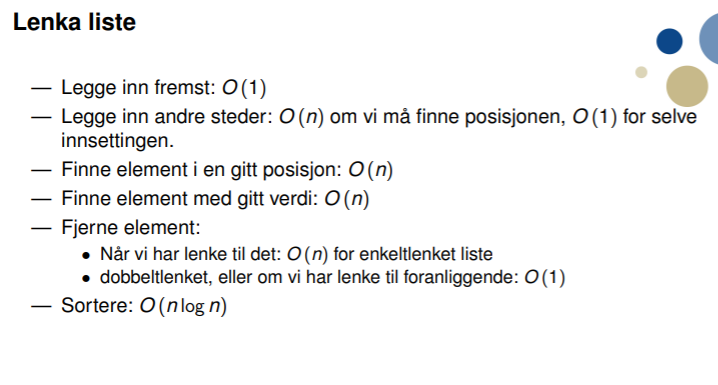
Algoritmer og Datastrukturer #3

Lister, kø og stakk + Trær

Datatypen liste

* Plass til mange elementer, som kan settes inn, fjernes og søke etter.
* Implementasjoner: *Object[], ArrayList.*
* Liste som usortert tabell:
  + Smarte implementasjoner av liste er *O*(1) i gjennomsnitt, dette oppnås ved at tabellstørrelsen *dobles* når vi utvider.
* Liste som sortert tabell:
  + Legge til: *O*(n), må flytte elementer
  + Fjerne *O*(n), må flytte elementer
* Tabellimplementasjoner - Tabeller er gode på oppslag
  + Finne element på en gitt posisjon: *O*(1)
  + Finne element med gitt verdi:
    - Sortert *O*(log n)
    - Usortert *O*(n)
  + Sortere: *O*(*n* log *n*)
  + Fjerne et gitt element:
    - Usortert *O*(1)
    - Sortert *O*(n)

Fordeler og ulemper:

* Tabell
  + Bruker ikke plass på neste/forrige referanser.
  + Direkte oppslag.
* Lenka lister
  + Når innsetting/uttak skjer andre steder enn bakerst.
  + Når antall elementer varierer mye.
  + Når vi har bruk for å flytte elementer eller hele kjeder.

Sammenligning av implementasjoner (kø)

* Tabell
  + Begrenset antall som kan legges inn
  + Må programmere sirkulær tabell selv
* Lenka liste
  + Lenkene bruker ekstra plass
  + LinkedList tilgjengelig

Stakk

* Lineær datastruktur
* Innsetting og uttak skjer øverst på stakken
* Implementasjoner
  + Tabell
  + Lenket liste

Trær

Datatypen tre

* Ulineær datastruktur
* Har ei rot, tilsvarende hodet i ei lenka liste
* Nodene kan lenke til *mer* enn en barnenode
* Alle utenom rota har foreldrenode
  + Noder uten barn: ytre noder / løvnoder
  + Noder med barn: indre noder

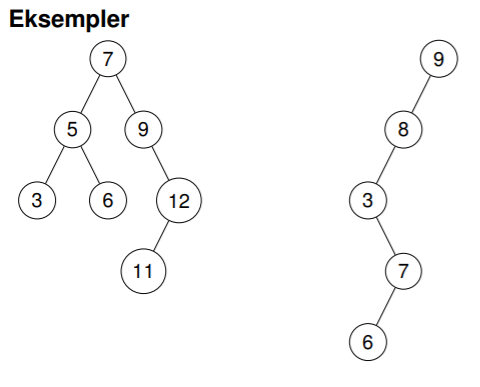
Binærtrær

* Hver node har maks. to barn
* Barnas rekkefølge er vesentlig – ordnet
* Venstre og høyre barn og subtre
* **Dybde** - antall lenker mellom noden og rota.
  + Dybden kan finnes ved å følge foreldrelenkene opp til rota, dette kan gjøres med en enkel løkke.
    - Hvis nodene ikke har foreldrelenke 🡪 Kan risikere å måtte sjekke alle noder, gjøres enklest med en rekursiv algoritme
* **Høyde –** antall lenker mellom noden og den noden som er lengst unna i et av nodens subtrær. Treets høyde er definert som rotas høyde.
  + Høyden kan finnes kan gjøres ved en rekursiv algoritme:
    - Finn høyden til venstre subtre
    - Finn høyden til høyre subtre
    - Velg den største av de to høydene og legg til én

Binært søketre

* Anvendelse: databaseindex med søk på større, mindre eller lik.
* Hver node har en nøkkel
  + Alle noder i venstre subtre har mindre nøkkel
  + Alle noder i høyre subtre har større nøkkel
* Operasjoner: Sette inn, søke, slette
* Sletting i binærtrær kan være komplisert

Kjøretider:

* Innsetting, søk og sletting går nedover i treet, dermed er kjøretiden proporsjonal med høyden på treet.
* Ideelt er treet perfekt balansert, og da vil høyden være log2 n, der n er antall noder i treet.
  + Verste fall 🡪 treet formet som lenket liste. Høyde og kjøretider blir da *n*.