Tentamen – extra del: lösning

Uppgifter: lösningar

Uppgift 1 (4 poäng + 1 poäng)

a) (4 poäng)

```
mellanstation (ai, i = 1 ... n,
               bi, i = 1 ... n)
{
    k = 1  // ordningsnummer för valda mellanstationen
    minimalVagLangd = a1 + b1
    for i = 2 ... n  // i är ordningsnummer för aktuella mellanstationen
    {
        vagLangd = ai + bi
        if vagLangd < minimalVagLangd
        {
            minimalVagLangd = vagLangd
            k = i
        }
    }

     return k
}
```

b) (1 poäng)

Man kan först bestämma och lagra längder för alla möjliga vägar, och sedan hitta den minsta vägen och motsvarande mellanstation. Eftersom det finns n möjliga vägar (en väg per mellanstation), ska motsvarande lagringsstruktur lagra n längder. Minneskomplexiteten blir $\theta(n)$.

Uppgift 2 (5 poäng)

```
import static java.lang.System.out;

// Worker representerar en arbetare
public class Worker
{
    // antalet arbetsdagar
    private int    countDays;

    // Worker skapar en arbetare som jobbar ett givet antal dagar
    public Worker (int countDays)
    {
        this.countDays = countDays;
    }

    // work utför ett visst jobb under ett givet antal arbetstimmar varje arbetsdag
    public void work (int countHours)
    {
        out.println ("working " + countDays + " x " + countHours + " hours");
    }


    // Supporter representerar en stödarbetare
    public class Supporter
    {
        // andel av stödarbete: antalet stöddagar relativt arbetarens antal arbetsdagar
        private double    ratio;
```

```
        // Supporter skapar en stödarbetare som stödjer arbetaren under ett givet andel
        // arbetsdagar
        public Supporter (double ratio)
        {
            this.ratio = ratio;
        }

        // support stödjer arbete på ett givet sätt
        public void support (String way)
        {
            out.println ("supporting " + ratio + " x " + countDays + " days " + way);
            // Eller:
            // out.println ("supporting " + this.ratio + " x " + Worker.this.countDays
            // + " days " + way);
        }
    }
}
```

Uppgift 3 (1 poäng + 2 poäng + 2 poäng)

a) (1 poäng)

```
public interface StringCollection
{
    // size returnerar antalet strängar i samlingen
    int size ();

    // add lägger till en given sträng i samlingen
    void add (String element);
}
```

b) (2 poäng)

```
public int size ()
{
    int     countElements = 0;
    Node    currentNode = firstNode;
    while (currentNode != null)
    {
        countElements++;
        currentNode = currentNode.nextNode;
    }

    return countElements;
}
```

c) (2 poäng)

```
public void add (String element)
{
    Node    newNode = new Node (element);
    newNode.nextNode = firstNode;
    firstNode = newNode;
}
```

Uppgift 4 (2 poäng + 1 poäng + 2 poäng)

a) (2 poäng)

I första pass genom `while`-loopen utförs $n - 1$ elementjämförelser, i andra pass $n - 2$ jämförelser, och så vidare. I sista pass utförs bara en jämförelse. Det totala antalet jämförelser är:

$$(n - 1) + (n - 2) + \ldots + 1 = n(n - 1)/2$$

Motsvarande komplexitetsfunktion är:

$t(n) = n(n - 1) / 2$

b) (1 poäng)

Komplexitetsfunktionen för algoritmen kan föreställas så här:

$t(n) = n^2/2 - n/2$

För stora n dominerar termen med n^2. Därför:

$t(n) \in \theta(n^2)$

Algoritmen är en kvadratisk algoritm när det gäller antalet elementjämförelser.

c) (2 poäng)

I bästa fall sker inget elementutbyte. Det händer när sekvensen redan från början är sorterad. Algoritmens tidskomplexitet är:

$b(n) = 0$

Värsta fall uppstår när den ursprungliga sekvensen är sorterad i avtagande ordning. I så fall utförs ett utbyte vid varje elementjämförelse: det finns så många utbyten som jämförelser. Tidskomplexiteten är:

$w(n) = n(n - 1) / 2$