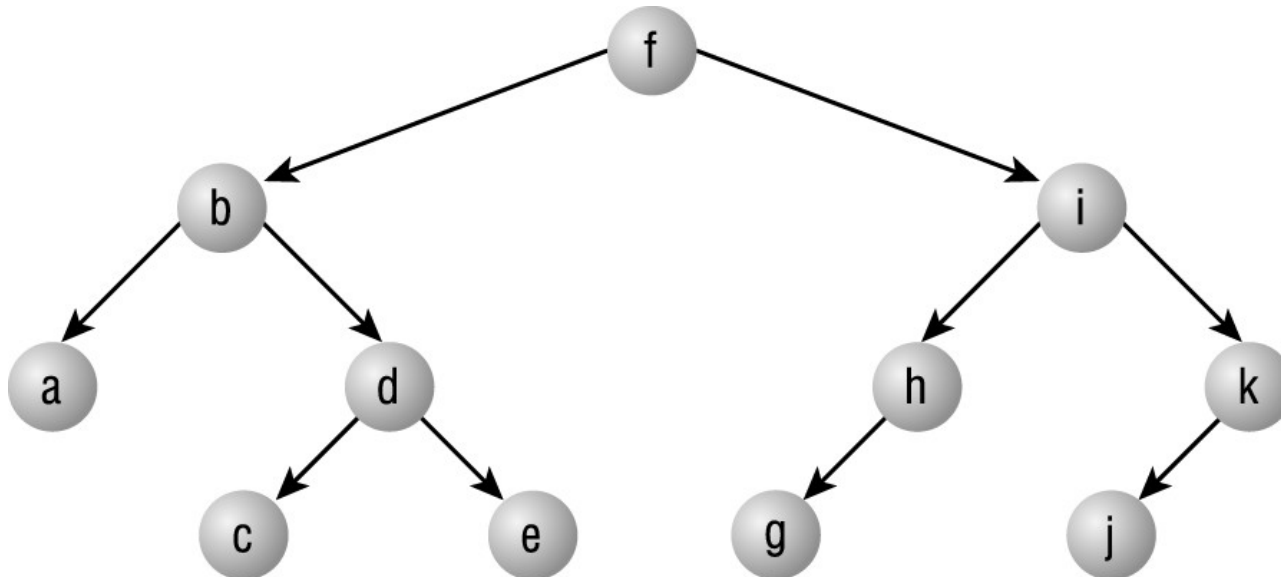


Binære trær

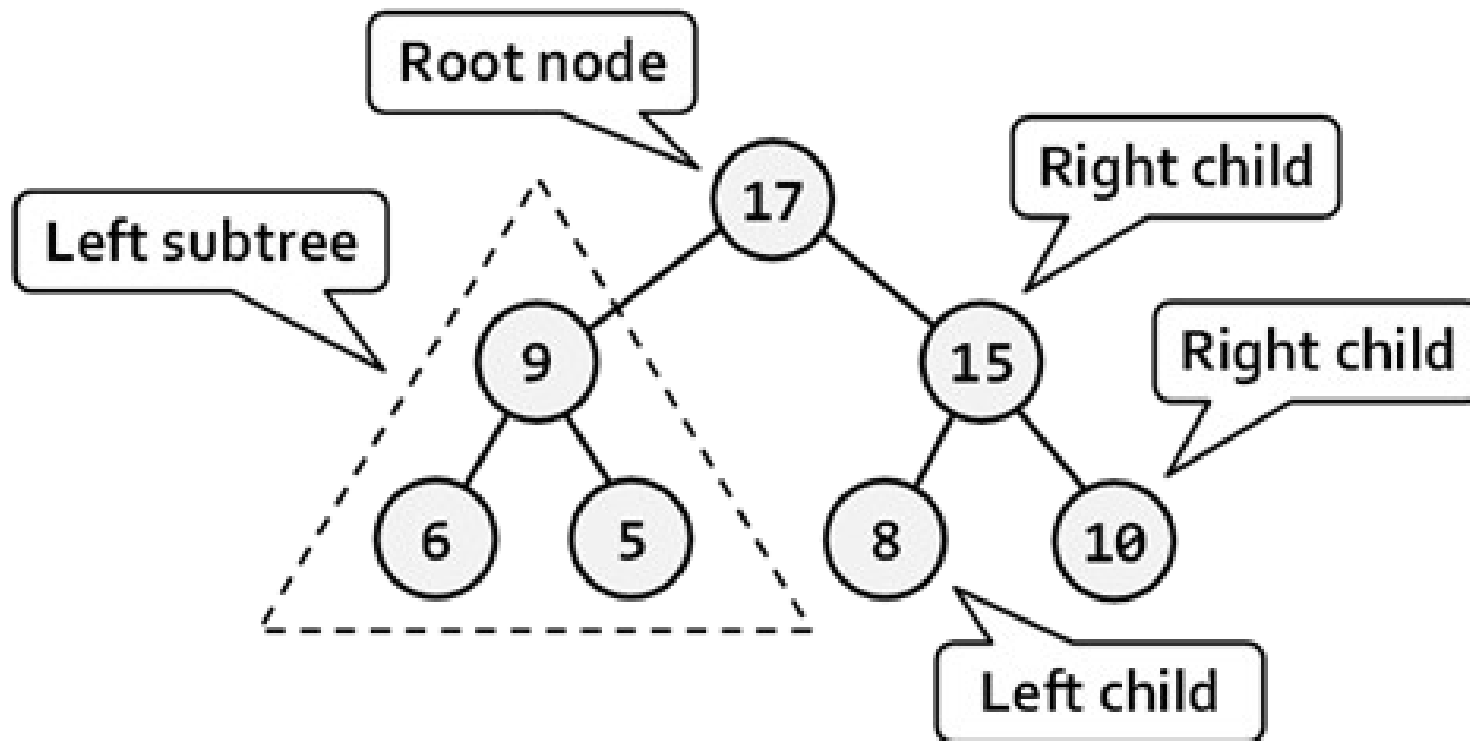


Definisjon

- I et binært tre har hver node enten 0, 1 eller 2 barn
- Rekursiv definisjon:
 - Et binært tre er enten tomt, eller:
 - Består av en rotnode og to binære trær som kalles *venstre* subtre og *høyre* subtre til roten




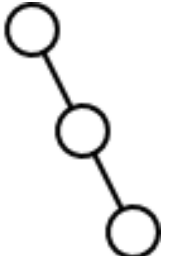
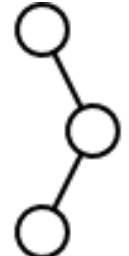


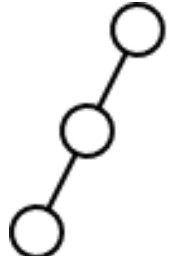
Venstre-høyre ordning i binære trær



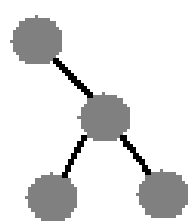
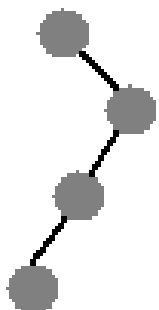
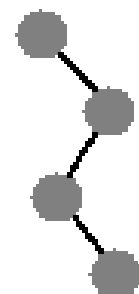
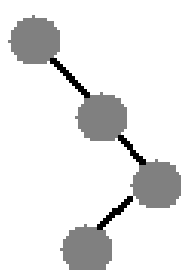
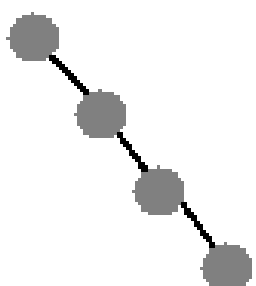
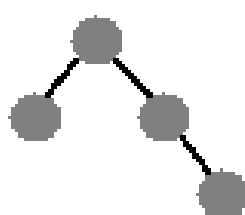
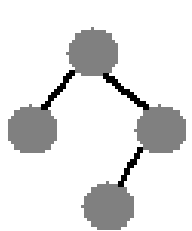
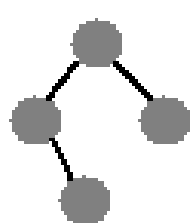
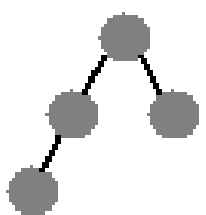
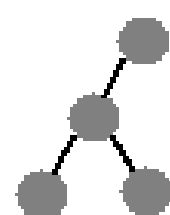
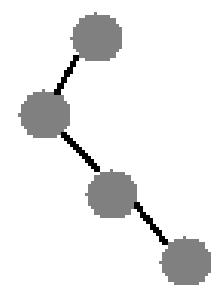
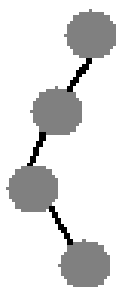
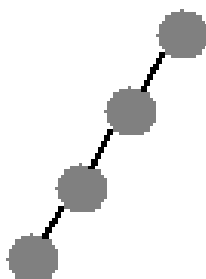
Alle binære trær med $n = 1, 2$ og 3 noder

 $n = 1$

 $n = 2$

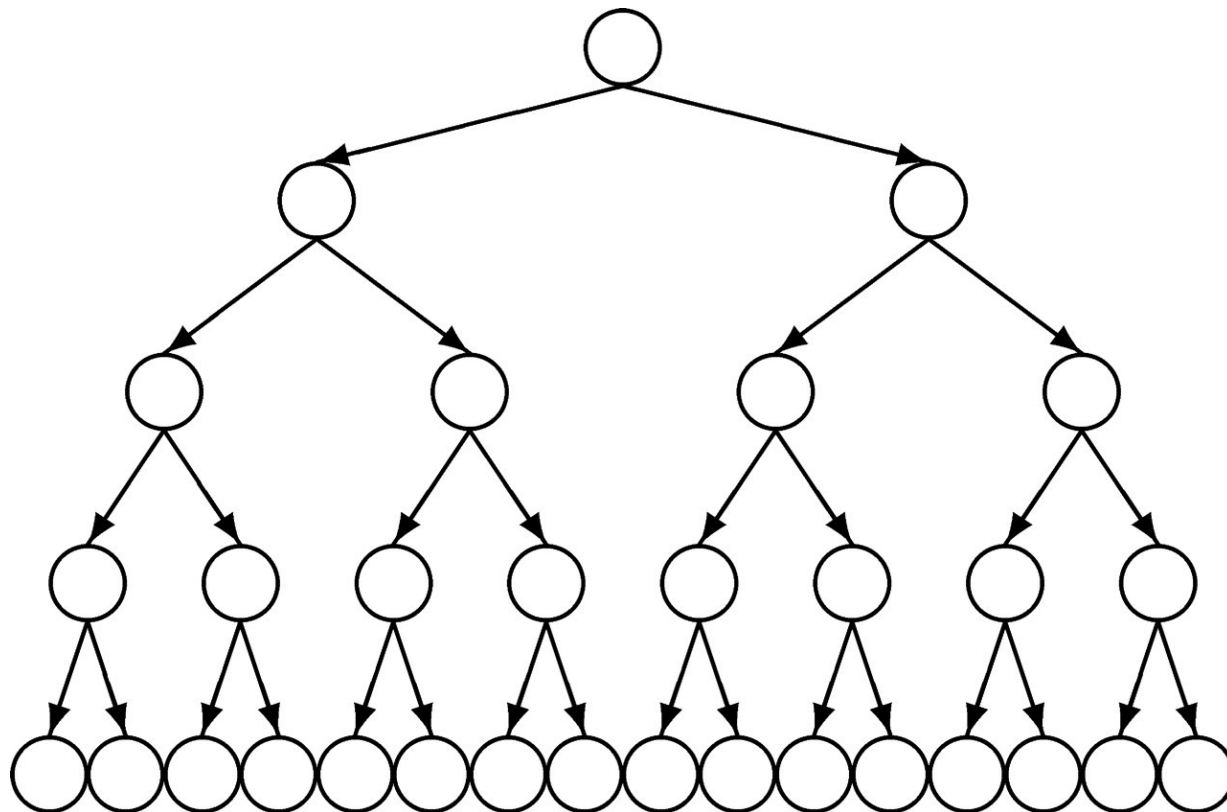
 $n = 3$

Oppgave: Tegn alle de 14 forskjellige binære trærne som kan lages med $n = 4$ noder



Fulle binære trær

- Et binært tre er *fullt* hvis det har maksimalt antall noder – alle nivåene i treet er helt fylt opp med noder
- Fullt binært tre med høyde $h = 4$:



Hvor mange noder er det i et fullt binært tre med høyde h ?

- Nivå 0: $1 = 2^0$ node
 - Nivå 1: $2 = 2^1$ noder, totalt $1 + 2 = 3$
 - Nivå 2: $4 = 2^2$ noder, totalt $1 + 2 + 4 = 7$
 - Nivå 3: $8 = 2^3$ noder, totalt $1 + 2 + 4 + 8 = 15$
 - \vdots
 - Nivå h : 2^h noder
-
- Totalt: $1 + 2 + 4 + 8 + 16 + \dots + 2^h = 2^{h+1} - 1$ noder
-

Hva er høyden til et fullt binært tre, uttrykt som funksjon av antall noder?

- n : Antall noder i et fullt binært tre
- h : Høyden av treet
- h som funksjon av n :

$$n = 2^{h+1} - 1$$

$$n + 1 = 2^{h+1}$$

$$\log_2(n + 1) = h + 1$$

$$h = \log_2(n + 1) - 1 = \lfloor \log_2 n \rfloor$$

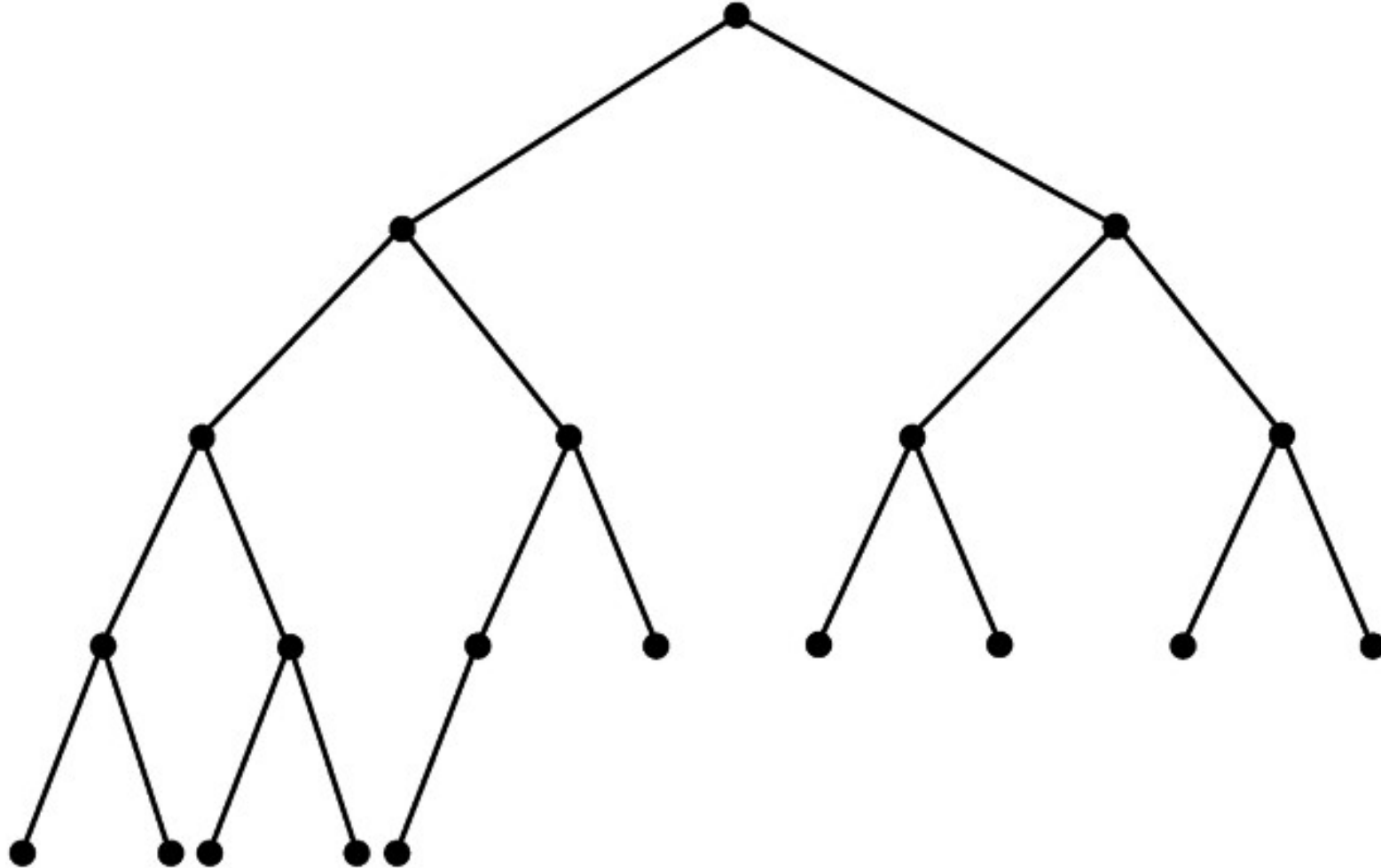
- Høyden h til et fullt binært tre med n noder er lik største heltall mindre eller lik $\log_2 n$

Komplette binære trær

- Et binært tre er *komplett* hvis:
 - Alle nivåene, unntatt muligens det nederste, er helt fulle med noder (2^k noder på hvert nivå k)
 - På det nederste nivået skal alle nodene ligge så langt til venstre i treet som mulig
- Kan enkelt bevises at høyden h for et komplett binært tre med n noder også er:

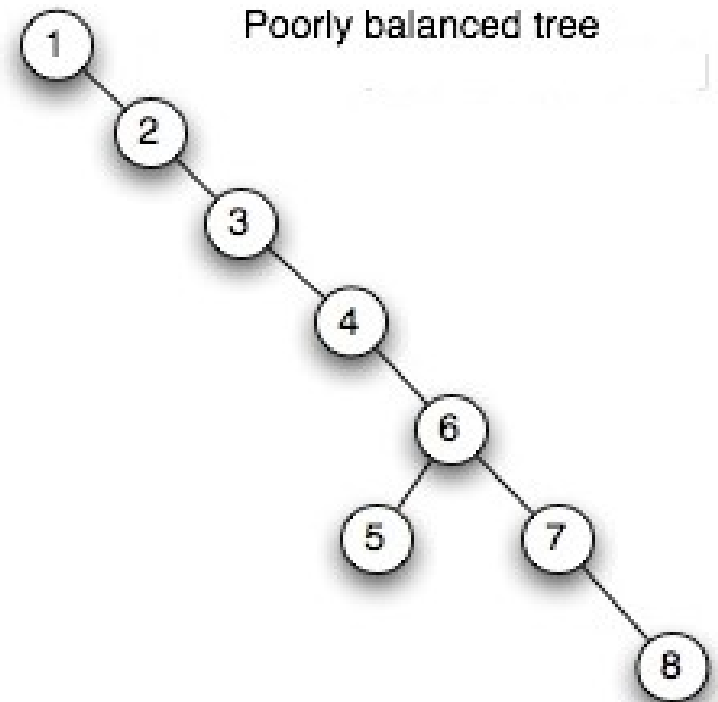
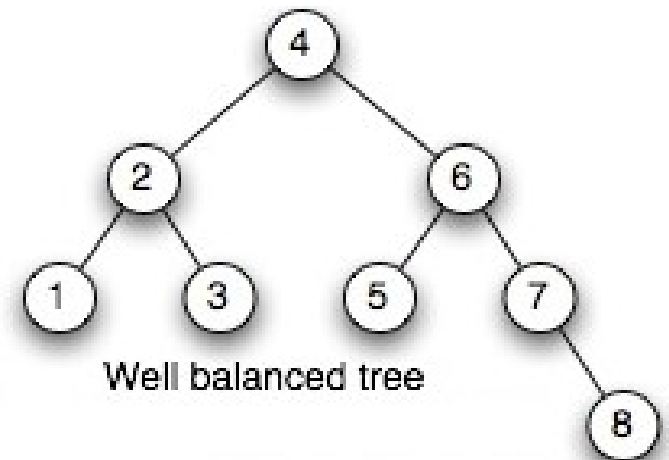
$$h = \lfloor \log_2 n \rfloor$$

Et komplett binært tre



Balanserte binære trær

- Hvis binære trær skal være effektive, må de ikke bli skjeve og ubalanserte
- Definisjon av et balansert binært tre:
 - For alle noder i treet er forskjellen i høyde på nodens venstre og høyre subtre maksimalt lik 1
- Kan vises at høyden av et balansert binært tre med n noder er $O(\log n)$



Balanserte binære trær og effektivitet

- Algoritmer som opererer på binære trær følger ofte en vei gjennom treet fra roten ut til et blad
- Effektivitet og kjøretid for algoritmene blir da proporsjonal med treet's høyde h
- Hvis treet blir ubalansert og degenererer til «nesten lenket liste», er $h = O(n)$ og treet blir ineffektivt
- Hvis vi klarer å holde det binære treet balansert er alltid $h = O(\log n)$, og algoritmene blir svært effektive
- Det finnes flere teknikker for holde et binært tre balansert under innsetting og fjerning av noder

Implementasjoner av binære trær

- Med en array:
 - Rotnoden ligger først i arrayen, på indeks 0
 - Vi *beregner* hvor barna til en node ligger lagret i arrayen, ut i fra foreldernodens indeks
- Med referanser/pekere:
 - Hele treet representeres med en peker til rotnoden
 - Hver node inneholder en peker til venstre barn og en peker til høyre barn

Standard arrayimplementasjon

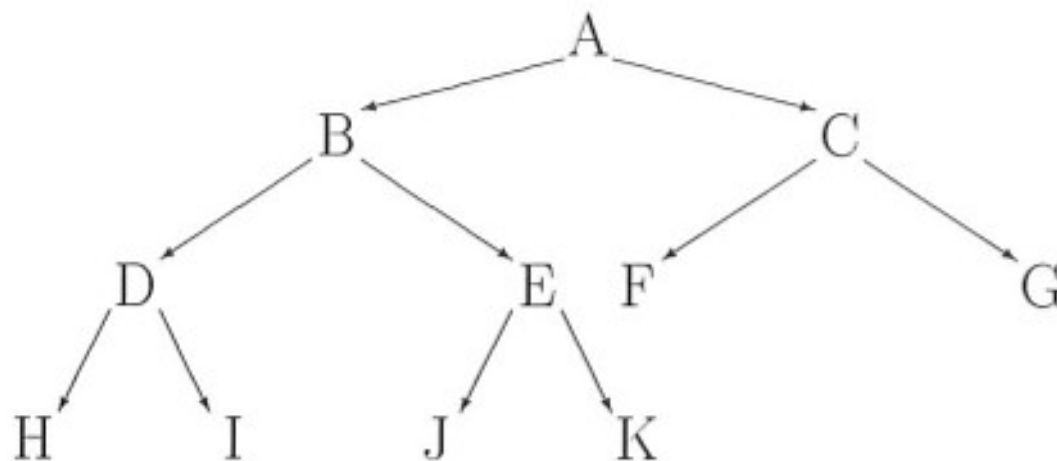
- Rotnoden lagres på indeks 0
- Venstre og høyre barn til noden på indeks i lagres på indeksene:

$$2 \cdot i + 1 \quad \text{og} \quad 2 \cdot i + 2$$

- Effektivt for (nesten) komplette binære trær
- Ineffektivt for ubalanserte trær som mangler mye på å være komplette – medfører lange arraysegmenter som blir stående ubrukte

Arrayimplementasjon, komplett tre

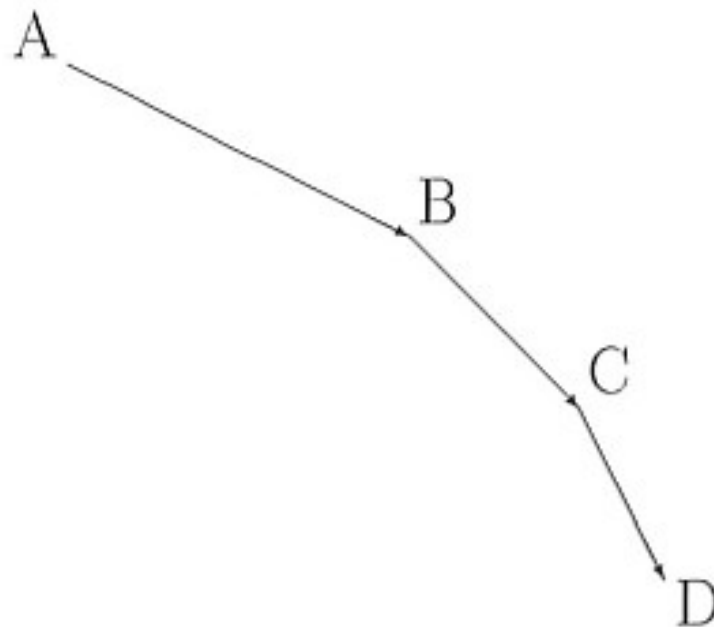
- Treet pakkes effektivt inn i arrayen, fra venstre mot høyre og fra roten og nedover



0	1	2	3	4	5	6	7	8	9	10
A	B	C	D	E	F	G	H	I	J	K

Arrayimplementasjon, ubalansert tre

- Sløser mye med plass fordi nivåene nesten er tomme



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	ε	B	ε	ε	ε	C	ε	ε	ε	ε	ε	ε	ε	D

Implementasjon av binære trær med pekere/referanser

- Tilgang til hele treet gjennom en peker til rotnoden
- Hver node inneholder:
 - (Referanse til) dataene som skal lagres
 - Peker til venstre subtre/barn
 - Peker til høyre subtre/barn
 - Evt. andre ting som måtte trengs (peker til forelder-node, antall noder i subtrærne, høyde, nivå...)

Binære trær med pekere/referanser:

Fordeler og ulemper

- Fordeler:
 - «Naturlig» løsning, enkel å forstå
 - Egner seg godt for rekursive algoritmer
 - Allokerer bare plass til de nodene i treet som faktisk brukes
- Ulemper:
 - Ikke like raskt som array, hvis trærne er komplette
 - Mer overhead til dynamisk memoryhåndtering

Forenklinger i forhold til læreboken

- Læreboken implementerer binære trær med:
 - Generisk ADT som kan lagre 'alt'
 - Java-iteratører
- For å holde fokus på prinsippene og algoritmene, og ikke på Java, forenkler vi til:
 - Bare implementasjon med pekere
 - Ingen ADT, «skreddersyr» i stedet noder og algoritmer for hvert enkelt eksempel
 - Noder som bare inneholder enkle data (heltall, tegn, strenger) i tillegg til pekere til venstre og høyre barn
 - Se eksempel neste side →

```
class treNode
{
    String ord;
    int frekvens;
    node venstre, høyre;

    public treNode(String S)
    {
        ord = S;
        frekvens = 1;
        venstre = høyre = null;
    }
}
```

```
class Oblig5BinTre
{
    treNode rot;

    public Oblig5BinTre()
    {
        rot = null;
    }
}
```

