Universidad Carlos III de Madrid

Algorithms and Data Structures (ADS)

Bachelor in Informatics Engineering
Computer Science Department

YEAR: 1º / SEMESTER: 2º

EXERCISES SHEET n1 (SOLUTION)

Authors:
Harith Al-Jumaily
Julian Moreno Schneider
Isabel Segura Bedmar
Juan Perea

Junio 2011

E-1	

Write a Java class to perform the following:

a) An operation to calculate the minimum and the maximum number of an array.
b) An operation to sort the elements of an array.

Analysis the complexity of both operations.
In the case of the sorting, ¿Can you propose different solutions?, What is the best solution?
Use Excel to draw the results.

Solution:

a)

```
public static int getMaxArray(int A[]) {
      if (A.length==0) {
            System.out.println("Array vacio");
            return -1;
      }
      int max=A[0];
      for (int i=1; i<A.length;i++) {
            if (A[i]>max) max=A[i];
      }
      return max;
}
```

The complexity of `getMaxArray` is linear.

```
public static int getMinArray(int A[]) {
      if (A.length==0) {
            System.out.println("Array vacio");
            return -1;
      }
      int min=A[0];
      for (int i=1; i<A.length;i++) {
            if (A[i]<min) min=A[i];
      }
      return min;
}
```

The complexity of getMinArray is also linear (O(n)). It traverses through all the elements of the array at least once, therefore, it needs at least n time units, where n is the size of the array.
The constants in the calculation of complexity can be dispensed.

b) Sort the elements of an array.

There are several sorting algorithms (bubble sort O (n^2), insertion sort O (n^2) and selection sort O (n^2)) (Advanced: quicksort and heapsort).

An interesting link:

http://lucas.hispalinux.es/Tutoriales/doc-programacion-algoritmos-ordenacion/alg_orden.pdf

E-2	

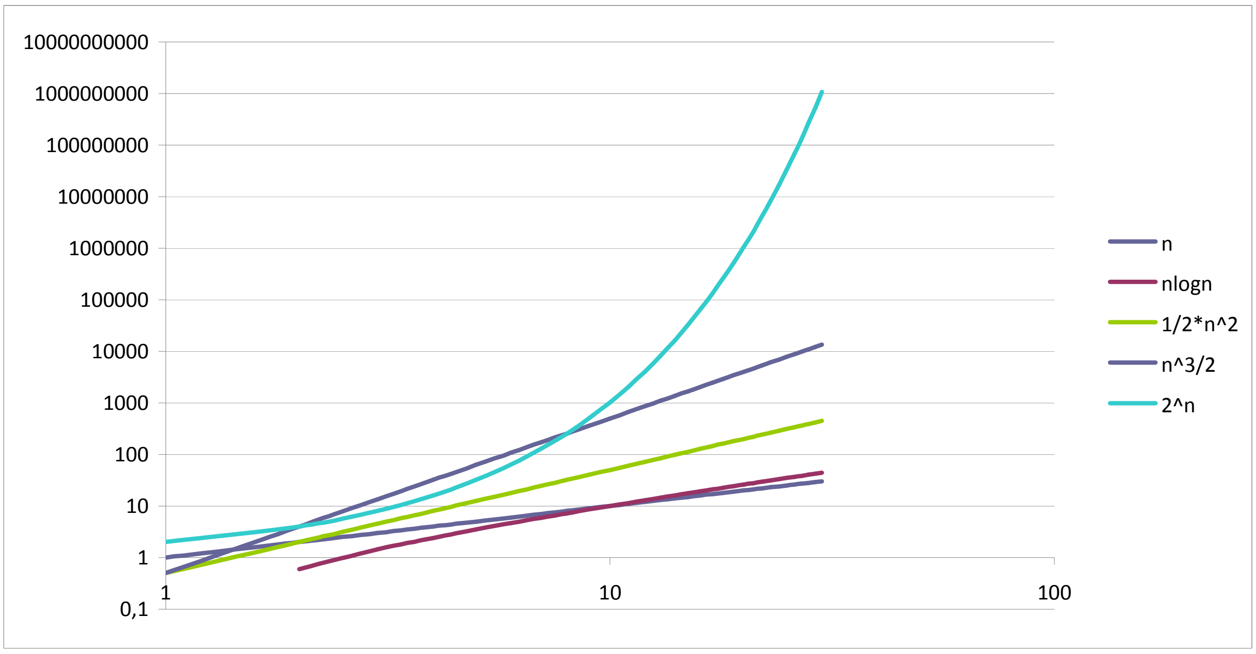
(a) Draw in the logarithmic scale the following functions: $f(n) = \mathrm{n}$, $g(n) = \mathrm{n} \cdot \log(n)$, $h(n) = 1/2.n^2$, $j(n) = n^3/2$, $k(n) = 2^n$. The logarithmic scale used in the x-axis and the y-axis.
(b) The algorithm A uses $25\mathrm{n} \cdot \log(n)$ operations while the algorithm B uses $2n^2$ operations.

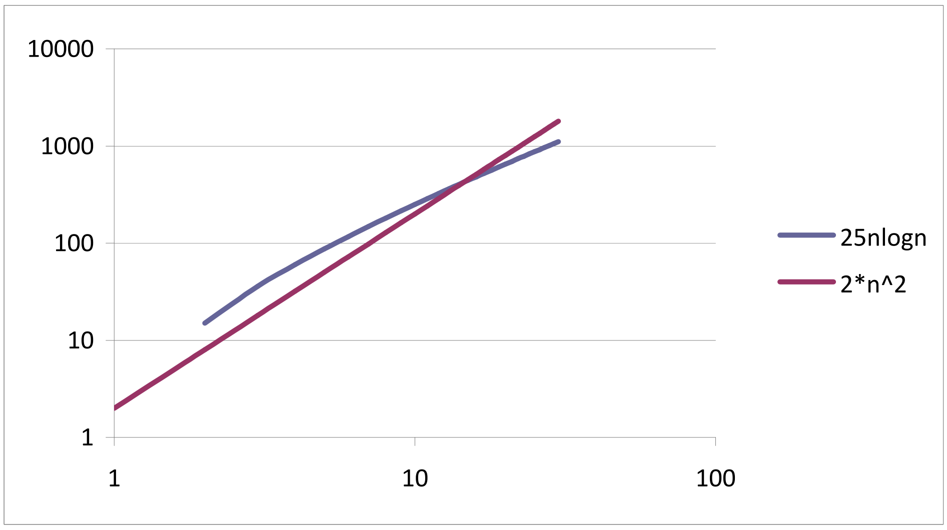
Which of the two algorithms is better asymptotically?
Determine the value n_0 that makes one algorithm is better than the another for problems of size $n \geq n_0$.

Note: Use Excel to draw the functions.

To use the logarithmic scale in Excel you must do the following:
- Draw the graph using the graph type XY (dispersión)
- Once the graph is finished, click on the x-axis
- Choose **Formato de eje** -> **Escala** -> **Escala logarítmica**
- Repeat the same operations with respect to the y-axis

Solution:





E-3	

Which of the following formulas are true?

a) $n^2 \log n \in \Omega(n)$

b) $2n+\log n \in O(n\log n)$

c) $3n^3 - 10 = \Theta(n^3)$

d) nlogn+4logn es O(nlogn)

Solution:

a. True

$n^2\log n \geq c.n$
for c=1 and for all $n_0 \geq 1$.

b. False

$2n+\log n \leq c.n\log n$
for c =1, and for all $n_0 \geq 9$.

c. True

$a.n^3 \leq n^3 - 10 \leq b.n^3$
for a=1, b=3 and for all $n_0 \geq 3$.

d. True

$n\log n+4\log n \leq c.n\log n$
for any $n \geq 1$ and c=1.

E-4	

Write a Java class to implement the following:

a) An operation to create a simple linked list with size 0, and the list size is increased each time a node is added.

b) An operation to calculate the maximum and minimum number of nodes in the list.

NOTE: You must implement yourself the list, thus you can not use the own data structures of Java such as ArrayList or Vector.

```java
/**
 * E-4-A
 * @param n
 */

public void insertarAlPrincipio(ListNode n) {
		n.next=head;
		head=n;
		size++;
}


/**E-4-B
 * To traverse a list from the beginning to the final to find the minimum
 * Complexity O(n)
 * @return
 */
public int buscarMinimo() {
		if (esVacia()) {
				System.out.println("La lista está vacía");
				return -1;
		}

		//we can use a node aux to traverse the list
		ListNode aux=head;

		//initialize the minimum and maximum with the value of the first item in the list
		int min=aux.item;

		while (aux.next!=null) {
				if (min>aux.item) min=aux.item;
				aux=aux.next;
		}


		System.out.println("The minimum of the list is " + min);
		return min;
}

/**E-
 * To traverse a list from the beginning to the final to find the maximum
 * Complexity O(n)
 * @return
 */
public int buscarMaximo() {
		if (esVacia()) {
				System.out.println("The list is empty");
				return -1;
		}

		//we can use a node aux to traverse the list
		ListNode aux=head;

		//initialize the minimum and maximum with the value of the first item in the list
		int max=aux.item;
	// traverse the list while it is not empty
	while (aux.next!=null) {
		if (max<aux.item) max=aux.item;
		aux=aux.next;
	}

	System.out.println("The maximum " + max);
	return max;
}
```

E-5	

Write a Java class to sort a simple list of integers from highest to lowest.

NOTE: Use the list class implemented in the exercise E-4 to solve this exercise. Don't use the own data structures of Java such as (ArrayList, Vectors, LinkedList, etc.).

```
/**
 * E-5
 * Given x, this method creates a new node, and it inserts the new node in the corresponding position take
into account that the nodes are saved from the high to the low.
 *
 * */
public void insertarMayorMenor(int x) {
	ListNode newNodo=new ListNode(x);

	if (esVacia()) {
		insertarAlPrincipio(newNodo);
	} else {
		ListNode aux=head;
		ListNode ant=null;
		// traverse the list until we find the position
		while (aux!=null && aux.item>x) {
			ant=aux;
			aux=aux.next;
		}

		if (aux==null) {
			//reach the final of the list
			ant.next=newNodo;
		} else if (aux.item<=x) {
			//insert before aux
			newNodo.next=aux;
			if (ant==null) {
				head=newNodo;
			} else {
				ant.next=newNodo;
			}

		}
		size++;
	}
```

E-6	

Write a Java class to perform the following:

a) An operation to find a value in a simple list.
b) An operation to delete the found value, or return "not found" in the case that the value does not exist.

NOTE: Use the list implemented in the exercise E-4 to solve this exercise. Don't use the own data structures of Java such as (ArrayList, Vectors, LinkedList, etc.)

```
/**
 * E-6
 * Delete the node that has the attribute (item=v)
 * if it is not exist any node then the method returns null
 * @param v
 * @return
 */
public void borrarNodo(int v) {
	//use the node aux to traverse the list

	if (esVacia()) {
		System.out.println("The list is empty. You can not delete any node");
		return;
	}
	ListNode aux=head;
	ListNode ant=null;

	while (aux!=null && aux.item!=v) {
		ant=aux;
		aux=aux.next;
	}


	if (aux==null) {
		System.out.println("Not found!!!. The list has not any node with value: " + v);
	} else {

		System.out.println("Found!!!. Proceed to delete: " + v);
		if (ant==null) {
			head=aux.next;
		} else {
			ant.next=aux.next;
		}
		size--;

	}

}
```

E-7	

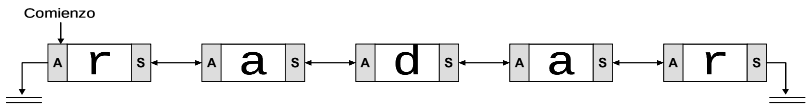
Wirte a Java class to implement a doubly linked list to store the products of a factory (**id, name, price**). The list should be ordered by **id**. You must do the following tasks:

- Implement in Java the ListaDoblementeEnlazada class
- Implement a method to insert a product in its proper place
- Implement a method to display the list of all products.

NOTE: Use the list implemented in the exercise E-4 to solve this exercise. Don't use the own data structures of Java such as (ArrayList, Vectors, LinkedList, etc.)

```java
/**
  * Because the list is sorted based on id of the products
  * we should find the position n of the list where a product is inserted.
  * The complexity of the method is O(n), in the worst case we have to traverse through the list
 * @param newNodo
 */
public  void insertarProducto(Producto producto) {
        if (producto==null) {
                System.out.println("The product is null, we can not insert it!!!");
                return; //exit the method
        }

        //a node is created to save this product
        DNode<Producto> newNodo=new DNode<Producto>(producto,null,null);
                if (size==0) {
                //the list is empty, so we can use the method insertFirst
                insertarFirst(newNodo);
        } else {
                //to see if we need to insert at the beginning of the list
                if (header.next.item.id>producto.id) insertarInicio(newNodo);
                else {
                        //to see if we have to insert in the final of the list
                        if (tailer.prev.item.id<producto.id) insertarAntes(tailer,newNodo);
                        else {
                                DNode<Producto> auxiliar=header.next;
                                //to traverse the list until the final or until the position is found
                                //where we should insert
                                while (auxiliar!=tailer && auxiliar.item.id<producto.id) {
                                        auxiliar=auxiliar.next;
                                }

                                /**we have to insert just before the auxiliary node * /
                                insertarAntes(auxiliar,newNodo);
                        }

                }

        }

}


/**
 * a method to traverse the list of products to show them
*/
public void showProductos() {
        DNode<Producto> auxiliar=header.next;
        / / to traverse through a list until the end or until you find the position n
        / / where we insert
        while (auxiliar!=tailer) {
                auxiliar.item.view();
                auxiliar=auxiliar.next;
        }
        System.out.println();
}

package sheet1.EJ07y08;
```

```
public class Producto {
int id;
String nombre;
float precio;

/**
 * Construct method
 * @param i
 * @param n
 * @param p
 */
public Producto(int i, String n, float p) {
        id=i;
        nombre=n;
        precio=p;
        }

/**
 * a method to show the information of a product.
 */
public void view() {
        System.out.println(id+"-"+nombre+":"+precio);
        }


}
```

E-8	
A palindrome is a word that can be read the same way in either direction, as example, the word “radar”. Design a Java method to determine if a given word is palindrome or not. The word is defined as a doubly linked list of characters.  NOTE: Use the list implemented in the exercise E-7 to solve this exercise. Don’t use the own data structures of Java such as (ArrayList, Vectors, LinkedList, etc.)	

```
/**
* to check if a word is palindrome or not
* @ paramlWord: gets a double-linked list of characters
* @return
*/
public  boolean esPalindroma() {
/ ** Node which helps us to explore the lsita from left to right * /
DNode<Character> fromLeft=header.next;
/ ** Node which helps us to explore the lsita from right to left* /
DNode<Character> fromRight=tailer.prev;
/**
* We stop when the left node has exceeded the right one or when the values are different
*/
while (fromRight.next!=fromLeft.prev && fromLeft.prev!=fromRight  && fromLeft.item==fromRight.item) {
fromLeft=fromLeft.next;
fromRight=fromRight.prev;
}
```

```java
/**
* If this condition is true, this means that the left node has exceeded the right one and therefore
* there is a value that is different.
*/
if (fromRight.next!=fromLeft.prev && fromLeft.prev!=fromRight) return false;
else return true;
}
```

E-9	

a) What is the difference between a simple list a circular list?

b) Write a Java class to perform the following:

- An method to create a circular list.
- An method to count the number of nodes in the circular list.

NOTE: don't use the own data structures to Java such as (ArrayList, Vectors, LinkedList, etc.)

The major difference between a circular list and a singly linked list is that: the last node in the circular list references (by its property next) to the first node in the list, however the last node in a singly linked list references to null.

```java
import sheet1.TADList.Node;

/**
 * Circular list implemented on a singly linked list with just one nodes centinel header
 * @author isegura
 *
 * @param <E>
 */
public class SListCircular<E> {
	private Node<E> header;
	private int size;

	/** Crea una lista vacia*/
	public SListCircular() {
		header=new Node<E>();
		header.next=null;
		size=0;
	}

	public boolean esVacia() {
		return (size==0);
		//return (header.next==null)
	}



	/**
	 * Devuelve el número nodos de la lista
	 * @return
	 */
	public int getSize() {
		return size;
	}
```

```
/**
 * Remove the first node
 * Complexity O(1).
 */
public void removeFirst() {
        if (esVacia()) {
                System.out.println("The list is empty!!!");
                return;
        }

        Node<E> oldFirst=header.next;
        //Now, header must be the following node in the list
        header.next=oldFirst.next;
        //decrease size
        size--;
}

/**
 * Inserts the newNodo at the beginning of the list
 * Complexity O(1)
 * @param newNodo
 */
public  void insertarFirst(Node<E> newNodo) {
        if (newNodo==null) {
                System.out.println("The node is null, we cannot insert it!!!");
                return;
        }

        Node<E> first=header.next;
        //Now, the first node must be the second one.
        newNodo.next=first;
        header.next=newNodo;

        //If the list is empty, the new first node should reference to header.
        if (first==null) newNodo.next=header;

        size++;

}

/**
 * Remove the last node in the list
 * We must traverse all nodes until to find the penultimum node (its next is null).
 * Complexity O(n), porque el número de operaciones depende del
 * tamaño de la lista (ya que debemos visitar sus n nodos).
 */
public void removeLast() {

        if (esVacia()) {
                System.out.println("The list is empty!!!");
                return; //exit the method
        }

        Node<E> aux=header.next;

        Node<E> penultimum=header;
        //traverse the list to find the last node
        while (aux.next!=header) {
                penultimum=aux;
                aux=aux.next;
```

```
		}


		if (penultimum==header) {
			header.next=null;//now the list is empty
		} else {
			//The list has two or more elements
			penultimum.next=header;
		}
		size--;


	}




	/**
	 *
	 * Inserts the newNodo at the end of the list
	 * Complexity O(1), since we only run a constant number of instructions
	 * @param newNodo
	 */
	public void insertarLast(Node<E> newNodo) {
		if (newNodo==null) {
			System.out.println("The node is null, we cannot insert it!!!");
			return;
		}
		if (esVacia()) insertarFirst(newNodo);
		else {
			Node<E> auxiliar=header.next;
			//The header node is pointing to the last node, therefore we must stop
			while (auxiliar.next!=header) {
				auxiliar=auxiliar.next;
			}
			// we reached the final node, its reference next
			// must be newNode
			auxiliar.next=newNodo;
			//newNodo.next must be header, because it is the final of the list
			newNodo.next=header;
			//increase the size
			size++;
		}
	}




	/**Trasverses the nodes in the list and shows them*/
	public void show() {
		if (esVacia()) {
			System.out.println("The list is empty");
			return;
		}

		Node<E> auxiliar=header.next;
		System.out.print(auxiliar.item+",");
		//We traverse the nodes until to return to the first node
		while (auxiliar.next!=header) {
			auxiliar=auxiliar.next;
			System.out.print(auxiliar.item+",");
		}
```

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


		/**
		 * Method to test the class
		 * @param args
		 */
		public static void main(String args[]) {
			SListCircular<Integer> lista=new SListCircular<Integer>();
			for (int i=0; i<5; i++) {
				Node<Integer> newNodo=new Node<Integer>(i);
				lista.insertarFirst(newNodo);
			}
			lista.show();
			for (int i=5; i<10; i++) {
				Node<Integer> newNodo=new Node<Integer>(i);
				lista.insertarLast(newNodo);
			}

			lista.show();

		}}
```

E-10

Given the operations shown in the table (run in the order in which they appear), what are the outputs and the contents of the stack after each operation is performed? Justify your answer.

Operation	*Output*	*content*	*Justification*
push(4)	-	4	Insert a node
push(1)	-	1, 4	Insert a node
pop()	1	4	Remove a node
push(6)	-	6, 4	Insert a node
pop()	6	4	Remove a node
top()	4	4	Remove a node
pop()	4	-	Last node
pop()	error	-	Stack is empty
isEmpty()	true	-	Stack is empty
Push(8)	-	8	Insert a node
Push(6)	-	6, 8	Insert a node
Push(1)	-	1, 6, 8	Insert a node
Push(4)	-	4, 1, 6, 8	Insert a node
Size()	4	4, 1, 6, 8	Size
Pop()	4	1, 6, 8	Remove a node
Push(9)	-	9, 1, 6, 8	Insert a node
Pop()	9	1, 6, 8	Remove a node
Pop()	1	6, 8	Remove a node

E-11

Given the recursive function ‘factorial ()’, write a Java class to simulate the growth and the decline of a stuck when the recursive calls are made.

```java
public static int factorialRec(int n) {
        stackCalls.push("factorial("+(n)+")");
        if (n<=0) {
                //stackCalls.pop();
                return 1;
        }
        else {
                int res=n*factorialRec(n-1);
                stackCalls.pop();

                //stackCalls.push("factorial("+(n-1)+")");
                for (int i=stackCalls.size()-1;i>=0;i--) System.out.println(stackCalls.get(i));
                System.out.println();
                return res;
        }
}
```

E-12

Write a Java program to use a stack for checking the parenthesis of an arithmetic expression as [(5 + x) - (y + z)], as the following:

1. If we find an open parenthesis [,(,{, then we insert it in the stack.
2. If we find a close parenthesis],},), then we consult the top of the stack. If they are of the same type then we remove the top of the stack.
   The arithmetic expression will be balanced if the stack is empty.

```java
public boolean checkParenthesis(char[] c) {
        LinkedListStack<Character> chars = new LinkedListStack<Character>();
        for(int i=0;i<c.length;i++){
                if (c[i]=='[' || c[i]=='(' || c[i]=='{'){
                                chars.push(new Character(c[i]));
                }else if (c[i]==']' || c[i]==')' || c[i]=='}'){
                        char aux = chars.top().charValue();
                        if ((c[i]==']' && aux=='[')||
                                (c[i]==')' && aux=='(')||
                                (c[i]=='}' && aux=='{')){
                                chars.pop();
                        }
                }
        }
        //Now we must check the list.
        if (chars.isEmpty()){
                return true;
        }else{
                return false;
        }
}
```

E-13

Write a Java program that receives a mathematical expression in postfix format and calculates the final result of the expression by using a stack. The program receives the mathematical expression as an array, assuming that this expression will always be well formulated and the operands values are between 0 and 9.
Example:
Postfix: **45 * 46** + /
Result of the program: **2**

```
public int postFixedFormat(char[] c) {
        LinkedListStack<Integer> ints = new LinkedListStack<Integer>();
        for(int i=0;i<c.length;i++){
                if (c[i]=='*'){
                   int v1 = ints.pop().intValue();
                   int v2 = ints.pop().intValue();
                        ints.push(new Integer(v2*v1));
                } else if (c[i]=='+') {
                   int v1 = ints.pop().intValue();
                   int v2 = ints.pop().intValue();
                        ints.push(new Integer(v2+v1));
                } else if (c[i]=='-') {
                   int v1 = ints.pop().intValue();
                   int v2 = ints.pop().intValue();
                        ints.push(new Integer(v2-v1));
                } else if (c[i]=='/') {
                   int v1 = ints.pop().intValue();
                   int v2 = ints.pop().intValue();
                        ints.push(new Integer(v2/v1));
                } else {
                        ints.push(new Integer(Character.digit(c[i],10)));
                }
        }
        return ints.pop();
}
```

E-14

The cooperative society TEIDE-HEASE needs a data structure to save and manage employees' data. For each employee, it needs to know the name, surname, DNI number, address and telephone. The data structure allows adding new professors and deleting employees when they are dismissed. The Human Resources Department has a policy to dismiss always the employee which has less time in the work.

Required:
Write a Java class to use the previous data structure and operations for the following algorithms:

- The algorithm **contratar(p)** to allow contracting new employees. Calculate the complexity of this algorithm. (P is a node of professor type).
- The algorithm **despedir()** to allow dismissing employees. Calculate the complexity of this algorithm.
- If the Human Resources Department modifies the dismissal policy of the centre by dismissing an employee using the DNI. Write a Java method for the new algorithm **despedir(dni),** and calculate the complexity of the new algorithm.

```
        public void hire(Worker e){
                workers.push(e);
        }

        public Worker fire(){
                if(workers.isEmpty()){
                        System.out.println("There is no worker at the enterprise");
                        return null;
                }
                return workers.pop();
        }

        public Worker fireByNIC(String nic){
                if(workers.isEmpty()){
                        System.out.println("There is no worker at the enterprise");
                        return null;
                }

                Worker aux = null;
                LinkedListStack<Worker> pilaAux = new LinkedListStack<Worker>();
                boolean found = false;

                while(!workers.isEmpty() || !found){
                        aux = workers.pop();
                        if(aux.nic.equals(nic)){
                                found = true;
                        }
                        else {
                                pilaAux.push(aux);
                        }
                }

                if(!found){
                        System.out.println("There is no worker with this NIC");
                        return null;
                }

                while(!pilaAux.isEmpty()){
                        workers.push(pilaAux.pop());
                }

                return aux;
        }


public class Worker {

        String name;
        String surname;
        String nic;
        String address;
        String phonenumber;

        public Worker(String name, String surname, String nic, String address,
                        String phonenumber) {
                super();
                this.name = name;
                this.surname = surname;
                this.nic = nic;
                this.address = address;
                this.phonenumber = phonenumber;
        }
}
```

E-15

Given the operations shown in the table (run in the order in which they appear), what are the outputs and the contents of the queue after each operation is performed? Justify your answer.

Operation	*Output*	*content*	*Justification*
enqueue(4)	-	4	Insert a node
enqueue(2)	-	2, 4	Insert a node
dequeue()	4	2	Remove a node
enqueue(8)	-	8, 2	Insert a node
dequeue()	2	8	Remove a node
front()	8	8	See the front
dequeue()	8	-	Remove a node
dequeue()	error	-	Remove a node
isEmpty()	true	-	Queue is empty
enqueue(6)	-	6	Insert a node
enqueue(8)	-	8, 6	Insert a node
size()	2	8, 6	size
enqueue(2)	-	2, 8, 6	Insert a node
enqueue(4)	-	4, 2, 8, 6	Insert a node
dequeue()	6	4, 2, 8	Remove a node

E-16

Let 'cola1' and 'cola2' are two queues, the elements are inserted in these two queues in an orderly manner. For example, cola1 = [2,3,6,8,9], cola2 = [0,1,4,5,7].

Required:

a) Write a Java class for an iterative algorithm that combines both queues in another ordered queue 'cola3':

b) Calculate and justify briefly the complexity of the algorithm.

```
    public LinkedListQueue<Integer> mixQueueIterative(LinkedListQueue<Integer> c1,

    LinkedListQueue<Integer> c2){
        LinkedListQueue<Integer> aux = new LinkedListQueue<Integer>();
        while(!c1.isEmpty() || !c2.isEmpty()){
            if(c1.isEmpty()){
                aux.enqueue(c2.dequeue());
            }
            else if(c2.isEmpty()){
                aux.enqueue(c1.dequeue());
            }
            else{
                if(c1.front().intValue()>c2.front().intValue()){
```

```
                                        aux.enqueue(c2.dequeue());
                                }
                                else{
                                        aux.enqueue(c1.dequeue());
                                }
                        }
                }
                return aux;
        }
```

E-17

Rewrite the solution of the previous exircise using recursive algorithm.

What do you think about the complexity of the recursive algorithm, is better, worse or the same as the iterative one? Briefly justify your answer.

```
        public LinkedListQueue<Integer> mixQueueRecursive(LinkedListQueue<Integer> c1,

                LinkedListQueue<Integer> c2){
                LinkedListQueue<Integer> aux = null;
                if(c1.isEmpty() && c2.isEmpty()){
                        aux = new LinkedListQueue<Integer>();
                        return aux;
                }
                Integer number;
                if(c1.isEmpty()){
                        number = c2.dequeue();
                        aux = mixQueueRecursive(c1, c2);
                        aux.enqueue(number);
                }
                else if(c2.isEmpty()){
                        number = c1.dequeue();
                        aux = mixQueueRecursive(c1, c2);
                        aux.enqueue(number);
                }
                else{
                        if(c1.front()>c2.front()){
                                number = c2.dequeue();
                                aux = mixQueueRecursive(c1, c2);
                                aux.enqueue(number);
                        }
                        else{
                                number = c1.dequeue();
                                aux = mixQueueRecursive(c1, c2);
                                aux.enqueue(number);
                        }
                }
                return aux;
        }
```

E-18

Write a Java class of a queue Cola and use this queue to perform the following algorithm:

The algorithm is used to eliminate the element that occupies the position k, return the last element in the queue.

For example, with k=3, for the queue Cola is processed as follows:

Cola =	1	2	3	4	5	6	7	8	9

dequeue(1); enqueue(1); dequeue(2); enqueue(2); dequeue(3);

Cola =	4	5	6	7	8	9	1	2

dequeue(4); enqueue(4); dequeue(5); enqueue(5); dequeue(6);

Cola =	7	8	9	1	2	4	5

dequeue(7); enqueue(7); dequeue(8); enqueue(8); dequeue(9);

Cola =	1	2	4	5	7	8

dequeue(1); enqueue(1); dequeue(2); enqueue(4); dequeue(4);

Cola =	5	7	8	1	2

dequeue(5); enqueue(5); dequeue(7); enqueue(7); dequeue(8);

Cola =	1	2	5	7

dequeue(1); enqueue(1); dequeue(2); enqueue(2); dequeue(5);

Cola =	7	1	2

dequeue(7); enqueue(7); dequeue(1); enqueue(1); dequeue(2);

Cola =	7	1

dequeue(7); enqueue(7); dequeue(1); enqueue(1); dequeue(7);

Cola =	1

dequeue(1); enqueue(1); dequeue(1); enqueue(1); dequeue(1);

Cola =	

Return (1)

E-19

Given two stacks S and T (not empty), and a deque D (Double-Ended Queues). Write a Java program to use D to save the elements of T below the elements of S.

NOTE: For more information on Double-Ended Queues see the Goodrich's book page 213.

```
        public int josephus(int x) {
```

```
        int n=0;
        while (!queue.isEmpty()){
            for (int i = 0; i<x; i++){
                queue.enqueue(queue.dequeue());
            }
            n = queue.dequeue();
            show(queue);
        }
        return n;
    }
```

E-20	

Write a Java class to show how to use a stuck and a queue to generate all the possible subset of the set T, without recursion.

For example; if T={A,B,C}, then all the possible subset of T are:
{}, {A}, {B}, {C}, {A,B}, {A,C}, {B,C}, y {A,B,C}.

```
    public static void mount(LinkedListStack<Integer> s, LinkedListStack<Integer> t) {

        if (s.isEmpty()||t.isEmpty()) return;

    DQueue<Integer> D=new DQueue<Integer>();


    while (!s.isEmpty()) {

        DNode<Integer> obj=new DNode<Integer>(s.pop());

        D.insertLast(obj);
        }
        System.out.print("D ");
        D.show();

        while (!t.isEmpty()) {

            DNode<Integer> obj=new DNode<Integer>(t.pop());

            D.insertLast(obj);

        }
    System.out.print("D ");

    D.show();

    System.out.println();
    while (!D.isEmpty()) {
    //D.show();
    s.push(D.removeLast());
    }
    System.out.println("S ");
    while (!s.isEmpty()) {
        System.out.println(s.pop());
        }
    }
```