Computer Science Degree

Data Structures & Algorithms, Group 89M, 2016/2017

14th March 2017

Surname and name:

…..

Exercise 1

Given a stack, IntegerStack implementing the interface IStack, develop a recursive method to remove all the elements in the stack. **(0,5 points)**

Solution

```
public void removeAll(){
        if(!this.isEmpty()){
            this.pop();
            removeAll();
        }
    }
```

Exercise 2

There is a need to implement a Request Management System (RMS) for a web application. When a request arrives to the web application, if another request is being served, the new one can be refused. To avoid loosing these requests, the RMS must store them in order of arrival. Each request includes an IP address, a user login and a password.

1.- Which linear structure will you use to implement this request management system? Provide the implementation of the basic elements for the linear structure. In this section, you ONLY have to write headers for classes, attributes and constructors (if! You consider them adequate) needed to implement a linear structure. Explain the election of the type of linear structure you have chosen to implement your solution. **(0,5 points)**

Solution

1.-
A queue works with a first-in first-out philosophy and requests must be served as they arrive to the application.

```
public class SRequest {

	public String sIP;
	public String sName;
	public String sPassword;

	SRequest(){
		sIP = null;
		sName = null;
		sPassword = null;
	}

	SRequest(String ip, String name, String pass){
		sIP = ip;
		sName = name;
		sPassword = pass;
	}
}

public class SNodeRMS {

	SRequest elem;
	public SNodeRMS nextNode = null;

	public SNodeRMS(SRequest e) {
		elem = e;
	}
	public SNodeRMS(SRequest elem, SNodeRMS nextNode) {
		this.elem = elem;
		this.nextNode = nextNode;
	}

}

public class SQueueRMS implements IQueueRMS {

	private SNodeRMS frontNode;
	private SNodeRMS tailNode;
```

```
    ...
}
```

2.- Implement a method returning the next request to be served. The request must be removed from the linear structure. **(0,5 points)**

Solution

```java
public SRequest dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty!");
        return null;
    } else {
        SRequest firstElem = frontNode.elem;
        frontNode = frontNode.nextNode;
        if (frontNode == null) {
            tailNode = null;
        }
        return firstElem;

    }
}
```

3.- Compute the running time function for the algorithm developed in 2. Which is the complexity order in Big-Oh! notation? **(0,5 points)**

Solution

```java
public SRequest dequeue() {
    if (isEmpty()) {                               //1
        System.out.println("Queue is empty!");     //1
        return null;                               //1
    } else {
        SRequest firstElem = frontNode.elem;       //1
        frontNode = frontNode.nextNode;            //1
        if (frontNode == null) {                   //1
            tailNode = null;                       //1
        }
        return firstElem;                          //1
    }
}
```

The worst case is when there are elements in the queue so:

$T(n) = 6$

The complexity order in Big Oh! notation is

$O(1)$