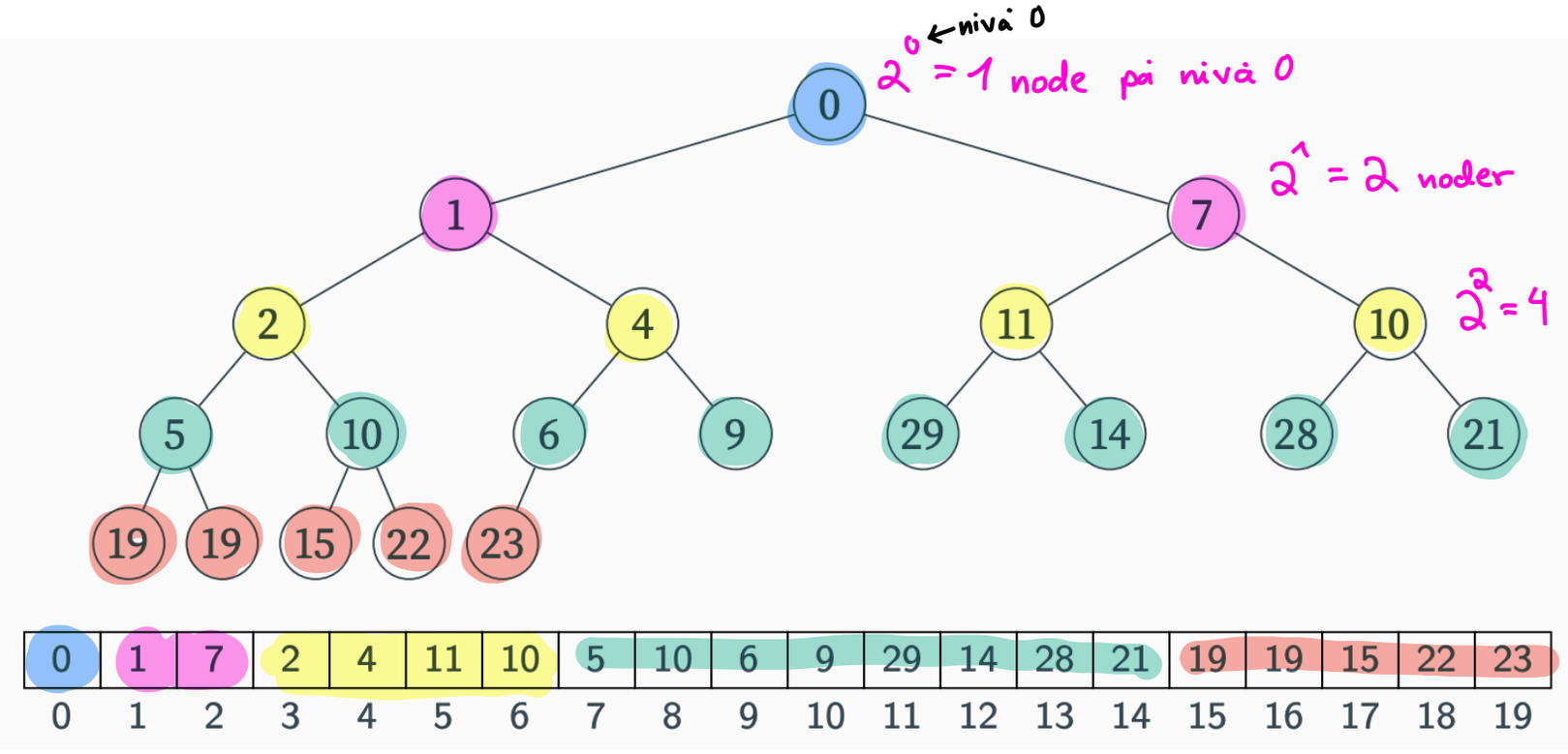


Binære heaps treet

VS.

Array implementasjon

Antall noder: 2^n n=nivå



0	1	7	2	4	11	10	5	10	6	9	29	14	28	21	19	19	15	22	23
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

```
1 Procedure ParentOf(i)
2 |  return ⌊(i−1)/2⌋
```

```
1 Procedure LeftOf(i)
2 |  return 2 · i + 1
```

```
1 Procedure RightOf(i)
2 |  return 2 · i + 2
```

Husk:

1. Hver node v, er større enn foreldrenoden. Roten har ingen forelder, så dette gjelder altså ikke for roten.

2. Binærtreet må være **komplett.**

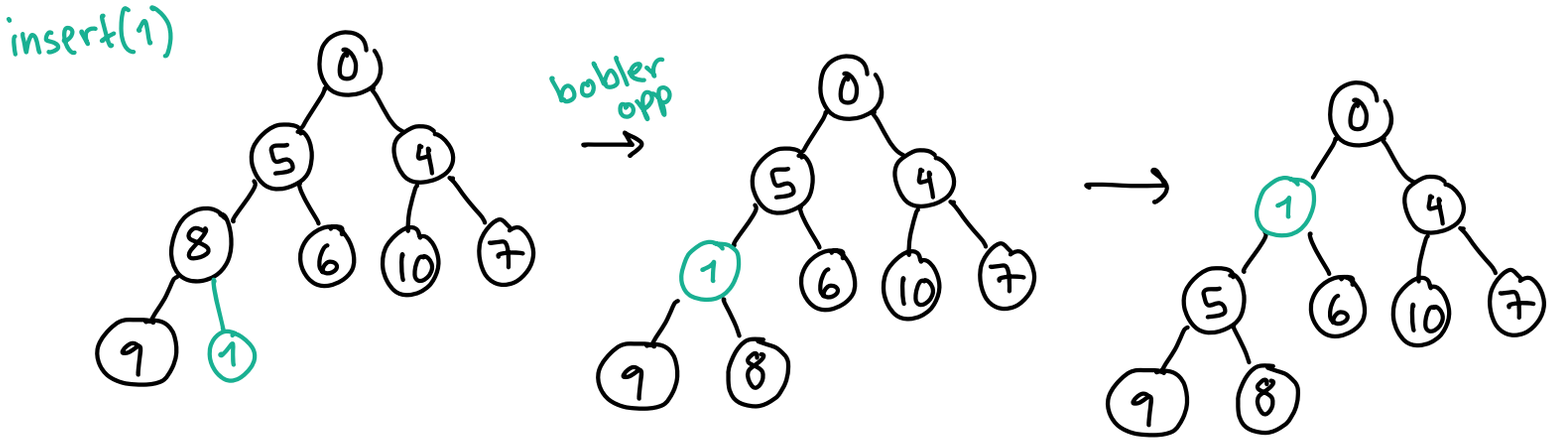
Innsetting i binære heaps - insert

1. Alltid legge til på neste ledige plass. Altså, der neste node må være for at treet fortsatt skal være komplett
2. Hvis noden på den nye plassen er **mindre** enn foreldrenoden, så må de bytte plass! (fortsett rekursivt)

ALGORITHM: INNSETTING I HEAP

Input: Et array A som representerer en heap med n elementer, og et element x
Output: Et array som representerer en heap, som inneholder x

```
1 Procedure Insert(A, x)
2     A[n] ← x
3     i ← n
4     while 0 < i and A[i] < A[ParentOf(i)] do
5         A[i], A[ParentOf(i)] ← A[ParentOf(i)], A[i]
6         i ← ParentOf(i)
```



Sletting i binære heaps - removeMin

1. Slett roten, bytt med **den siste noden** i treet. Altså, den eneste noden som kan fjernes og treet fortsatt er komplett.
2. Hvis noden er **større** enn en av barna, så må den bytte plass med den minste. (fortsett rekursivt)

Binære heaps – fjern minste (implementasjon)

ALGORITHM: FJERNING AV MINSTE ELEMENT FRA HEAP

Input: Et array A som representerer en ikke-tom heap med n elementer
Output: Det minste elementet fjernes og returneres

```
1  Procedure RemoveMin(A)
2      x ← A[0]
3      A[0] ← A[n - 1]
4      i ← 0
5      while LeftOf(i) < n − 1 do
6          j ← LeftOf(i)
7          if RightOf(i) < n − 1 and A[RightOf(i)] < A[j] then
8              j ← RightOf(i)
9          if A[i] ≤ A[j] then
10             return x
11         A[i], A[j] ← A[j], A[i]
12         i ← j
13     return x
```

bytter ut roten med siste

så lenge venstre barn ikke er siste

hvis høyre barn ikke er siste OG høyre barn er mindre enn venstre barn

sammenlikner med minste barn

Hvis den er mindre enn barnet bytter vi

