Supervision work
Supervision 9

Tudor Avram
Homerton College, tma33@cam.ac.uk

19 Jan 2016

1 Algorithms Sheet 1

1.1 Exercise 1

When specifying an algorithms **preconditions** and **postconditions** have an important role when we want to avoid the algorithm to crash. For example, we need to have a precondition when we divide 2 numbers, to be sure that we don't divide by 0.
Assertions have a big role when implementing an algorithm, because they allow us to handle errors when they appear.

1.2 Exercise 2

(i) For *double sqrt(double)* :
preconditions : checking if the method's argument is a positive number.
postconditions : checking if the square of the result is equal to the method's argument

(ii) For *double divide(double a, double b)* :
preconditions : checking if b is non-zero, so that we don't divide by 0.
postconditions : checking if the result is correct

(iii) For *void normalise(int[] a)* :
preconditions : checking if the size of a is 3.

1.3 Exercise 3

In my opinion, the given statement is true up to one point. Assertions should not be in the final product, which should run indefinitely. But they are mostly used for developing purposes and that's why I think that they can be appropriate even when designing such programmes.

1.4 Exercise 4

For each complexity, I try to find 2 constants : c = the multiplication factor and n_0, such as $c * g(n) - f(n) \geq 0, \forall n \geq n_0$
(i) I choose $c = 7$ and $n_0 = 7$; $g(n) = n^2 \Leftrightarrow c * g(n) = 7n^2$
$h(n) = c * g(n) - f(n) = 7n^2 - 6n^2 - 7n = n^2 - 7n$
$h(n) \geq 0, \forall n \geq n_0$ =>The complexity of f(n) is O(n^2)

(ii) I choose $c = 6$ and $n_0 = 1$; $g(n) = n \Leftrightarrow c * g(n) = 6n$
$h(n) = c * g(n) - f(n) = 6n - \frac{6n^2+7log(n)}{n} = 6n - 6n - \frac{7log(n)}{n} = \frac{7log(n)}{n}$
$h(n) \geq 0, \forall n \geq n_0$ =>The complexity of f(n) is O(n)

(iii) I choose $c = 2$ and $n_0 = 1$; $g(n) = n^2 \Leftrightarrow c * g(n) = 2n^2$
$h(n) = c * g(n) - f(n) = 2n^2 - 4n^2 - 2 = -2n^2 - 2$
$h(n) \leq 0, \forall n \geq n_0$ =>The complexity of f(n) is $\theta(n^2)$

(iv) $f(n) = 1^2 + 2^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6} = \frac{2n^3+3n^2+n}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$
I choose $c = \frac{1}{3}$ and $n_0 = 1$; $g(n) = n^3 \Leftrightarrow c * g(n) = \frac{1}{3}n^3$
$h(n) = c * g(n) - f(n) = \frac{1}{3}n^3 - \frac{1}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n = \frac{1}{2}n^2 - \frac{1}{6}n$
$h(n) \leq 0, \forall n \geq n_0$ =>The complexity of f(n) is $\theta(n^3)$

2 Java Past Paper

2.1 Question 4, Paper 1, 2014

(a) The requested class definition is in the following code :

```
package uk.ac.cam.tma33.y2014p1q4;

public class StudentInfo implements
        Comparable<StudentInfo>{

private String mName;
private int mCompleted;


public StudentInfo(String name, int complete) {
        mName = name;
        mCompleted = complete;
}

public void setCompleted(int n) {
        mCompleted = n;
}
```

```
public boolean equals(StudentInfo student) {
        if (!mName.equals(student.mName)) return false;
        if (mCompleted != student.mCompleted) return false;
        return true;
}

@Override
public int compareTo(StudentInfo arg0) {
        if (this.equals(arg0)) return 0; // they are equal
        if (this.mCompleted > arg0.mCompleted) return 1;
        if (this.mCompleted < arg0.mCompleted) return -1;
        return this.mName.compareTo(arg0.mName);
}

}
```

(b) (i) The implementation of SubscribableStudentInfo is :

```
package uk.ac.cam.tma33.y2014p1q4;

import java.util.ArrayList;

public class SubscribableStudentInfo extends StudentInfo{

private ArrayList<UpdatableTreeSet> list;

public SubscribableStudentInfo(String name, int complete) {
        super(name, complete);
        list = new ArrayList<UpdatableTreeSet>();
}

@Override
public void setCompleted(int n) {
        for (UpdatableTreeSet treeSet: list) {
                treeSet.beforeUpdate(this);
        }

        super.setCompleted(n);

        for (UpdatableTreeSet treeSet: list) {
                treeSet.afterUpdate(this);
        }
}

public boolean isSubscribed(UpdatableTreeSet s) {
        for (UpdatableTreeSet treeSet: list) {
```

```
                if (treeSet.equals(s)) return true;
        }
        return false;
}

public boolean addSubscriber(UpdatableTreeSet s) {
        return list.add(s);
}

public boolean removeSubscriber(UpdatableTreeSet s) {
        return list.remove(s);
}

}
```

(ii) The complete implementation of UpdatableTreeSet is :

```
package uk.ac.cam.tma33.y2014p1q4;

import java.util.TreeSet;

public class UpdatableTreeSet extends
        TreeSet<SubscribableStudentInfo> {

public void beforeUpdate(SubscribableStudentInfo s) {
        remove(s);
}


public void afterUpdate(SubscribableStudentInfo s) {
        add(s);
}

public boolean add(SubscribableStudentInfo s) {
        if (s.isSubscribed(this)) return false;
        super.add(s);
        return s.addSubscriber(this);
}

public boolean remove(SubscribableStudentInfo s) {
        if (s.isSubscribed(this)) {
                super.remove(s);
                return s.removeSubscriber(this);
        }
        else return false;
}
```

```
}
```