LØSNINGSFORSLAG for EKSAMEN i INF110 H'2002:

Løsningsforslag 1.1:
Hvis node `i` har `foreldre j`, så representeres det ved `foreldre[i] = j.`
Hvis node `i` er en rotnode, så representeres det ved `foreldre[i] = -1.`

Innholdet av `foreldre` i eksemplet:
`foreldre = { 6, -1, -1, 6, 2, -1, 8, 2, -1, 7 }`

Løsningsforslag 1.2:
```
public void veiMotRot(int node) {
  System.out.print("Fra node:"+node);
  while (foreldre[node] != -1) {
    node = foreldre[node];
    System.out.print(" til node:"+node);
  }
  System.out.println(" som er rot.");
}
```

Løsningsforslag 1.3:
```
public void veiFraRot(int node) {
  if (foreldre[node] == -1)
    System.out.print("Fra rot:"+node);
  else {
    veiFraRot(foreldre[node]);
    System.out.print("  til node:"+node);
  }
}
```

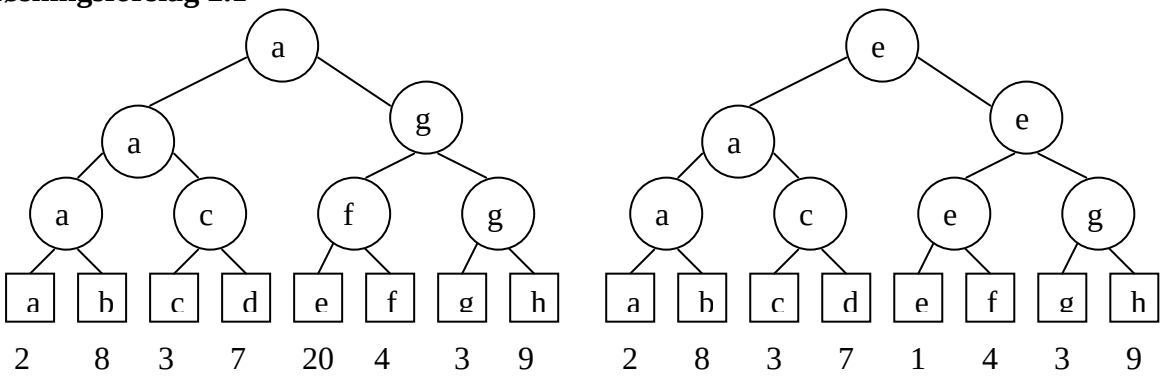
Løsningsforslag 1.4:
Neste tilstand blir:
```
int nyOpp=opp+s-2*i;
```
Restriksjoner på antallet oppGlass:
```
// ma maksimalt antall oppGlass som kan snus
// mi minimalt antall oppGlass som kan snus
int ma = opp<s ? opp : s;
int mi = n-opp<s ? s-(n-opp) : 0;
```

Løsningsforslag 1.5:
```
public class Glass {
  private int n;           // antall glass på bordet
  private int s;           // antall glass som snus i en operasjon
  // Nodene nummereres 0, 1, ... , n
  // hvor nummeret angir antall oppGlass.
  private int [] foreldre; // hvis node sett, indeks til foreldre-node
  private int [] koe;      // kø av sett, men ikke behandlede noder
  private int neste;       // indeks til første i koe
  private int nestefri;    // indeks til første ledige i koe
  // Invarianter:
```

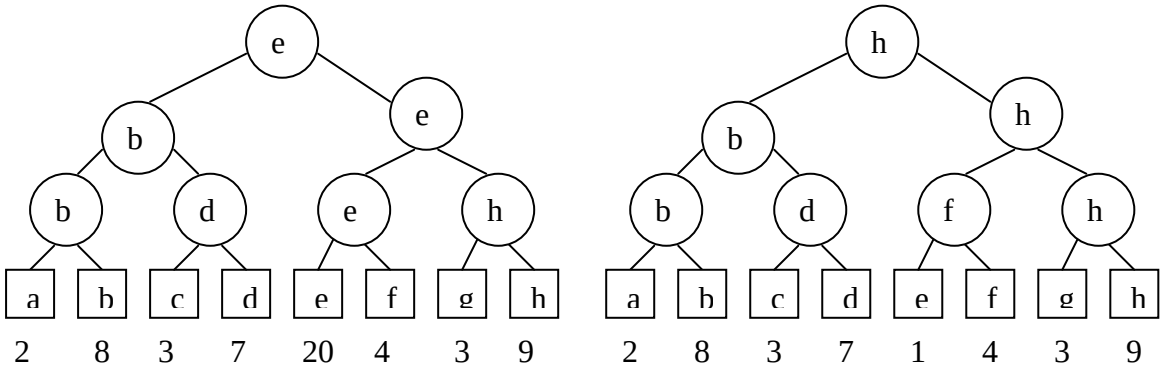
```
  //    0 < s <= n
  //    0 <= neste <= nestefri
  //    neste==nestefri <==> koe tom
  //    foreldre[i]==-1 ==> (i==n || i ikke i koe)

  public Glass(int aG, int aS) {
    n = aG;
    s = aS;
    foreldre = new int[aG+1];
    for (int i=0; i<=aG; i++) foreldre[i] = -1;
    koe = new int[aG+1];
    // koe initialieres med startnoden (node n)
    koe[0] = aG; neste = 0; nestefri = 1;
  }

  public void snu() {
    while (neste<nestefri) {
      int opp = koe[neste++];
      if (opp==0) {// løsning
        veiFraRot(0);
        return;
      }
      // ma maksimalt antall oppGlass som kan snus
      // mi minimalt antall oppGlass som kan snus
      int ma = opp<s ? opp : s;
      int mi = n-opp<s ? s-(n-opp) : 0;
      for (int i=ma; i>=mi; i--) {
        int nyOpp=opp+s-2*i;
        if (foreldre[nyOpp]==-1 && nyOpp!=n) {
          foreldre[nyOpp] = opp;
          koe[nestefri++] = nyOpp;
        }
      }
    }
    System.out.println("Start: "+n+" opp, 0 ned har ingen løsning.");
  }

  public void veiFraRot(int node) {
    if (foreldre[node] == -1)
      System.out.print("Start: "+node+" opp, "+(n-node)+" ned.");
    else {
      veiFraRot(foreldre[node]);
      int i = (foreldre[node]-node+s)/2;
      System.out.println(" Snu "+i+" oppGlass og "+(s-i)+" nedGlass");
      System.out.print("Gir:   "+node+" opp, "+(n-node)+" ned.");
    }
  }

  public static void main(String[] args) {
    int aGlass = Integer.parseInt(args[0]);
    int aSnu = Integer.parseInt(args[1]);
    if (aSnu>aGlass || aSnu<1) {
      System.out.println("0<antall glass i snuoperasjon<=antall glass");
      System.exit(-1);
    }
    System.out.println("Antall glass="+aGlass+
                       " Antall glass i snuoperasjon="+aSnu);
    new Glass(aGlass, aSnu).snu();
  }
}
```

Løsningsforslag 1.6:
Av uttrykket for neste tilstand (`nyOpp=opp+s-2*i`) ser vi at dersom `opp` er et oddetall og `s` er et partall, så må også `nyOpp` være et oddetall. Dvs. vi kan aldri nå sluttilstand 0 (som er et partall).

Løsningsforslag 2.1



min tre



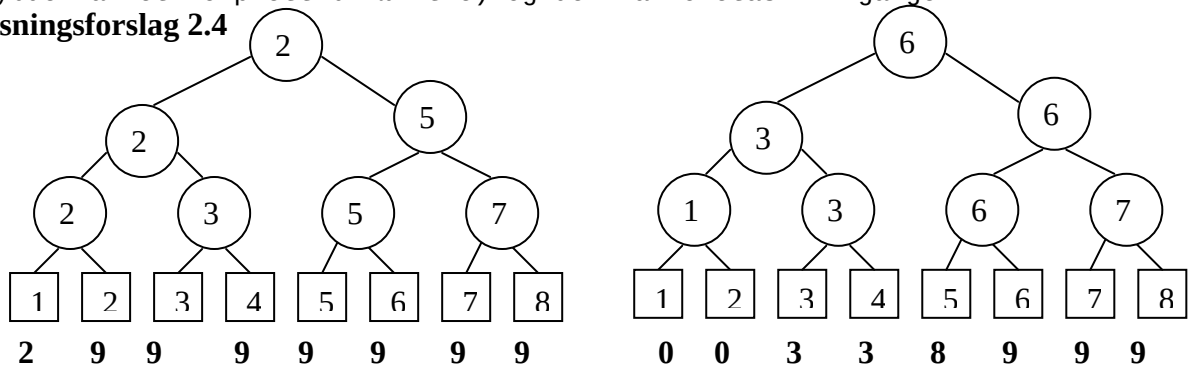
max tre

Løsningsforslag 2.2
Verdiene er brukt for å avgjøre vinneren av hver kamp. Vinneren av hele turnamenten er den med minst verdi. Den verdien må forandres til en veldig stor tall slik at den kan aldri vinne igjen. Operasjon rePlay brukes for å ta vare på forandringene. Den neste vinneren er da den som er den neste i sortert rekkefølge. Vi gjentar samme strategien (forandrer verdien av vinneren og spiller igjen) til vi har sortert ferdig.

Løsningsforslag 2.3

Lavere grensen er den samme som alltid når sammeligninger brukes: O(nlogn). Øvre grensen er også O(nlogn) og det er fordi hver rePlay tar O(logn) tid (dybden av et komplett binær tre) og den må foretas n-1 ganger.

Løsningsforslag 2.4



Løsningsforslag 2.5

```
/* Det følgende er et kjørbart program.
   Kandidatene er bare bedt om å programmere metoden
   public static void firstFitPack(int[] objectSize,
                int binCapacity, WinnerTree maxWT)
*/

public class Player {
  int name;
  int value;

  public Player(int name, int value) {
    this.name=name;
    this.value=value;
  }
}

public interface WinnerTree {
  public void initialize(Player[] thePlayers);
  public Player getWinner(int i);
  public void rePlay(int i);
}

public class WinnerTreeImpl implements WinnerTree {//max winner tree
  int n; // number of players
  Player[] matchTree;

  /**
   * Construction of a WinnerTree given the players
   * Array implementation a la heap
   */
  public void initialize(Player[] players) {
    n = players.length;
    matchTree = new Player[2*n];
    for (int j=0; j<n; j++) { // insert the players
      matchTree[j+n] = players[j];
    }
    for (int i=n-1; i>0; i--) {// play the matches
      if (matchTree[2*i].value<matchTree[2*i+1].value)
        matchTree[i] = matchTree[2*i+1]; // winner is right children
      else
        matchTree[i] = matchTree[2*i+1]; // winner is left children
    }
  }

  /**
   * Return the Player refered to by position given by the Navigator
   */
  public Player getWinner(int i) {
    return matchTree[i];
  }

  /**
   * Replay the matches started from the Player given by the Navigator
   * (ikke lagt inn test for tidlig avslutning)
   */
  public void rePlay(int i) {
    while (i>1) {
      i /= 2;
      if (matchTree[2*i].value<matchTree[2*i+1].value)
        matchTree[i] = matchTree[2*i+1]; // winner is right children
      else
        matchTree[i] = matchTree[2*i+1]; // winner is left children
    }
  }
}

public class BinPack {
```

```
  public static void firstFitPack(int[] objectSize, int binCapacity,
                                  WinnerTree maxWT) {
    int n = objectSize.length;
    Player[] players = new Player[n];
    for (int i=0; i<n; i++) {
      players[i] = new Player(i+1, binCapacity);
    }
    maxWT.initialize(players);

    // Everything set up and ready
    for (int i=0; i<n; i++) {// pack obj[i]
      int s = objectSize[i];
      int nav = 1;
      if (maxWT.getWinner(nav).value<s) {
        System.out.println("Ikke plass til objekt:"+(i+1)+
                           " av størrelse:"+s);
        return;
      }
      while (nav<n) {
        nav *= 2;
        if (maxWT.getWinner(nav).value<s) nav += 1;
      }
      // here when leftmost player with enough space is found
      Player p = players[nav-n];
      p.value -= s;
      maxWT.rePlay(nav);
      System.out.println("Objekt:"+(i+1)+" av størrelse:"+s+
          " legges i Bin:"+p.name+" Gjenværende kapasitet:"+p.value);
    }
  }

  public static void main(String[] args) {
    int[] obj= {7, 2, 5, 3, 6, 6, 1, 8};
    firstFitPack(obj, 9, new WinnerTreeImpl());
  }
}
```

Løsningsforslag oppgave 3.1

En asyklisk urettet graf med max antall kanter må være et spanntre (eller skog, hvis grafen er ikke sammenhengende). Hvilken som helst algoritmen for å konstruere et spantre kan brukes, kantene som ikke er med i treet er de som må fjernes. Hvis man bruker dybde først søk, deler man kanter i 2 grupper: tre kanter og bakkanter. Alle bakkantene må fjernes.

Løsningsforslag oppgave 3.2

```
Algoritme for å klassifisere kantene i den rette grafen G med start i V.
Traverser G fra V dybde-først. Global (eller static) teller (kan også
overføres som parameter og returneres av metoden med litt mas).

dybdeForst()
  Merk node som A (aktiv)
  int minTeller=teller++;
  For alle kanter se på etterfølgernode:
    hvis usett, merk kant som treeEdge;
                etterfolgernode.dybdeForst();
    else
      hvis A-node, merk kant som backEdge.
      hvis S-node,
        hvis minTeller i S-noden > nånodens minTeller,
          merk kant som forwardEdge
        else
          merk kant som crossEdge
  Merk node som S (sett og behandlet)
```

```
  return //backtrack
```

Virker også for skog, dersom vi bare fortsetter å telle der vi slapp.
På ytterste nivå:

```
  Node usett=s;
  while (usett != null)
      s.dybdeForst();
      usett = let etter usett node
```

Løsningsforslag oppgave 3.3

Her må man bare observere at kryss kanter og forover kanter danner aldri løkker. Dermed må man fjerne bakkanter igjen. Eller kjøre topologisksortering algoritmen (fult akseptabel svar).