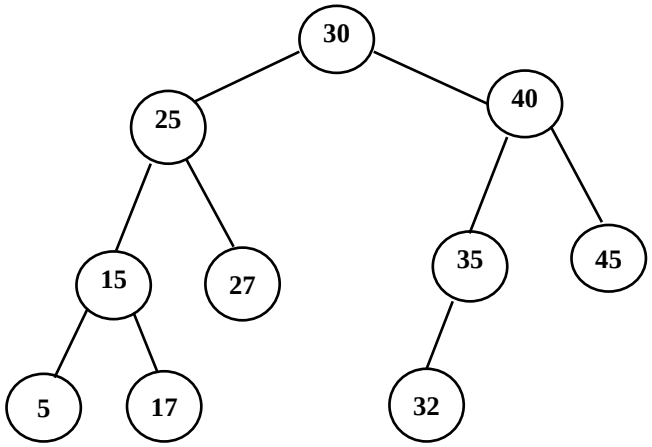
# DAT102 – Våren 2025

**Øvingsoppgaver uke 15 (7. april – 11. april)**

## Oppgave 1(ikke innlevering)

Figuren under viser et binært søketre (bs-tre):




a) Sett inn elementet 23 og deretter elementet 19. Tegn opp bs-treet vi får etter at begge elementene er satt inn.

b) Fjern rotelementet 30 fra det **opprinnelige** bs-treet. Tegn opp bs-treet vi får etter fjerning av 30.

## Oppgave 2 (obligatorisk)

Om vi går gjennom et binært søketre i inorden og skriver ut verdiene, kommer de ut sortert. Metoden under skriver ut verdier i dette BS-treet mellom to grenseverdier.

```
public void skrivVerdier(T nedre, T ovre) {
      skrivVerdierRek(rot, nedre, ovre);
}

private void skrivVerdierRek(BinaerTreNode<T> t, T min, T maks) {
      if (t != null) {
            skrivVerdierRek(t.getVenstre(), min, maks);
            if ((t.getElement().compareTo(min) >= 0) &&
              (t.getElement().compareTo(maks) <= 0)) {
                  System.out.print(t.getElement() + " ");
            }
            skrivVerdierRek(t.getHogre(), min, maks);
      }
}
```

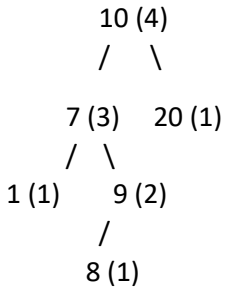
a) Forklar hvorfor metdoen `skrivVerdierRek` ikke er optimal. Ta metoden med i klassen `BS_Tre (F22)` og prøv den ut.

b) Modifiser metoden litt slik at den blir mer effektiv (bruker færre kall til compareTo. Ta metoden med i klassen `BS_Tre` og prøv den ut.

   Tips: Du kan godt bruke flere hjelpemetoder.

## Oppgave 3 (obligatorisk)

For at operasjonene inneholder, finn, leggtil og fjern på et binært søketre skal være effektive, må treet være balansert. For effektivt å kunne holde treet balansert etter innsetting og sletting, legger vi til en ekstra objektvariabel, hogdeU, som angir høyden på undertreet som har noden som rot. Grunnen til vi skriver hog`deU` er at vi allerede har en metode `getHoyde()` som finner høyden på hele treet. Det vil si: `getHogdeU()` returnerer bare verdien som er lagret i noden. Eksempel der høyden (hogdeU) står i parentes.




a) Modifiser BinaerTreNode ved å legge til objektvariabelen, `hogdeU` med tilhørende get-og set-metode, som angir høyden på undertreet der noden er rot. Modifiser også konstruktøren som oppretter et binært søketre med en node.

b) Lag en metode, `erBalansert()`, som sjekker om et binært tre er balansert. Her kan du anta at høyde-variablen ovenfor er korrekt. Et tre er balansert hvis vi for **alle** noder har at forskjellen mellom høyden av venstre undertre og høyden av høyre undertre er maksimalt 1.

   Tips for å skrive kode: Når vi har lagt til høyde som objektvariabel, kan vi finne høyden på venstre undertre som `p.getVenstre().getHogdeU()`.

c) **Frivillig**. Modifiser leggTil-metoden slik at den oppdaterer objektvariabelen, hogdeU, for de nodene det kan være aktuelt

   Tips. Bruk figurer.

## Oppgave 4 (ikke innlevering)

a) Definer fullt binært tre og komplett binært tre.

b) Tabeller kan brukes for å implementere trær. Tegn de to binære trærne som også kan illustreres ved hjelp av følgende tabeller (roten i posisjon 1):

i)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | i | E | n | s | t | O | r | b | å | t |

ii)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | i | e | N | | s | | t | | | O | r | | | b | å | | | | | t |

c) Vis rekkefølge ved gjennomgang i nivå-orden for de to trærne dere har tegnet under oppgaven over.

## Oppgave 5 (obligatorisk)

a) En spesiell type binære tre er kalt en haug ("heap"). Definer hva som menes med en haug. (Tips få fram de to vesentlige egenskapene definisjonen.)

b) I minst en av de tre tabellene under er elementene ordnet slik at de danner en **makshaug** (node >= venstrebarn og høgrebarn). Vi bruker ikke plass 0 i tabellen, så roten ligger i posisjon 1 i tabellen. Finn hvilke(n). Begrunn svaret

| a[0] | | | | | | | | | | a[10] |
|---|---|---|---|---|---|---|---|---|---|---|
| | 9 | 15 | 12 | 7 | 4 | 2 | 1 | 6 | 5 | 3 |

| b[0] | | | | | | | | | | b[10] |
|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 12 | 10 | 11 | 2 | 6 | 3 | 4 | 8 | 1 |

| c[0] | | | | | | | | | | c[10] |
|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 10 | 14 | 8 | 7 | 13 | 6 | 2 | 5 | 4 |

c) Elementene i tabellen under utgjør en makshaug.

| d[0] | | | | | | | | | | d[10] |
|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 10 | 14 | 8 | 7 | 13 | 6 | 2 | 5 | 4 |

i. Tegn haugen som tre og vis hvordan treet ser ut når vi først fjerner elementet med verdi 15 og deretter organiserer til en haug (makshaug) (tips: etter fjerning av 15, flyttes først siste element (4) til rotposisjon, osv..

ii. Etter at vi har utført pkt i) setter vi nå et nytt element med verdi 13 inn sist i d og organiserer igjen til en haug. Vis hvordan treet nå ser ut.

d) Forklar kort hvordan prinsippet i pkt c) kan brukes til å sortere elementer

e) Det skal lages en minimumshaug ved å bruke reparerNed (reheap). Du starter på siste interne node (node som ikke er blad og så går du mot venstre helt til roten). Ved start er elementene plassert i en tabell som på figuren under.

| d[0] | | | | | | | | | d[10] |
|---|---|---|---|---|---|---|---|---|---|
| | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Tegn først opp treet med de elementene som her er gitt. Du skal nå lage en minimumshaug. Vis alle overgangene ved å tegne opp treet på nytt etter hvert kall av reparerNed inntil du har utført reparerNed med roten som argument.