Avdeling for informasjonsteknologi, Høgskolen i Østfold
Sensorveiledning/Løsningsforslag
Eksamen i ITF20006 Algoritmer og datastrukturer
14.05.2019

Oppgave 1

1 3
2 18
3 ABCD
4 Både 1 og 2
5 Bare 2
6 Ingen av dem
7 Alle tallene er heltall med maksimalt m siffer, der m er mye mindre enn n
8 Arrayen er nesten sortert, det er bare noen få tall som står feil
9 O(n)
10 O(n^2)
11 O(n logn)
12 Minst (m - 1)/2
13 63
14 c
15 2
16 "Last come, first served"-hashing
17 1, 4, 6, 3, 2, 5, 7, 8
18 1, 4, 5, 6, 7, 2, 3, 8
19 0 7 8 2 3 8 14 15
20 1, 4, 5, 2, 3, 6, 7, 8

Oppgave 2

a)
```
int startSekvens1(int A[])
{
  // Sekvensielt søk for å finne ut hvor i arrayen den stigende
  // sekvensen starter

  int n = A.length;

  // Finner startindeksen, som er den eneste der verdien er
  // mindre enn foregående verdi
  for (int i = 1; i < n; i++)
      if (A[i] < A[i - 1])
        return i;

  // Hvis vi ikke fant noen startindeks, begynner sekvensen i
  // indeks 0
  return 0;
}
```

b)
```
static int startSekvens2(int A[])
{
  // Binært søk for å finne ut hvor i arrayen den stigende
  // sekvensen starter

  int n = A.length;

  // Sjekker om sekvensen starter i indeks 0
  if (!(A[0] > A[n - 1]))
      return 0;
```

```
    int low = 0, high = n - 1, mid = 0;

    while (true)
    {
        // Beregner indeksen til midtpunktet i arraysegmentet
        // f.o.m. indeks low t.o.m. indeks high
        mid = low + (high - low + 1)/2;

        // Sjekker om midtpunktet er starten på sekvensen
        if (A[mid] < A[mid - 1])
          return mid;

        // Går videre til den halvdelen der starten på sekvensen
        // må befinne seg
        if (A[low] > A[mid])
          high = mid - 1;
        else
          low = mid + 1;
    }
  }
```

Oppgave 3

```
// Finner og skriver ut alle løsninger av tårnproblemet
//
// Merk: Dette er nøyaktig det samme som å finne alle permutasjoner av
// tallene 1, 2, 3, ..., n

public class tårn
{
    public static int n;           // Størrelse på brettet
    public static int p[];         // Lagrer en løsning på tårnproblemet

    public static boolean brukt[]; // Merker av brukte kolonner

    public static void lagLøsning(int rad)
    {
      // Skriver ut alle mulige løsninger på tårnproblemet rekursivt
      // ved å sette ut tårn fra og med denne raden og til og med rad n.
      // Det er allerede satt ut tårn på radene 1, 2, ..., rad-1

      if (rad == n+1)
          // Laget ferdig en løsning, skriver ut
          skrivLøsning();
      else
      {
          // Setter tårn inn på alle ledige kolonner i denne raden,
          // for hver plassering går vi rekursivt videre til neste
          // rad

          for (int kol = 1; kol <= n; kol++)
          {
            if (!brukt[kol])
            {
                p[rad] = kol;
                brukt[kol] = true;
                lagLøsning(rad + 1);
                brukt[kol] = false;
            }
```

```
        }
      }
    }

    public static void skrivLøsning()
    {
      for (int i = 1; i <= n; i++)
          System.out.print(p[i] + " ");
      System.out.println();
    }

    public static void main(String args[])
    {
      System.out.print("n?: ");
      n = Integer.parseInt(System.console().readLine());

      // Oppretter de to arrayene som brukes
      p = new int[n + 1];
      brukt = new boolean[n + 1];

      // Setter alle kolonner til å være ledige
      for (int i = 1; i <= n; i++)
          brukt[i] = false;

      // Lager og skriver ut alle løsninger
      lagLøsning(1);
    }
}
```

Oppgave 4

a)
```
    void settInn(int verdi)
    {
      // Setter inn en ny forekomst av en verdi i søketreet

      if (erTomt())
          rot = new treNode(verdi);
      else
      {
          treNode denne = rot;
          boolean ferdig = false;

          while (!ferdig)
            if (verdi == denne.data)
            {
                denne.antall++;
                ferdig = true;
            }
            else if (verdi < denne.data)
                if (denne.venstre == null)
                {
                  denne.venstre = new treNode(verdi);
                  ferdig = true;
                }
                else
                  denne = denne.venstre;
            else
                if (denne.høyre == null)
                {
                  denne.høyre = new treNode(verdi);
```

```
            ferdig = true;
          }
          else
            denne = denne.høyre;

      }
    }
```

b)
```
  int settHøyde(treNode t)
  {
    // Setter høyden riktig for alle noder i treet med rot i t
    // Returnerer verdien som blir satt for høyden i t

    if (t == null)
        return -1;

    // Setter høyden rekursivt i begge subtrærne
    int h_venstre = settHøyde(t.venstre);
    int h_høyre   = settHøyde(t.høyre);

    // Setter høyden lik høyden av høyeste subtre pluss 1
    if (h_venstre > h_høyre)
        t.høyde = h_venstre + 1;
    else
        t.høyde = h_høyre + 1;

    return t.høyde;
  }
```

c)
```
  boolean erPerfekt(treNode t)
  {
    // Returnerer true hvis treet med rot i t er et perfekt binært tre
    // Forutsetter at høyden er satt riktig i alle noder

    // Regner et tomt tre for å være perfekt
    if (t == null)
        return true;

    // Hvis treet bare består av én node er det et PBT
    if (t.høyde == 0)
        return true;

    // Hvis bare et av subtrærne er null, er det ikke et PBT
    if (t.venstre == null || t.høyre == null)
        return false;

    // Sjekker rekursivt at begge subtrærne er PBT'er og har korrekt høyde
    if ((erPerfekt(t.venstre) && (t.høyde - t.venstre.høyde == 1)) &&
        (erPerfekt(t.høyre)   && (t.høyde - t.høyre.høyde   == 1)))
        return true;

    return false;
  }
```