Grado en Ingeniería Informática y Doble Grado en Informática y Administración de Empresas	Asignatura Estructura de Datos y Algoritmos **EXAMEN FINAL, CONVOCATORIA ORDINARIA**

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). Use un bolígrafo para rellenarlos.
2. El examen está compuesto por 5 Preguntas.
3. Solamente se evaluará la contestación en este cuadernillo de preguntas.
4. Cuando finalice la prueba, se deben entregar el enunciado del examen y cualquier hoja que haya empleado.
5. No está permitido salir del aula por ningún motivo hasta la finalización del examen.
6. Desconecten los móviles durante el examen.
7. La duración del examen es de **2 horas**.
8. **Todos los métodos necesarios de las librerías EdaLib son públicos y para simplificar la implementación de sus soluciones puede considerar que los atributos de las clases son públicos.**

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

Problema 1 (2.5 puntos).

a) (0.5 punto) Crea una clase que implemente una lista de caracteres. La lista debe ser una lista simplemente enlazada.

b) (2 punto) Añade un método a la clase, **eliminaConsecutivas()**, que recorre la lista eliminando los caracteres consecutivos que estén repetidos. El método debe ser un método de objeto, es decir, no debe ser estático.

Entrada: k,p,a,a,a,a Salida: k,p,a Entrada: e,e,a,a Salida: e,a

Nota: No se puede utilizar arrays, ArrayList, LinkedList. Sólo está permitido utilizar las clases SList y SNode.

Solución:

a) `public class ListaCaracteres extends SList<Character> {...}`

b)

```
public void eliminaConsecutivas() {
                SNode<Character> nodeIt=this.firstNode;
                SNode<Character> nodeAnt=null;

                while (nodeIt!=null) {
                        char c=nodeIt.getElement();
                        if (nodeAnt!=null) {
                                int antChar=nodeAnt.getElement();
                                if (antChar==c) {
                                        //eliminamos el nodoIt
                                        nodeAnt.nextNode=nodeIt.nextNode;
                                        nodeIt=nodeIt.nextNode;
                                } else {
                                        nodeAnt=nodeIt;
                                        nodeIt=nodeIt.nextNode;
                                }
                        } else {
                                nodeAnt=nodeIt;
                                nodeIt=nodeIt.nextNode;
                        }

                }
        }
```

Problema 2 (2 puntos)

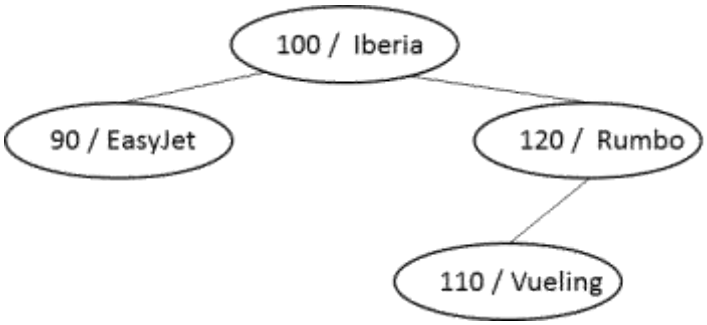
Escribe un método estático que reciba un nodo de árbol binario y devuelva en una estructura lineal. La lista que devuelve el método debe contener los elementos de los nodos visitados en el recorrido por niveles del árbol cuya raíz es el nodo que se pasa como argumento.

```
public static<E> IList<E> getLevelOrder(BinTreeNode<E> node) {


}
```

Solución:

```
public static<E> IList<E> getLevelOrder(BinTreeNode<E> node) {
		if (node==null) return null;

		//esta es la lista en la que almacenaré los elementos visitados
		SList<E> list=new SList<E>();

		//En esta cola vamos a ir guardando los nodos que visito
		//Es una cola de nodos.
		SQueue<BinTreeNode<E>> queue=new SQueue<BinTreeNode<E>>();
		queue.enqueue(node);

		while (!queue.isEmpty()) {
			BinTreeNode<E> visit=queue.dequeue();
			list.addLast(visit.getElement());

			if (visit.getLeftChild()!=null)
				queue.enqueue(visit.getLeftChild());
			if (visit.getRightChild()!=null)
				queue.enqueue(visit.getRightChild());
		}

		return list;
	}
```

Problema 3 (2,5 puntos) Una plataforma Web de búsqueda de vuelos económicos utiliza Árboles Binarios de Búsqueda (ABB o BST en inglés) para almacenar los precios de los vuelos y qué compañía oferta ese vuelo. Un ejemplo de búsqueda de vuelos a Londres podría devolver el siguiente árbol binario de búsqueda[1], por ejemplo:



a) **(0,5 puntos)** Dibuja el árbol resultante de insertar las siguientes ocurrencias en orden sobre el ABB que aparece en la figura anterior:
   - <105, Logitravel>
   - <125, Norwegian>

[1] Datos no reales (ejemplo totalmente inventado)

- <107, Travelgenio>
- <117, Expedia>
- <85, RyanAir>
- <95, Atrapalo>

Solución:

b) (0,5 puntos) Escribe la cabecera de la clase "ArbolVuelos" que permite almacenar este árbol binario de búsqueda. Para ello, será necesario hacer uso de la librería sobre Árboles Binarios de Búsqueda proporcionado durante el curso (en particular, la librería BSTree). Los precios serán de tipo "Integer" y las compañías se definirán como "String".

Solución:

```
public class ArbolVuelos extends BSTree<Integer, String> {

}
```

c) (1,5 puntos) Implementa el algoritmo **recursivo** que reciba un nodo, un precio mínimo y un precio máximo y devuelva en una lista las compañías más económicas para este viaje en orden (la más barata la primera) siempre que tengan ofertas en el rango de precios indicado.
Este algoritmo estará formado por dos métodos con las siguientes cabeceras. Se pide: escribe el código de estos métodos.

```
public IList<String> devuelveCompanyasBaratas (int precioMin, int precioMax){




}
```

```
public static void devuelveCompanyasBaratas(BSTNode<Integer, String> nodo,
                  int precioMin, int precioMax, IList<String> lista) {
```

}

Solución:

```
public IList<String> devuelveCompanyasBaratas (int precioMin, int precioMax){
    IList<String> lista = new SList<String>();
    devuelveCompanyasBaratas(root, precioMin, precioMax, lista);
    return lista;
}
```

```java
/**
 * Algoritmo RECURSIVO para recorrer el árbol en recorrido IN-ORDEN
 * (ordenado por precios)
 *
 * @param precioMin
 *            --> precio mínimo
 * @param precioMax
 *            --> precio máximo
 * @param lista
 *            --> lista de compañías ordenadas de precios más baratos a
 *            precios más caros (siempre en el rango de precios indicados).
 *            Se actualizará poco a poco en el algoritmo recursivo
 */
public static void devuelveCompanyasBaratas(BSTNode<Integer, String> nodo,
        int precioMin, int precioMax, IList<String> lista) {

    // Si el nodo es vacio, acaba la recursión
    if (nodo != null) {

        // realiza recorrido IN-ORDEN para devolver los precios de forma
        // ordenada
        // (hijo izquierdo) --> solo si puedo encontrar valores en rango
        if (nodo.getKey() > precioMin)
            devuelveCompanyasBaratas(nodo.getLeftChild(), precioMin,
                    precioMax, lista);

        // (nodo)
        if ((nodo.getKey() >= precioMin) && (nodo.getKey() <= precioMax))
            lista.addLast(nodo.getElement());

        // (hijo derecho) --> solo si puedo encontrar valores en rango
        if (nodo.getKey() < precioMax)
            devuelveCompanyasBaratas(nodo.getRightChild(), precioMin,
                    precioMax, lista);
    }//if

}//devuelveCompanyasBaratas
```

Problema 4 (2 punto). Para cada afirmación, indica si es verdadera o falsa, y razona brevemente tu respuesta.

a) Para que un método recursivo funcione correctamente se debe cumplir que exista al menos un caso base que pare la recursión y al menos un caso recursivo que llama al mismo método.

Solución: V

b) Para comparar algoritmos se utiliza la complejidad temporal como una medida del grado de dificultad del algoritmo. Ésta indica el tiempo de ejecución de un algoritmo en función, entre otras cosas, del tamaño de los datos entrada.

Solución: V

c) Para poder buscar un elemento concreto en un TAD Pila donde sus elementos están ordenados se puede utilizar un algoritmo de búsqueda dicotómica (o binaria), donde primero se accederá al elemento intermedio mediante el método del TAD denominado *getAt(size/2)*. Posteriormente, dependiendo de si es menor o mayor, se buscará de

nuevo el elemento en la parte superior o inferior.

Solución: F

d) En un TAD lineal Array los elementos se almacenan en posiciones consecutivas de memoria por lo que el acceso a los elementos del TAD es directo.

Solución: V

e) El TAD Cola en ningún caso se podría implementar mediante Arrays.

Solución: F

f) El algoritmo para obtener el tamaño de una lista de elementos es igual de eficiente, es decir, tiene la misma complejidad temporal, si se implementa mediante un recorrido sobre una lista simplemente enlazada o recorriendo los elementos de un array.

Solución: V

g) Eliminar el final de una lista doblemente enlazada (el último elemento) es más eficiente, es decir, tiene una mejor complejidad temporal, que eliminar el final de una lista simplemente enlazada.

Solución: V

h) La búsqueda dicotómica (o binaria) sobre un Array de elementos tiene complejidad cuadrática.

Solución: F

i) Es posible definir un algoritmo para invertir los elementos de un array usando una pila cuya complejidad sea lineal, O(n)

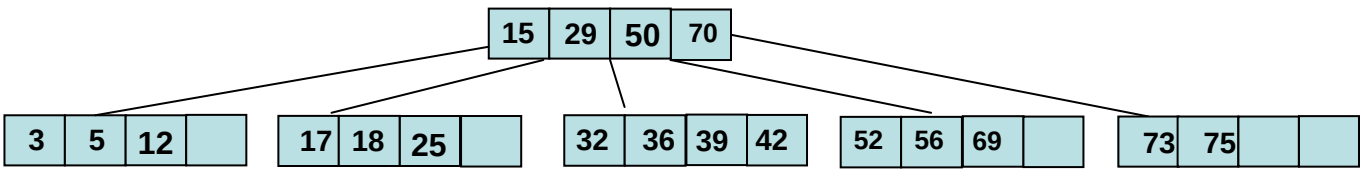
Solución: V

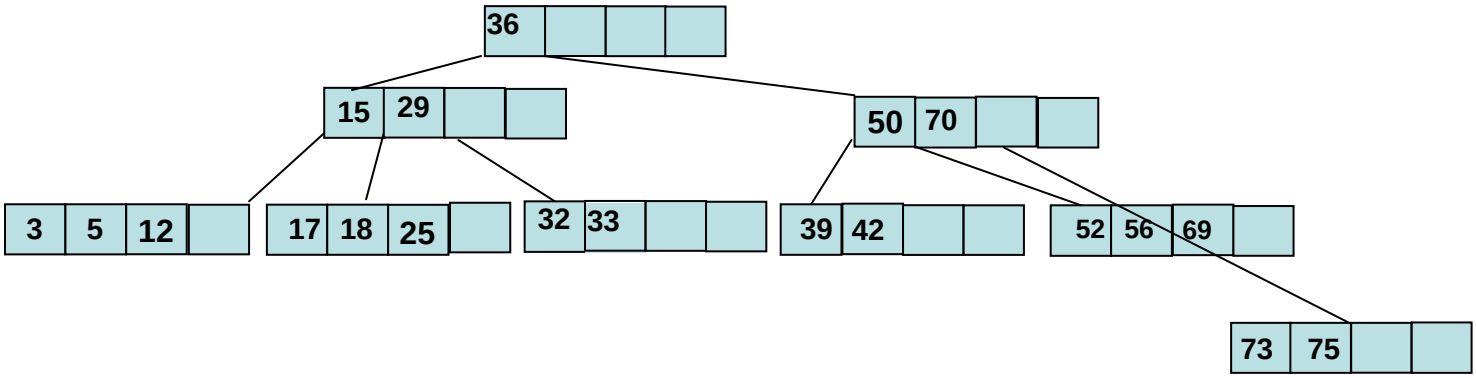
j) Un algoritmo para recorrer una pila de enteros eliminando solo los elementos pares requiere obligatoriamente el uso de una estructura de datos auxiliar para almacenar los elementos que se extraen de la cola y que contienen un número impar.

Solución: F

Problema 5 (1 punto). Para los siguientes árboles B, conteste a los siguientes apartados

a) (0.5 puntos) Para el siguiente árbol B de orden 4, dibuja el árbol resultante de insertar la clave **33**.





b) (0.5 puntos) Para el siguiente árbol B de orden 4, dibuja el árbol resultante de borrar la clave **33**.