Tietorakenteet, laskuharjoitus 9, 28.3.-1.4

1. Ongelmana on löytää maksimikeosta pienin alkio. Keko oletetaan toteutetuksi normaaliin tapaan taulukkona. Esitä ongelmalle algoritmi, joka tutkii mahdollisimman vähäisen määrän keossa olevia avaimia.

2. Ongelmana on tulostaa maksimikeosta kaikki annettua arvoa x suuremmat avaimet. Keon sisältöä ei saa muuttaa. Esitä ongelmalle algoritmi, jonka aikavaativuus on lineaarinen tulostettavien alkioiden lukumäärän suhteen.

3. (a) Esitä maksimikeolle operaatio `heap-delete-key(H,i)`, joka poistaa keosta indeksissä i olevan avaimen. Mikä on algoritmisi aikavaativuus ja tilavaativuus?

   (b) Esitä maksimikeolle operaatio `heap-change-key(H,i,k)` joka muuttaa indeksissä i olevan avaimen arvoksi k:n. Avaimen arvo voi siis nousta tai laskea. Mikä on algoritmisi aikavaativuus ja tilavaativuus?

   Bonus: Miten operaatio `heap-change-key(H,i,k)` voitaisiin toteuttaa ilman rekursiota?

4. Toteuta maksimikeko haluamallasi ohjelmointikielellä. Toteuta operaatiot `heapify`, `heap-insert`, `heap-max` ja `heap-del-max`. Testaa myös, että toteutuksesi toimii oikein.

   Monisteen pseudokoodi olettaa että taulukon indeksöinti alkaa 1:stä. Toteutuksessasi kannattaa sijoittaa keko alkamaan taulukon indeksistä 0. Näin seuraavan tehtävän kekojärjestäminen onnistuu "normaaliin tapaan" täytetyille taulukoille.

5. Toteuta monisteen sivulla 339 esitetty kekojärjestämisalgoritmi.

   Toteuta myös monisteen sivulla 18 esitelty lisäysjärjestämisalgoritmi.

   Vertaa algoritmien suorituskykyä erikokoisilla suurilla syötteillä, tapauksissa, joissa syötteenä oleva taulukko on

   - satunnaisessa järjestyksessä
   - valmiina järjestyksessä
   - valmiina käänteisessä järjestyksessä

   Suositus: Koska jatkamme tehtävää ensi viikolla, kannattaa tehdä järjestämisalgoritmeistasi olioita jotka toteuttavat rajapinnan

```
public interface Sort {
    void sort(int[] table);
}
```

   Tällöin algoritmia kutsutaan seuraavasti:

```
int[] taulukko = {2, 7, 1, 3, 6, 5, 4};
tulosta(taulukko);
new InsertionSort().sort(taulukko);
```

Eli luodaan algoritmiolio ja kutsutaan sen `sort`-metodia parametrinaan järjestettävä taulukko.

Jos järjestämisalgoritmisi toteuttaa rajapinnan, voi käyttää osoitteesta

`https://wiki.helsinki.fi/display/javak11/Jarjestysalgoritmien+suorituskykymittaaja`

löytyvää helppokäyttöistä mittaustyökalua suorituskykymittausten tekemiseen. Työkalua käytetään seuraavaan tyyliin:

```
public static void main(String[] args) {
    Mittaaja mittaaja = Mittaaja.RANDOM();
       // tai Mittaaja.ORDER() tai Mittaaja.REVERSE();

    mittaaja.lisaa( new QuickSort() , "quick");
    mittaaja.lisaa( new TunedQuickSort(10), "tquick" );
    mittaaja.lisaa( new MergeSort(), "merge" );
    mittaaja.lisaa( new InsertionSort(), "insert" );

    mittaaja.teeJaTulostaMittaus();
}
```

Eli ensin pyydetään sopivaa aineistoa käyttävä mittaajaolio. Tämän jälkeen lisätään mitattavat algoritmit mittaajalle ja pyydetään tekemään mittaus.

Tulostuu

```
Mittausaineisto Arvottu:

       n   quick  tquick   merge  insert
----------------------------------------
   10000      43       6      95     124
   20000      45       8       7     316
   40000      15      14      16    1360
   80000      32      24      30    5837
  160000      60      53      68      -1
  320000     131     115     149      -1
  640000     275     235     282      -1
 1280000     583     477     624      -1
 2560000    1195     992    1288      -1
 5120000    2875    2398    2596      -1
```

Tulostuksessa -1 tarkoittaa että mittausta ei tehty.