 **Grado en Ingeniería Informática**	**Asignatura Estructura de Datos y Algoritmos** **Junio de 2015** **Convocatoria Extraordinaria**

Nombre: .....
Apellidos: .....
Grupo: .....

LEA ATENTAMENTE ESTAS INSTRUCCIONES ANTES DE COMENZAR LA PRUEBA

1. Es necesario poner todos los datos del alumno en el cuadernillo de preguntas (este documento) y en las hojas de cuadros. Use un bolígrafo para rellenarlos. No se corregirá nada que esté escrito a lapicero.
2. Cuando finalice la prueba, se deben entregar el enunciado del examen y cualquier hoja que haya empleado.
3. No está permitido salir del aula por ningún motivo hasta la finalización del examen. Desconecten los móviles durante el examen.
4. **En este examen, excepto en los apartados donde indique lo contrario no está permitido usar las clases de la librería EdaLib y tampoco las clases que proporciona el api de java (ArrayList, LinkedList, etc). Tampoco está permitido utilizar Arrays. Es decir, en tus soluciones debes implementar tus propias estructuras de datos.**

NO PASE DE ESTA HOJA hasta que se le indique el comienzo del examen

Problema 1 (4 puntos). Dada la siguiente clase:

```
public class Contacto {
      protected String nombre;
      protected String email;
}
```

Nota: Es recomendable que leas todos los apartados de este ejercicio antes de decidir la estructura lineal más adecuada.

Se pide lo siguiente:

a) (0.5 puntos) Implementar una **estructura lineal,** Agenda, que permita almacenar los contactos de una agenda. En este apartado SÓLO tienes que escribir la(s) cabecera(s) de la(s) clase(s), atributos y constructores (si los consideras necesarios) necesarios para implementar una estructura lineal. No debes añadir ningún método por el momento. **Justifica la elección del tipo de estructura lineal elegido para implementar tu solución**

```
public class AgencyNode {
    Contact elem;

    AgencyNode previousNode;
    AgencyNode nextNode;

    public AgencyNode(Contact cont) {
        elem=cont;
    }

}
```

```
public class Agency {

    protected AgencyNode header;
    protected AgencyNode trailer;

    public Agency() {
        header = new AgencyNode(null);
        trailer = new AgencyNode(null);
        header.nextNode = trailer;
        trailer.previousNode = header;
    }
```

b) Implementar los siguiente métodos:

i. (1.5 puntos) Insertar por orden alfabético (ascendente). Este método recibe un nombre y un email y almacena dicho contacto en la agenda según el orden alfabético del nombre. No se permiten duplicados.

```
/**
 * Exercise 1.b - insert in order  O(n)
 */
public void insert(String name, String email) {
    Contact c = new Contact(name, email);
    AgencyNode newNode = new AgencyNode(c);

    AgencyNode nodeIt = header.nextNode;
    while (nodeIt != trailer &&
            name.compareTo(nodeIt.getElement().getName())>0) {
        nodeIt=nodeIt.nextNode;
    }

    if (nodeIt==trailer ||
            name.compareTo(nodeIt.getElement().getName())<0) {
        nodeIt.previousNode.nextNode = newNode;
        newNode.previousNode = nodeIt.previousNode;
        nodeIt.previousNode = newNode;
        newNode.nextNode = nodeIt;
    } else {
        System.out.println("This name already exists.");
    }

}
```

ii. (1.5 puntos) Mostrar el contenido de la agenda en orden alfabético. Este método recibe un parámetro, de tipo char, indicando el tipo de

orden, y en función de este parámetro, muestra los contactos en orden ascendente ('A') o descendente ('D').

```
/**
 * Exercise 1.b.2 - print in order O(n)
 */
public void print(char orden) {
    if (orden == 'A') {
        System.out.println("AGENCY (ascending order):");
        for (AgencyNode nodeIt = header.nextNode; nodeIt != trailer;
                nodeIt = nodeIt.nextNode) {
            Contact obj=nodeIt.elem;
            System.out.println(obj.name+"; " + obj.email);
        }
    } else if (orden == 'D') {
        System.out.println("AGENCY (descending order):");
        for (AgencyNode nodeIt = trailer.previousNode; nodeIt != header;
                nodeIt = nodeIt.previousNode) {
            Contact obj=nodeIt.elem;
            System.out.println(obj.name+"; " + obj.email);
        }
    } else {
        System.out
                .println("Please, insert 'A' to print the contacts in "
                        + "ascending order and 'D' to print them in descending order. "
                        + "Other letters are not accepted in this method");
    }
}
```

c) (0.5 puntos) ¿Cuál es la complejidad de cada uno de los métodos del apartado b)?.
O(n).

Nota: Recuerda que no puedes usar EdaLib, y que por tanto, deberás implementar cualquier método auxiliar necesario para tu solución

Problema 2 (5 puntos). Teniendo en cuenta la clase Contacto del ejercicio anterior, se pide resolver los siguientes apartados:

a) (0.50 puntos) Implementa una clase que permita almacenar los contactos en un árbol binario de búsqueda (ABB). Esta estructura permitirá almacenar los contactos por orden alfabético (ordenados por nombre) y sin permitir duplicados. Si es necesario, modifica la clase Contacto para incluir nuevos atributos relacionados con la estructura, que te permitan implementar la solución. En este apartado SÓLO tienes que escribir la cabecera(s) de la(s) clase(s), atributos y constructores (si los consideras necesarios). Es decir, no debes añadir ningún método por el momento. ¿Qué ventajas ofrece un ABB frente a una estructura lineal a la hora de almacenar los contactos en orden alfabético?..

```java
public class Contact {
    protected String name; //it will be the key
    protected String email;


    protected Contact parent;
    protected Contact leftChild;
    protected Contact rightChild;

    public Contact(String name, String email){
        this.name = name;
        this.email = email;
    }

    public String toString(){
        return "Contact: "+name+"; "+email;
    }
```

```java
public class AgencyBST {

    protected Contact root;
```

ABB permite almacenar los contactos y que las operaciones de búsqueda, inserción y borrado tengan un complejidad logarítmica, mientras que en una estructura lineal, la complejidad de estas operaciones (en el peor de los casos) es lineal.

b) (1.25 puntos) Implementar un método recursivo para guardar un contacto.

```
public void insert(String name, String email) {
    Contact newNode = new Contact(name, email);
    if (root == null) {
        root = newNode;
        return;
    }
    insert(newNode, root);
}

private void insert(Contact newNode, Contact current) {
    String name = newNode.name;
    if (name.compareTo(current.name) == 0) {
        System.out.println("Cannot insert, the name already exists");
        return;
    }
    if (name.compareTo(current.name) < 0) {
        if (current.leftChild == null) {
            current.leftChild = newNode;
            newNode.parent = current;
            return;
        } else {
            insert(newNode, current.leftChild);
        }
    } else {
        if (current.rightChild == null) {
            current.rightChild = newNode;
            newNode.parent = current;
            return;
        } else {
            insert(newNode, current.rightChild);
        }
    }
}
```

c) (1.25 puntos) Implementar un método recursivo que muestre los contactos por orden alfabético.

```java
/**
 * Exercise 2c - showContacts O(n)
 * */
public void showContacts() {
    printInOrder(root);
}

 void printInOrder(Contact node) {
    if (node == null)
        return;
    printInOrder(node.leftChild);
    System.out.println(node.toString());
    printInOrder(node.rightChild);
}
```

d) (2 puntos) Implementar un método iterativo que permita recorrer por niveles el ABB que almacena los contactos. En este caso si pueden usar la clase SQueue<E> de EdaLib o cualquier clase que implemente una cola.

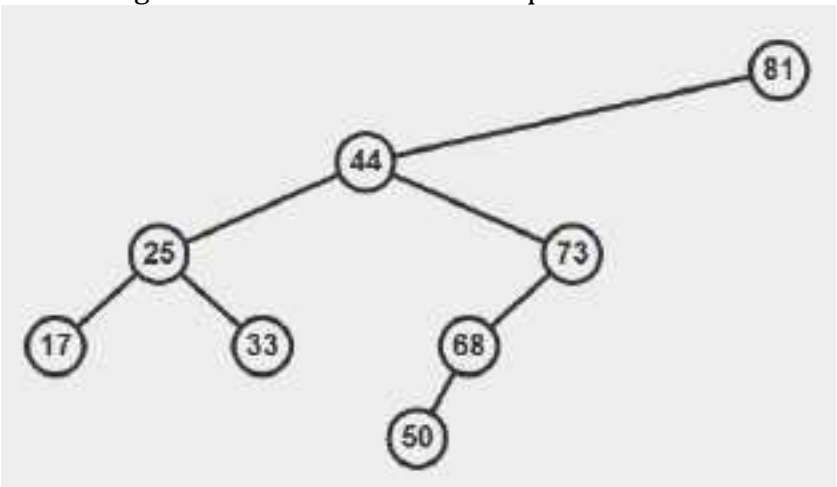
```java
/**
 * Exercise 2d - printLevelOrder O(n)
 * */
public void printLevelOrder() {
    System.out.println("***************LEVEL_ORDER:***************");
    printLevelOrder(root);
}


void printLevelOrder(Contact node) {
    if (node != null) {
        SQueue<Contact> q = new SQueue<Contact>();
        q.enqueue(node);

        while (!q.isEmpty()) {
            Contact n = q.dequeue();
            System.out.println(n.toString());
            if (n.leftChild != null)
                q.enqueue(n.leftChild);
            if (n.rightChild != null)
                q.enqueue(n.rightChild);
        }
    }
}
```

Problema 3 (1 puntos).

Dado el siguiente árbol binario de búsqueda.



Se pide:

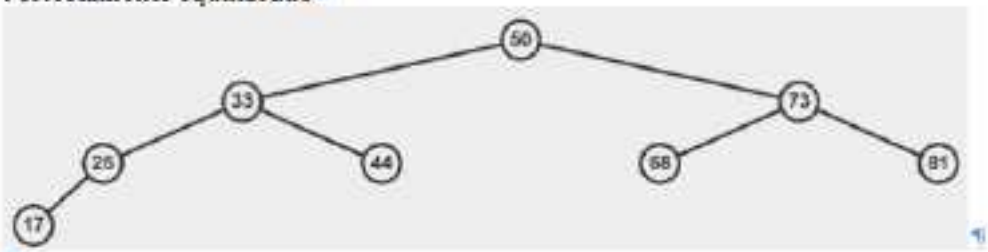
a) Versión perfectamente equilibrada del árbol (equilibrio en tamaño). (0.50)
b) Versión AVL (equilibrio en altura). (0.50)

Nota: La solución debe incluir cada uno de los pasos que se han realizado hasta llegar al resultado final.

Solución:

Solución:

Perfectamente equilibrado



AVL