Tentamen – extra del

Förklaringar

Den här delen av tentamen gör möjligt att få betyg högre än E

Utöver den obligatoriska delen, kan studenten göra även den här delen av tentamen. Denna del har sin betydelse bara i fall att studenten har klarat den obligatoriska delen. I så fall kan studenten samla tillräckligt antal poäng på den här delen, och uppnå ett betyg som är högre än E.

Antalet poäng och betyg

Totalt: 20 poäng

För betyget D räcker med: 4 poäng
För betyget C räcker med: 8 poäng
För betyget B räcker med: 14 poäng
För betyget A räcker med: 17 poäng

Uppgifter

Uppgift 1 (4 poäng)

En algoritm, som sorterar en sekvens med jämförbara element, kan beskrivas som nedan.

Sätt en pekare som heter `forst` att peka till första positionen i sekvensen, en pekare `sist` att peka till sista positionen i sekvensen, och en pekare `aktuell` ett steg efter pekaren `forst`. En ytterligare pekare som heter `hal` ska finnas, samt en extra variabel som heter `e`.

Upprepa följande så länge pekaren `aktuell` inte ligger till höger av pekaren `sist`.

Flytta det element som pekaren `aktuell` pekar på till variabeln `e`. Det uppstår ett hål på den position som pekaren `aktuell` pekar på. Sätt pekaren `hal` att peka dit.

Upprepa följande så länge pekaren `hal` ligger till höger av pekaren `forst`, och det element som finns i variabeln `e` är mindre än det element som finns framför hålet.

Flytta det element som finns framför hålet in i hålet. Hållet går ett steg till vänster, så sätt pekaren `hal` att peka dit.

Flytta det elementet som finns i variabeln `e` in i hålet.

Flytta pekaren `aktuell` ett steg fram.

Skapa en metod `sort` som tar emot en heltalsvektor, och sorterar den enligt givna algoritmen.

Uppgift 2 (4 poäng)

En metod `calculate` utför en beräkning med heltal:

```
public static int calculate (int m, int n)
{
    int    p = m / n;
    int    q = m % n;
    int    r = p + q;
    int    res = (int) Math.pow (r, 2);

    return res;

}
```

Metoden `calculate` kan anropas, till exempel, så här:

```
int    m = 6405;
int    n = 3200;
int    res1 = calculate (m, n);  // res1 blir 49
```

Men denna metod kan inte utföra beräkningen om heltalen är alltför långa. I så fall kan man representera heltalen med motsvarande teckensträngar, och få resultatet i form av en teckensträng. Motsvarande metod skulle kunna anropas, till exempel, så här:

```
String    n1 = "6400000000000000000005";
String    n2 = "3200000000000000000000";
String    res2 = calculate (n1, n2);  // res2 blir "49"
```

Skapa en sådan metod `calculate`, som kan utföra motsvarande beräkning med långa heltal. Inuti metoden ska objekt av typen `java.math.BigInteger` skapas utifrån givna teckensträngar. Beräkningen ska utföras med motsvarande metoder i klassen `BigInteger`.

Uppgift 3 (4 poäng)

En student lyckades skapa följande program, bestående av två klasser:

```
class List
{
     private static class Node
     {
        public int     value;
        public Node    next;

        public Node (int value)
        {
            this.value = value;
            this.next = null;
         }
     }

     private Node    first = null;

    // add lägger till ett givet heltal till listan
    public void add (int value)
    {
        // koden här
    }

     // ytterligare metoder
}


class UseList
{
    public static void main (String[] args)
    {
        List    list = new List ();
        int[]    v = {1, 4, 5, 7, 10};
        for (int i = 0; i < v.length; i++)
            list.add (v[i]);

        System.out.println (list);
     }
}
```

Vid exekveringen av programmet fick studenten följande utskrift:

```
[1, 4, 5, 7, 10]
```

Studenten gjorde sedan ett misstag: han tog bort koden i metoden `add`, och hade därefter svårt att återställa programmet.

Hjälp studenten: skapa metoden `add`.

Uppgift 4 (4 poäng)

Klasserna `Int` och `UseInt` är givna, som nedan:

```
class Int
{
    private int    n;

    public Int (int n)
    {
        this.n = n;
    }

    public String toString ()
    {
        return "" + n;
    }
}


class UseInt
{
    public static void main (String[] args)
    {
        Int[]    i = new Int[4];
        i[0] = new Int (2);
        i[1] = new Int (4);
        i[2] = new Int (1);
        i[3] = new Int (3);

        java.util.Arrays.sort (i);
        System.out.println (i[0] + " " + i[1] + " " + i[2] + " " + i[3]);
    }
}
```

Vid exekveringen av programmet `UseInt` kastas ett undantag i metoden `sort`: metoden kan inte sortera en vektor med objekt av typen `Int`. Om man vill kunna sortera vektorn, måste man lägga till ytterligare funktionalitet i klassen `Int`.

Komplettera klassen `Int`, så att följande utskrift erhålls vid exekveringen av programmet `UseInt`:

```
1 2 3 4
```

Uppgift 5 (1 poäng + 2 poäng + 1 poäng)

En metod `min` bestämmer det minsta heltalet i en sekvens:

```
// min returnerar det minsta heltalet i en sekvens med heltal
public static int min (int[] element)
{
    if (element.length == 0)
        throw new IllegalArgumentException ("tom sekvens");

    int[]    sekvens = element;
    int      antaletPar = sekvens.length / 2;
    int[]    delsekvens = new int[antaletPar];
    int      i = 0;
    int      j = 0;
    while (antaletPar >= 1)
    {
        // urskilj en delsekvens med de tänkbara heltalen
        i = 0;
        j = 0;
        while (j < antaletPar)
        {
            delsekvens[j++] = (sekvens[i] < sekvens[i + 1])?
                              sekvens[i] : sekvens[i + 1];
```

```
            i += 2;
        }

        // utgå nu ifrån delsekvensen
        sekvens = delsekvens;
        antaletPar = antaletPar / 2;
    }

    // sekvens[0] är det enda återstående tänkbart heltal -
    // det är det minsta heltalet
    return sekvens[0];
}
```

a) Bevisa att metoden inte är korrekt: ange en heltalssekvens då metoden ger ett felaktigt svar.

b) Vad är otillräckligt i den strategi som metoden använder?

c) För vissa sekvenslängder fungerar metoden bra: man kan variera både heltal och deras ordning – svaret blir alltid korrekt. Vilka är dessa sekvenslängder?