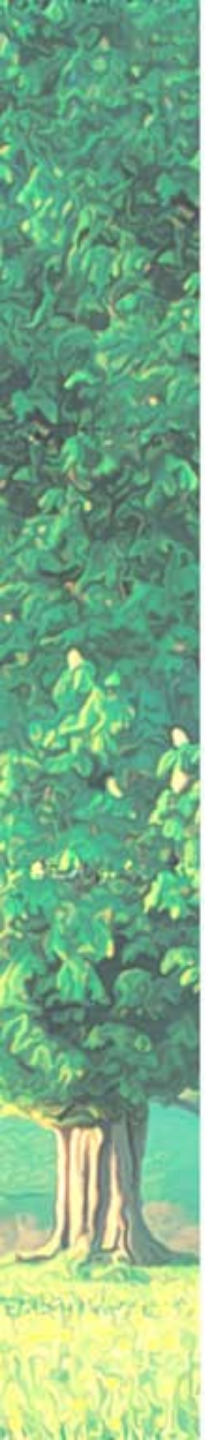
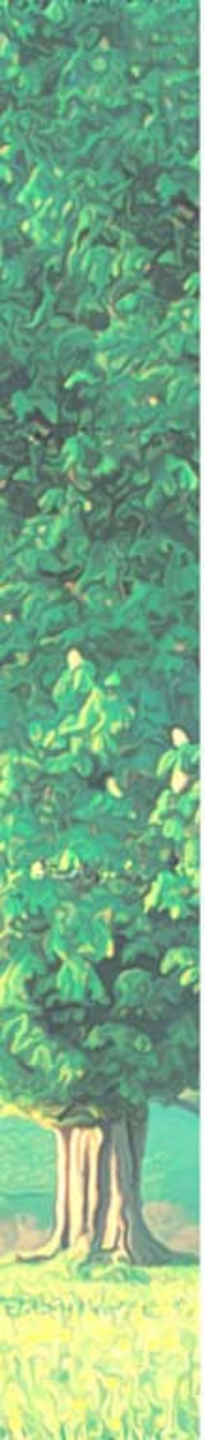


Fra Kap.10 Binære søketre (BS-tre)

Sist oppdatert 20.03.10

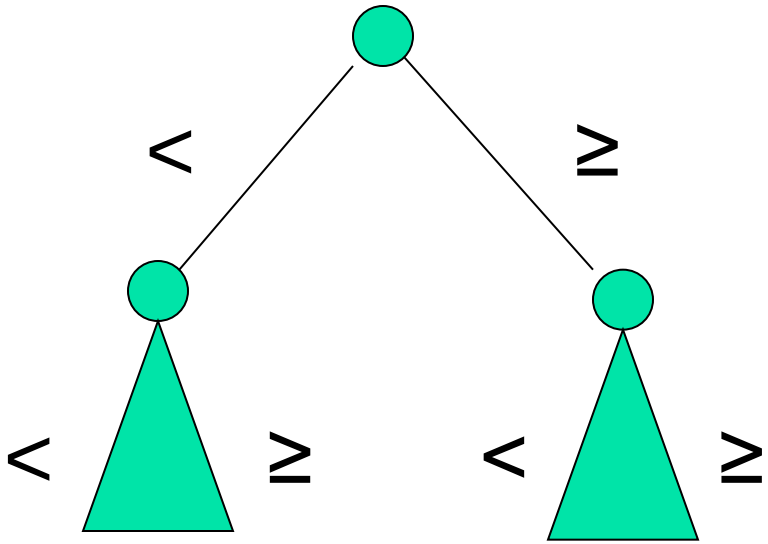
- **Definere en abstrakt datastruktur *binært søketre.***
- **Vise hvordan binær søketre kan brukes til å løse problemer.**
- **Undersøke ulike implementasjoner for binært søketetre.**
- **Sammenligne ulike implementasjoner.**



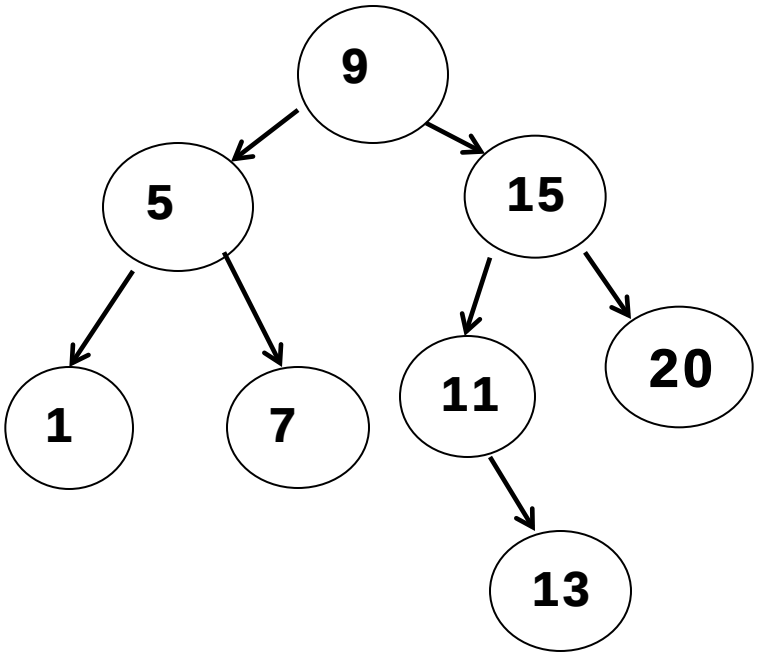
Binær søketre

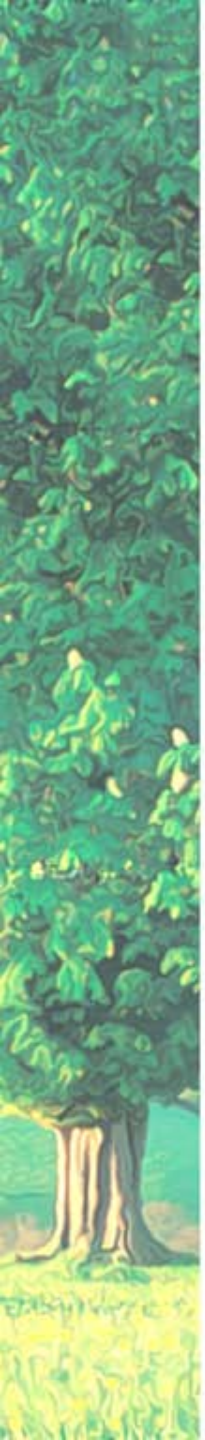
- **Et binært søketre er et binært tre med tilleggsegenskapen: For hver node x (unntatt bladene) er alle elementene i venstreundertre til x mindre enn elementet til x og alle elementene i høyre undertre til x er større eller lik elementet til x.**

Binær søketre



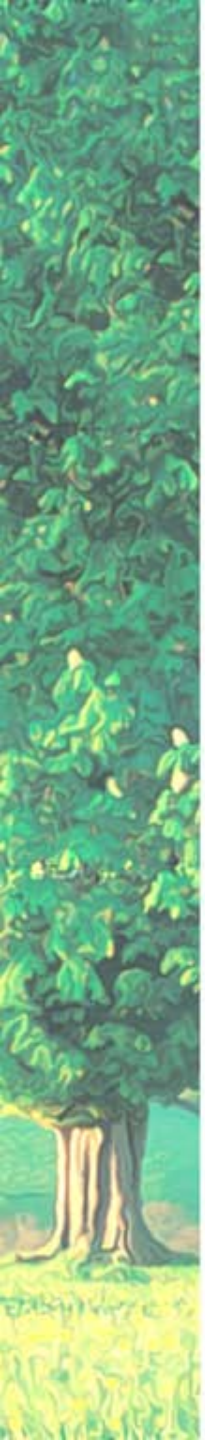
Binær søketre





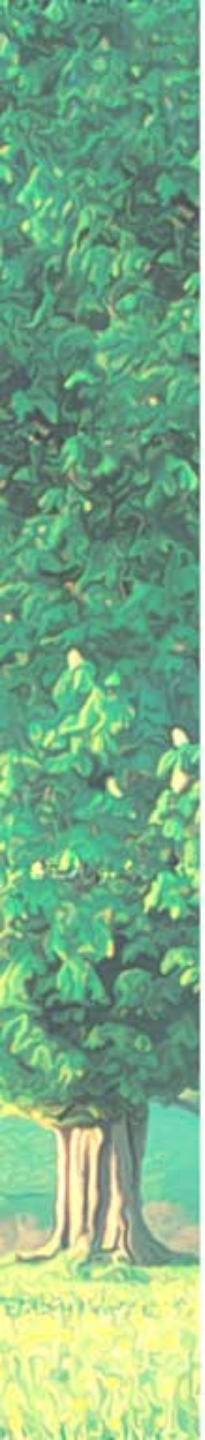
Fordeler

- **Effektiv søking dersom treet er brukbart balansert (tilsvarer binærsøk i sortert tabell)**
- **Effektiv innsetting og sletting (sparer parallellforskyvingen som vi må ha i en sortert tabell dersom vi har vanlig implementering med noder og pekere.**



Operasjoner for et bs-tre

- **leggTilElement** – legger til et element
- **fjern** - fjerne et element hvis det fins
  - prinsippet, eget ark
- **fjernMin** - fjerne det minste elementet
- **fjernMaks** - fjerne det største elementet
- **finn** - finner et element hvis det fins
- **inOrden –** systematisk gjennomgang (får sortert)
- **antall og tom**



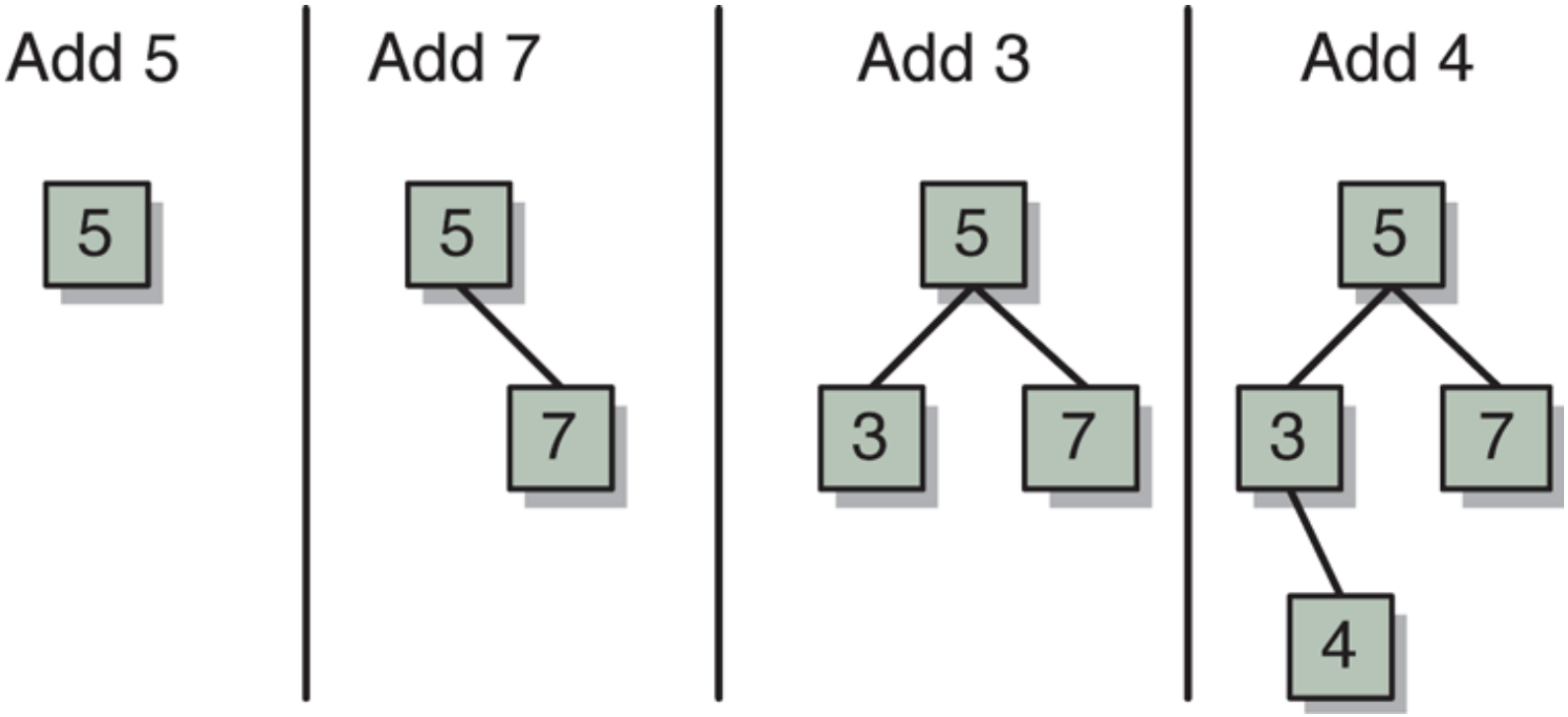
Implementere bs-tre med kjeding

- **Vi kan lage en klasse *KjedetBinærSøkeTre*. *Se utdelt kode***

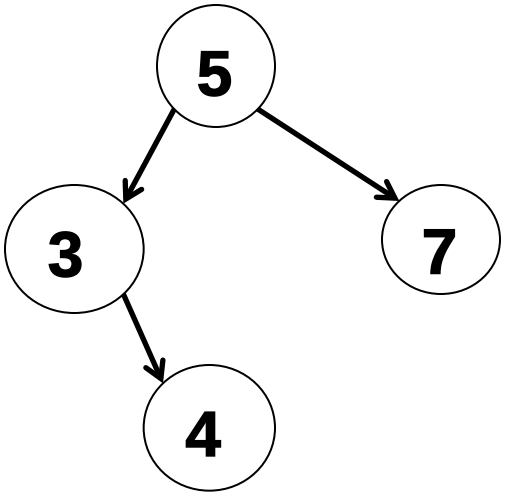
- **Denne klassen vil ha to konstruktører, en som oppretter et tomt tre og en som oppretter et tre med en node.**

Binært søketre

- ***leggTil-metoden*** **finner den passende lokasjonen for et gitt element og legger det til treet som et blad.**



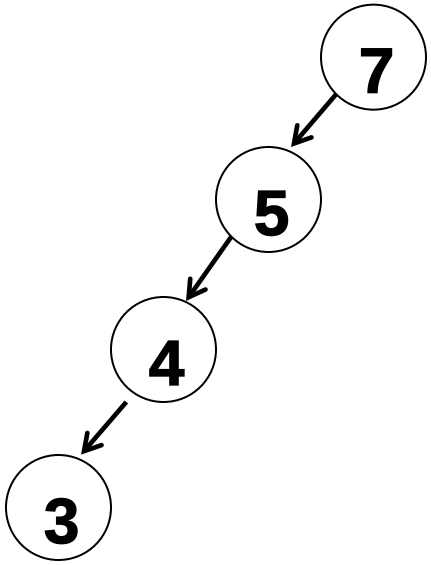
Inorden gjennomgang av et BS-tre

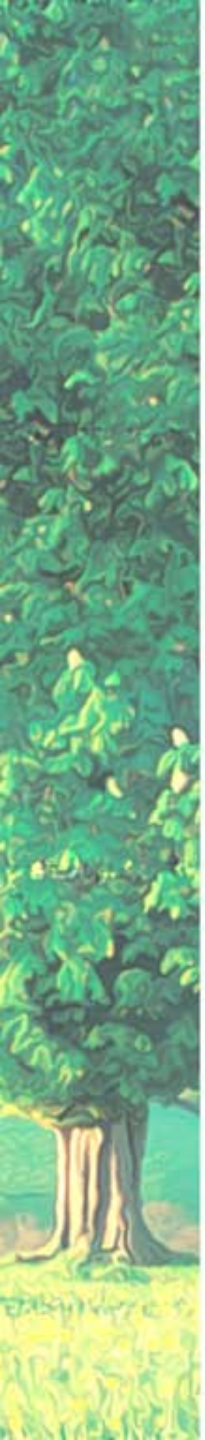


Inorden gjennomgang av et bs-tre vil gi en ordnet sekvens av elementene i stigende rekkefølge: 3, 4, 5, 7

Trestrukturen i et BS-tre avhengig av inndatarekkefølgen!

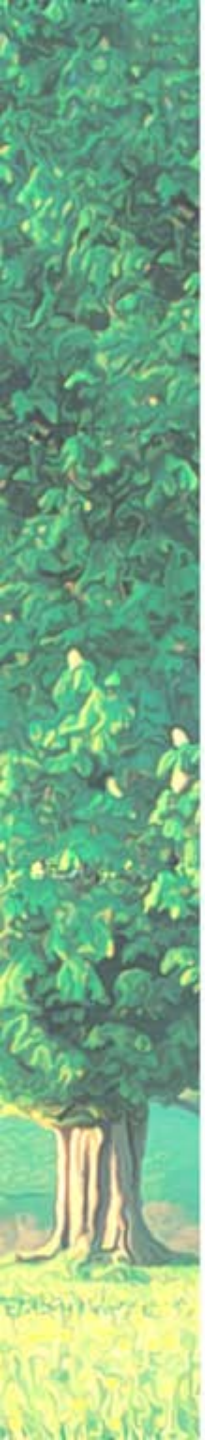
- **Samme data som i sted, men inndatarekkefølgen er nå 7, 5, 4, 3**





Fjerne et element se eget ark

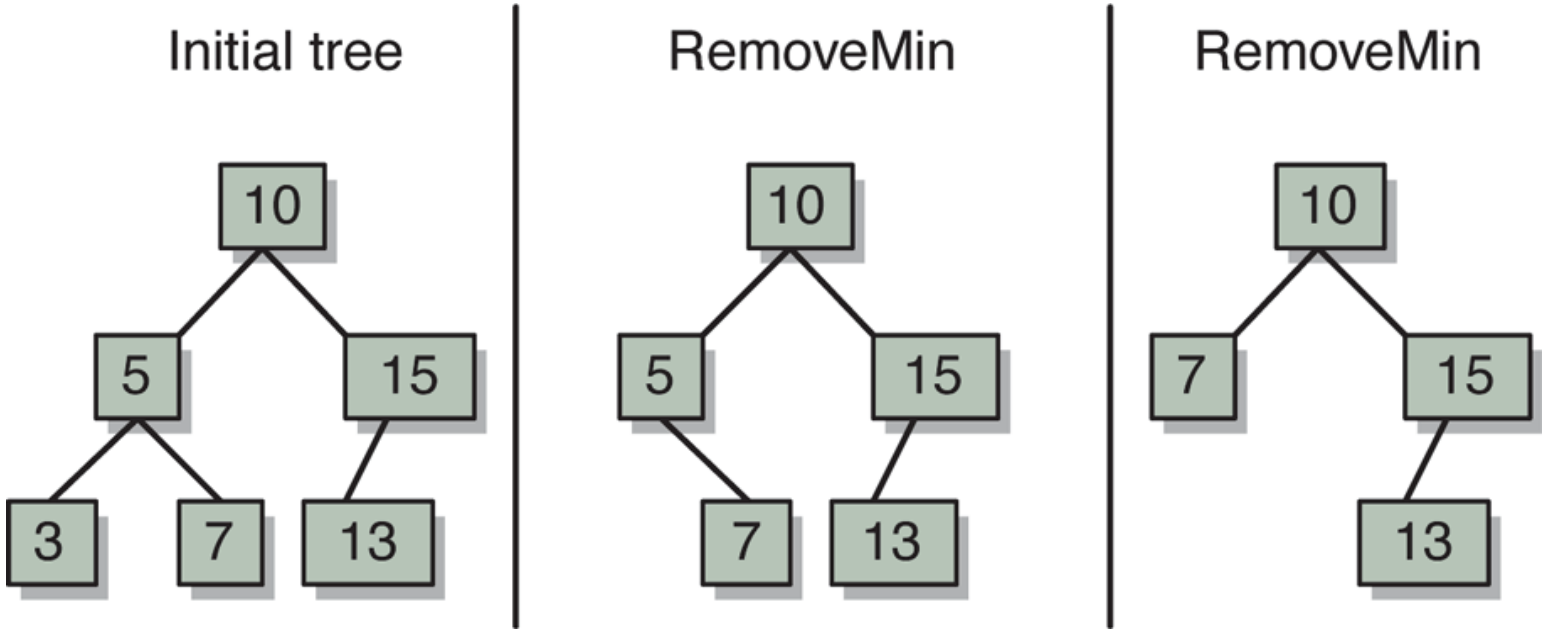
- **Fjerne et element fra et binært søketre krever tre steg:**
  - **Finne elementet**
  - **Hvis noden ikke er et blad, så *erstatt* det med dens inorden etterfølger. (største elementet i nodens venstre undertre, eller minste elementet i nodens høyre undertre)**
  - **Returner elementet som er fjernet**

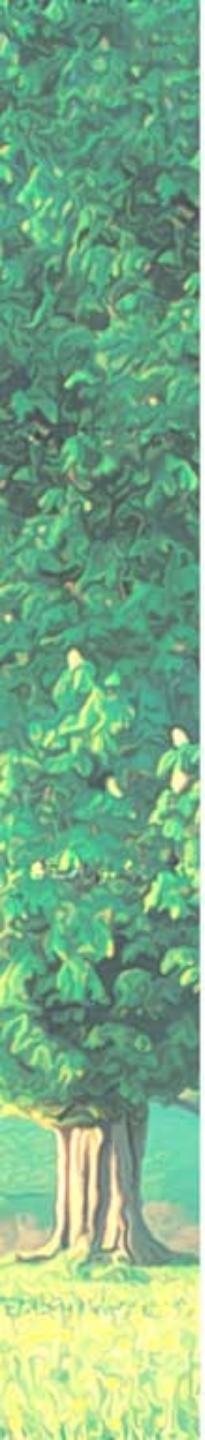


Fjerne det minste elementet

- **Det er tre tilfeller for lokasjonen til det minste elementet i et binært søketre.**
  - **Hvis roten ikke har venstrebarn, så er roten det minste elementet og høyre barn til roten blir den nye roten.**
  - **Hvis noden lengst mot venstre er et blad, så settes foreldrens venstrebarn til null.**
  - **Hvis noden lengst til venstre er en intern node, så settes foreldrens venstrebarn til å peke på nodens høyrebarn.**

FIGUR 10.5 Fjerne det minste elementet



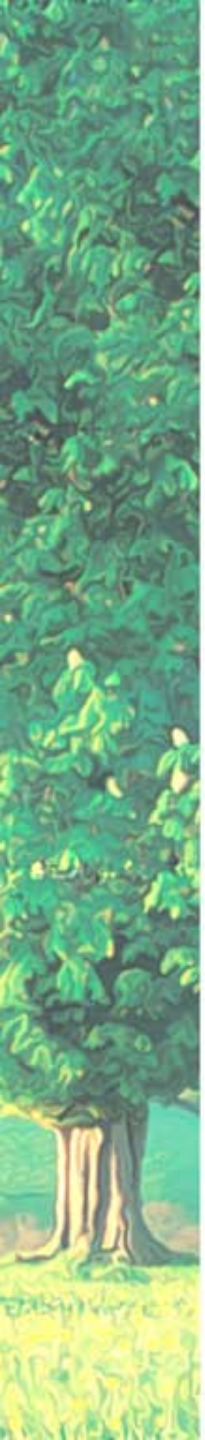


Binært søketre som ordnet liste

- **En effektiv implementasjon av en ordnet liste dersom treet er balansert.**

- **class OrdnaListeBST{….}**

- **Operasjoner som innsetting, søking og sletting vil være av O(logn) dersom treet er balansert. Disse operasjonene vil være mindre effektive , O(n), i kjedet lineær struktur.**



Binært søketre som ordnet liste

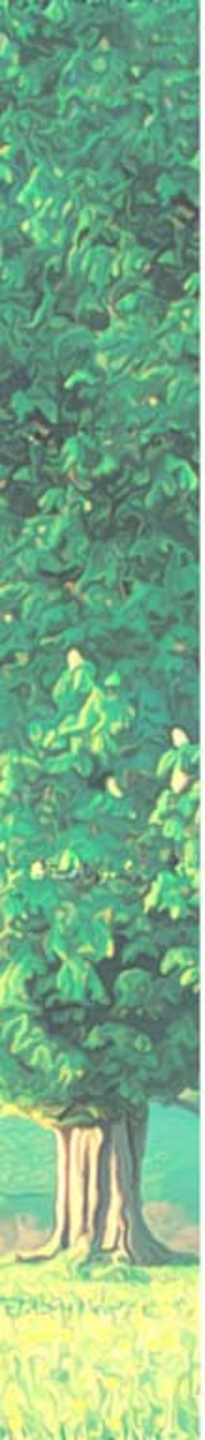
Operasjoner i ordnet liste

- fjernFørste(), fjernSiste(), siste(), første()
- fjern(el), finn(el)
- tom(), antal()
- leggTil(el)

Alternativ 1 Lærebok Listing 10.2
Bruker arv og lager klassen **OrdnaListeBST** som arver fra klassen **BSTre** (super)
Eks: public T fjernFørste(){ return fjernMin() }

Alternativ 2: Lager ny, selvstendig klasse **OrdnaListeBST**

- Data i et OrdnaListeBST-objekt:
  Et objekt av klassen BSTre, aggregering.
  Se utdelt kode.

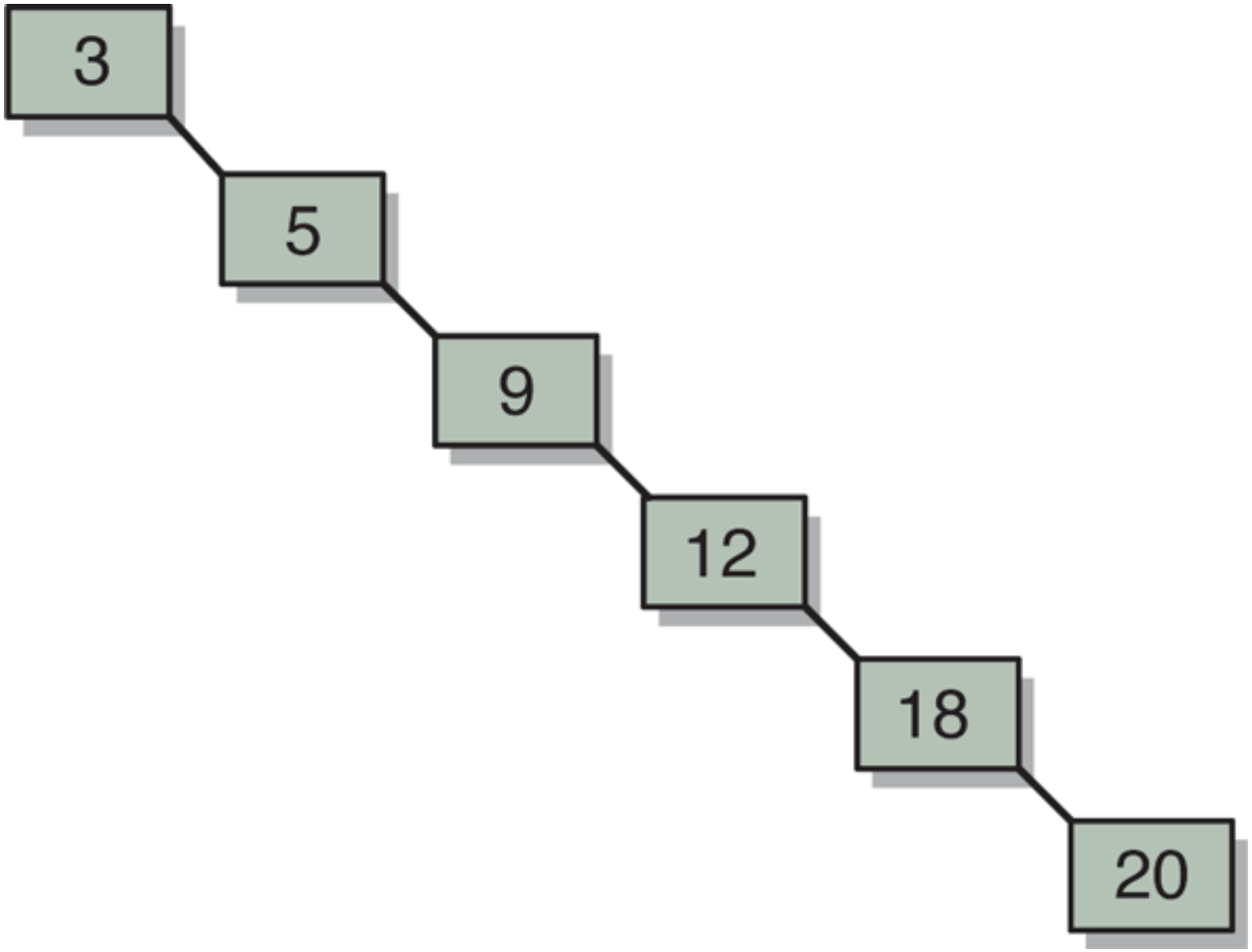


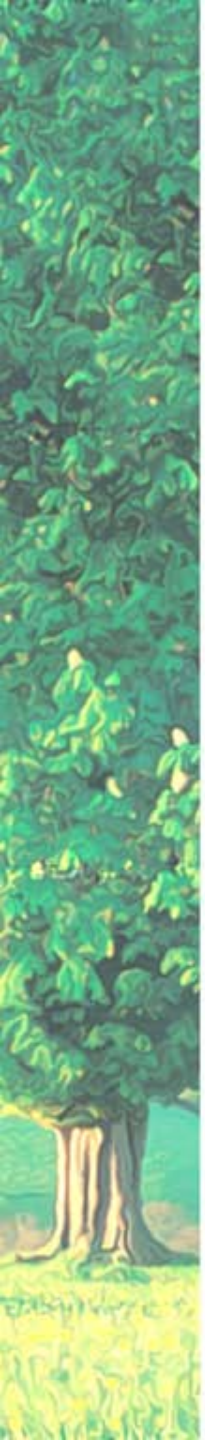
Balansert binært søketre

- **Hvorfor er balansering av søktre så avgjørende?**
- **La oss se på hva som skjer når vi setter inn følgende tall uten å balansere:**

  3, 5, 9, 12, 18, 20

FIGUR 10.9
Et helt ubalansert binært søketre

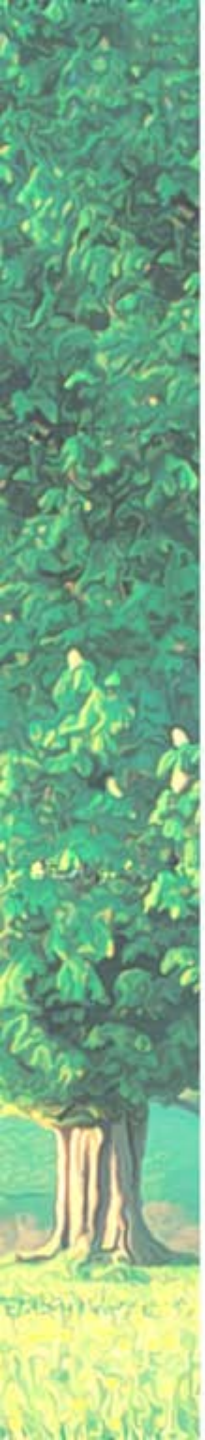




Degenerert binært søketre

- **Et slikt helt ubalansert søketere er kalt et degenerert binært søketre.**

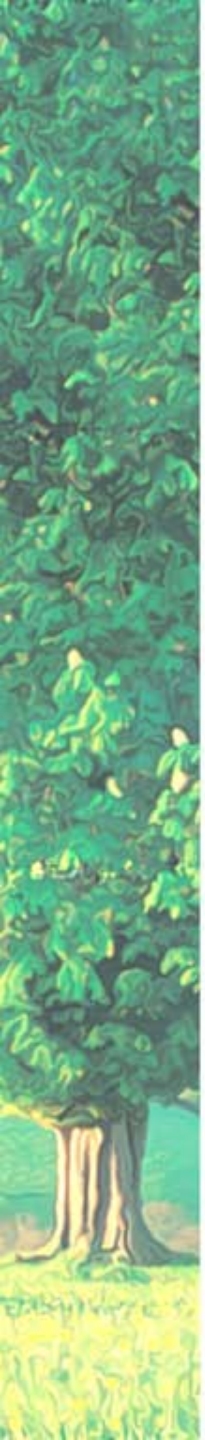
- **Degenererte binære søketre har høyde lik ≈ n og søkingen blir da O(n) som er minst effektiv.**
- **For et balansert søketre er høyden ≈ $\log_2 n$ og søkingen blir da O(logn) som er mest effektiv.**



Balansering av binært søketre

Metode 1)

- **Gjennomgå treet i inorden og skrive til tabell. Får da ut en ordnet liste.**
- **Lager så et nytt tre ved rekursiv innsetting.**
  - **Velg midtelementet som rot.**
  - **Sette inn venstre undertre rekursivt.**
  - **Sette inn høyre undertre rekursivt.**



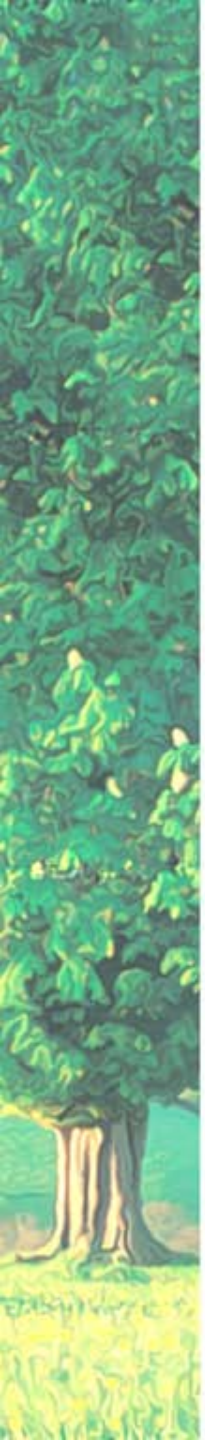
Balansering av binært søketre

Metode 2)

En mindre arbeidskrevende metode er å bruke *rotasjoner*.

Vi ser her på enten venstrerotasjon eller høyrerotasjon.

Noen ganger må en kombinere både venstre- og høyrerotasjon for å oppnå balansering.



Balansering av binært søketre

Høyrerotasjon vil løse en ubalanse når det fins en lang sti i venstre undertre til venstrebarn(nodeC) til roten (nodeN). nodeC skal bli ny rot.

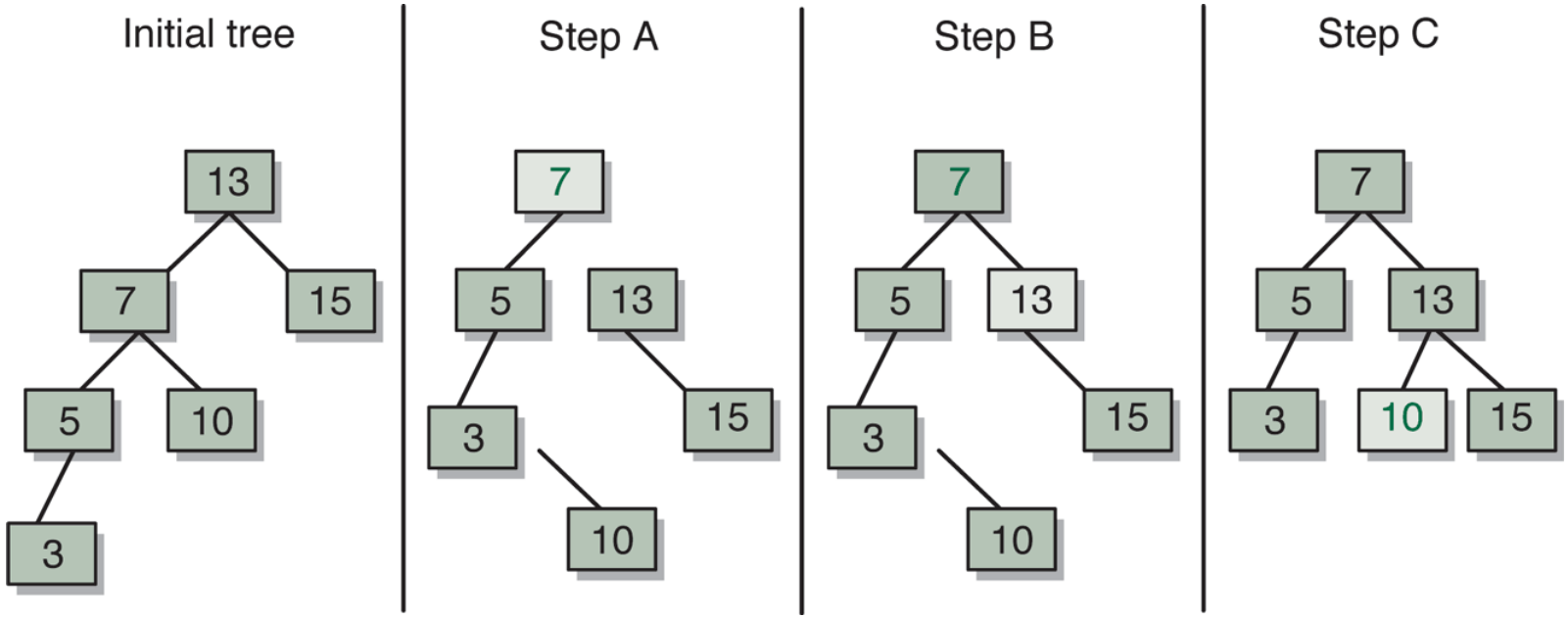
Algortitme roterHøyre(nodeN)

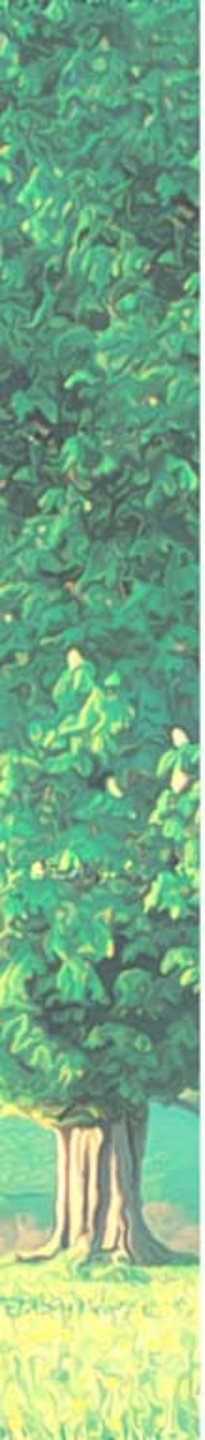
nodeC = venstrebarn til nodeN

Sett nodeN sitt venstrebarn til nodeC sitt høyrebarn

Sett nodeC sitt høyrebarn til nodeN

FIGUR 10.10 Ubalansert tre og balansert tre etter en høyrerotasjon





Balansering av binært søketre

- **Venstrerotasjon vil løse en ubalanse hvis det fins en lang sti i høyre undertre til høyrebarn (nodeC) til roten(nodeN). nodeC skal bli ny rot.**

Algoritme roterVenstre(nodeN)

nodeC = høyrebarn til nodeN

Sett nodeN sitt høyrebarn til nodeC sitt venstrebarn

Sett nodeC sitt venstrebarn til nodeN

FIGUR 10.11 Ubalansert tre og balansert tre etter en venstrerotasjon.