CSC 212 (Honor) Midterm 1 - Spring 2015

College of Computer and Information Sciences, King Saud University
Exam Duration: 2 Hours

12/03/2015

Question 1 [25 points]

1. (a) Show that $f \in \Theta(g)$ and $h \in \Omega(g)$ implies that $f \in O(h)$.

   (b) Give and compare the growth rates of $3n + 2\log(n^n) + \log(n)^2$ and $n^2 + 3\log(n^n) + 2^{\log n}$.

2. Consider the following method:

```
int func (int A[], int n) {
        boolean B = new boolean[n];
        for (int i = 0; i < n; i++) {
                B[i] = true;
        }
        int nbr = 0;
        for (int i = 0; i < n; i++) {
                for (int j = i+1; j < n; j++) {
                        if (A[j] == A[i] && B[j]) {
                                nbr++;
                                B[j] = false;
                        }
                }
        }
        return n - nbr;
}
```

   (a) What is the return value of this method for the array: 1, 2, 1, 1, 3, 2, 1, 3, 4. What does this method do?

   (b) Give its performance.

   (c) Knowing that an array can be sorted in $O(n \log n)$, What would be the performance of re-implementation of the above method that uses such sorting algorithm?

Question 2 [25 points]

1. Write the method *swapHalves*, member of the class *LinkedList*, that swaps the two halves of the list.

   Example 2.1. *The list* $A \to B \to C \to D$ *becomes* $C \to D \to A \to B$*, whereas* $A \to B \to C \to D \to E$ *becomes* $C \to D \to E \to A \to B$*.*

2. Write the method *semiReverse*, member of the class *DoubleLinkedList* that reverses half the elements of the list as shown in the following example:

   Example 2.2. *The list* $A \to B \to C \to D \to E \to G$ *becomes* $G \to B \to D \to C \to E \to A$*, whereas* $A \to B \to C \to D \to E$ *becomes* $E \to B \to C \to D \to A$*.*

Question 3 [25 points]

In a randomized queue, the element to be served is chosen uniformly at random. This ADT supports the following operations:

```
public boolean full() // is the queue full?
public int length() // return the number of elements in the queue
public void enqueue(T val) // add the element val
public T serve() // remove and return a random element
```

1. Give an array implementation (ArrayRQueue) of this ADT.
2. Give a linked implementation (LinkedRQueue) of this ADT.
3. Compare the performance of the two implementations. Which one has the better performance?

Question 4 [25 points]

A map containing n cities which are connected by roads is represented as a list of lists. The length of the list is n, and each element i in this list contains the list of the cities that are connected to city i (if a city i is connected to a city j, then j is also connected to i). The information about a road is contained an object of class *Edge* shown below.

```
public class Edge {
        public int i; // The starting node
        public int j; // The end node
        public int w; // The weight
        public Edge(int i, int j, int w) {
                this.i = i;
                this.j = j;
                this.w = w;
        }
}
```

Example 4.1. *The map in Figure 1 is represented as follows:*

$$
\begin{array}{ll}
\square & \rightarrow (0,1,1) \rightarrow (0,2,2) \rightarrow (0,4,1) \\
\downarrow & \\
\square & \rightarrow (1,0,1) \rightarrow (1,2,3) \\
\downarrow & \\
\square & \rightarrow (2,0,2) \rightarrow (2,1,3) \rightarrow (2,3,2) \\
\downarrow & \\
\square & \rightarrow (3,2,2) \rightarrow (3,4,1) \\
\downarrow & \\
\square & \rightarrow (4,0,1) \rightarrow (4,3,1)
\end{array}
$$

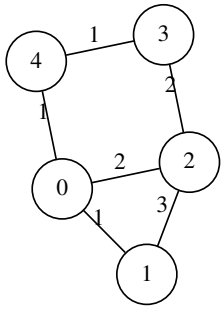

Figure 1: A map

Consider the class *Map* below.

1. Write the method *nbLess(int d)* that returns the number of roads with length at most d (count only edges where $i < j$).

2. Write the method *getRoads(int d)* that returns all the edges that have exactly length d (return only edges where $i < j$).

```
public class Map {
        private int nbCities; // The number of cities.
        private List<List<Edge>> roads; // The roads
        ...
        public int nbLess(int d) {
        }
        public List<Edge> getRoads(int d) {
        }
}
```

Specification of ADT List

- findFirst ( ): **requires**: list L is not empty. **input**: none. **results**: first element set as the current element. **output**: none.

- findNext ( ): **requires**: list L is not empty. Current is not last. **input**: none. **results**: element following the current element is made current. **output**: none.

- retrieve (Type e): **requires**: list L is not empty. **input**: none. **results**: current element is copied into e. **output**: element e.
- update (Type e): **requires**: list L is not empty. **input**: e. **results**: the element e is copied into the current node. **output**: none.
- insert (Type e): **requires**: list L is not full. **input**: e. **results**: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. **output**: none.
- remove ( ): **requires**: list L is not empty. **input**: none. **results**: the current element is removed. If L is empty, current will point to null. If the next element exists, it is made current, else the first element is made current. **output**: none.
- full (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L has reached the maximum then flag is set to true otherwise false. **output**: flag.
- empty (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false. **output**: flag.
- last (boolean flag): **requires**: L is not empty. **input**: none. **results**: if the last element is the current element then flag is set to true otherwise false. **output**: flag.

Specification of ADT Queue

- enqueue (Type e): **requires**: Queue Q is not full. **input**: Type e. **results**: Element e is added to the queue at its tail. **output**: none.
- serve (Type e): **requires**: Queue Q is not empty. **input**: none. **results**: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
- length (int length): **requires**: none. **input**: none. **results**: The number of elements in the Queue Q is returned. **output**: length.
- full (boolean flag): **requires**: none. **input**: none. **results**: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.