



Entregue este folio con sus datos consignados

Alumno:

Identificación:

C. Asociado en que realizó la Práctica Obligatoria:

P1 (1'5 puntos) **Práctica.** Explíquense y justifíquense cuáles serían las diferencias entre utilizar como representación interna del tipo una lista o un árbol binario para la torre de control.

1. (3 puntos) Implemente un algoritmo que reciba una lista de enteros, y haciendo uso de solamente un árbol binario de búsqueda con equilibrado AVL, devuelva una lista con el mismo contenido que la primera pero con los elementos en orden creciente. El algoritmo debe tener un coste $\mathcal{O}(n \log n)$ y no debe implementar ni la ordenación por inserción, ni por mezcla. La cabecera en Java del algoritmo que se pide es:

```
ListIF<int> ordenar (ListIF<int> datos)
```

2. Un conjunto es una colección de datos de capacidad ilimitada cuyos elementos no pueden repetirse y donde no existe un orden establecido. Las operaciones que pueden aplicarse sobre los conjuntos aparecen definidas en la siguiente interfaz:

SetIF

```
// Representa un conjunto de elementos
public interface SetIF<T>{

    // añade un elemento al conjunto
    public void add (T e);

    // elimina un elemento del conjunto
    public void remove (T e);

    // devuelve cierto si el elemento pertenece al conjunto
    public boolean contains (T e);

    // devuelve la unión con el conjunto set (@param)
    public SetIF<T> union (SetIF<T> set);

    // devuelve la intersección con el conjunto set (@param)
    public SetIF<T> intersection (SetIF<T> set);

    // devuelve la diferencia simétrica con el conjunto set (@param)
    public SetIF<T> difference (SetIF<T> set)
}
```

- a) (0'5 puntos) Describa detalladamente cómo realizaría la representación interna de este tipo (usando los TAD estudiados en la asignatura). Justifique su elección
- b) (4'5 puntos) Basándose en la respuesta anterior, implemente todos los métodos de la interfaz **SetIF<T>**
- c) (0'5) ¿Qué coste asintótico temporal en el caso peor tiene el método de adición (add) en su implementación?

ListIF (Lista)

```

/* Representa una lista de elementos */
public interface ListIF<T>{
    /* Devuelve la cabeza de una lista*/
    *
    public T getFirst ();
    /* Devuelve: la lista excluyendo la cabeza. No
       modifica la estructura */
    public ListIF<T> getTail ();
    /* Inserta una elemento (modifica la estructura)
       * Devuelve: la lista modificada
       * @param elem El elemento que hay que añadir*/
    public ListIF<T> insert (T elem);
    /* Devuelve: cierto si la lista esta vacia */
    public boolean isEmpty ();
    /* Devuelve: cierto si la lista esta llena*/
    public boolean isFull();
    /* Devuelve: el numero de elementos de la lista*/
    public int getLength ();
    /* Devuelve: cierto si la lista contiene el
       elemento.
       * @param elem El elemento buscado */
    public boolean contains (T elem);
    /* Ordena la lista (modifica la lista)
       * @Devuelve: la lista ordenada
       * @param comparator El comparador de elementos*/
    public ListIF<T> sort (ComparatorIF<T>
        comparator);
    /*Devuelve: un iterador para la lista*/
    public IteratorIF<T> getIterator ();
}

```

StackIF (Pila)

```

/* Representa una pila de elementos */
public interface StackIF <T>{
    /* Devuelve: la cima de la pila */
    public T getTop ();
    /* Incluye un elemento en la cima de la pila
       (modifica la estructura)
       * Devuelve: la pila incluyendo el elemento
       * @param elem Elemento que se quiere añadir */
    public StackIF<T> push (T elem);
    /* Elimina la cima de la pila (modifica la
       estructura)
       * Devuelve: la pila excluyendo la cabeza */
    public StackIF<T> pop ();
    /* Devuelve: cierto si la pila esta vacia */
    public boolean isEmpty ();
    /* Devuelve: cierto si la pila esta llena */
    public boolean isFull();
    /* Devuelve: el numero de elementos de la pila */
    public int getLength ();
    /* Devuelve: cierto si la pila contiene el
       elemento
       * @param elem Elemento buscado */
    public boolean contains (T elem);
    /*Devuelve: un iterador para la pila*/
    public IteratorIF<T> getIterator ();
}

```

QueueIF (Cola)

```

/* Representa una cola de elementos */
public interface QueueIF <T>{
    /* Devuelve: la cabeza de la cola */
    public T getFirst ();
    /* Incluye un elemento al final de la cola
       (modifica la estructura)
       * Devuelve: la cola incluyendo el elemento
       * @param elem Elemento que se quiere añadir */
    public QueueIF<T> add (T elem);
    /* Elimina el principio de la cola (modifica la
       estructura)
       * Devuelve: la cola excluyendo la cabeza */
    public QueueIF<T> remove ();
    /* Devuelve: cierto si la cola esta vacia */
    public boolean isEmpty ();
}

```

```

/* Devuelve: cierto si la cola esta llena */
public boolean isFull();
/* Devuelve: el numero de elementos de la cola */
public int getLength ();
/* Devuelve: cierto si la cola contiene el
   elemento
   * @param elem elemento buscado */
public boolean contains (T elem);
/*Devuelve: un iterador para la cola*/
public IteratorIF<T> getIterator ();
}

```

TreeIF (Árbol general)

```

/* Representa un arbol general de elementos */
public interface TreeIF <T>{
    public int PREORDER = 0;
    public int INORDER = 1;
    public int POSTORDER = 2;
    public int BREADTH = 3;
    /* Devuelve: elemento raiz del arbol */
    public T getRoot ();
    /* Devuelve: lista de hijos de un arbol.*/
    public ListIF <TreeIF <T>> getChildren ();
    /* Establece el elemento raiz.
       * @param elem Elemento que se quiere poner como
       raiz*/
    public void setRoot (T element);
    /* Inserta un subarbol como ultimo hijo
       * @param child el hijo a insertar*/
    public void addChild (TreeIF<T> child);
    /* Elimina el subarbol hijo en la posicion
       index-esima
       * @param index indice del subarbol comenzando en
       0*/
    public void removeChild (int index);
    /* Devuelve: cierto si el arbol es un nodo hoja*/
    public boolean isLeaf ();
    /* Devuelve: cierto si el arbol es vacio*/
    public boolean isEmpty ();
    /* Devuelve: cierto si la lista contiene el
       elemento
       * @param elem Elemento buscado*/
    public boolean contains (T element);
    /* Devuelve: un iterador para la lista
       * @param traversalType el tipo de recorrido, que
       * sera PREORDER, POSTORDER o BREADTH */
    public IteratorIF<T> getIterator (int
        traversalType);
}

```

BTreeIF (Árbol Binario)

```

/* Representa un arbol binario de elementos */
public interface BTreeIF <T>{
    public int PREORDER = 0;
    public int INORDER = 1;
    public int POSTORDER = 2;
    public int LRBREADTH = 3;
    public int RLBREADTH = 4;
    /* Devuelve: el elemento raiz del arbol */
    public T getRoot ();
    /* Devuelve: el subarbol izquierdo o null si no
       existe */
    public BTreeIF <T> getLeftChild ();
    /* Devuelve: el subarbol derecho o null si no existe
       */
    public BTreeIF <T> getRightChild ();
    /* Establece el elemento raiz
       * @param elem Elemento para poner en la raiz */
    public void setRoot (T elem);
    /* Establece el subarbol izquierdo
       * @param tree el arbol para poner como hijo
       izquierdo */
    public void setLeftChild (BTreeIF <T> tree);
    /* Establece el subarbol derecho
       * @param tree el arbol para poner como hijo derecho
       */
}

```

```

    public void setRightChild (BTreeIF <T> tree);
    /* Borra el subarbol izquierdo */
    public void removeLeftChild ();
    /* Borra el subarbol derecho */
    public void removeRightChild ();
    /* Devuelve: cierto si el arbol es un nodo hoja*/
    public boolean isLeaf ();
    /* Devuelve: cierto si el arbol es vacio */
    public boolean isEmpty ();
    /* Devuelve: cierto si el arbol contiene el elemento
    * @param elem Elemento buscado */
    public boolean contains (T elem);
    /* Devuelve un iterador para la lista.
    * @param traversalType el tipo de recorrido que
    sera
    PREORDER, POSTORDER, INORDER, LRBREADTH o
    RLBREADTH */
    public IteratorIF<T> getIterator (int
    traversalType);
}

```

ComparatorIF

```

/* Representa un comparador entre elementos */
public interface ComparatorIF<T>{
    public static int LESS = -1;
    public static int EQUAL = 0;
    public static int GREATER = 1;
    /* Devuelve: el orden de los elementos
    * Compara dos elementos para indicar si el primero
    es
    * menor, igual o mayor que el segundo elemento

```

```

    * @param e1 el primer elemento
    * @param e2 el segundo elemento */
    public int compare (T e1, T e2);
    /* Devuelve: cierto si un elemento es menor que otro
    * @param e1 el primer elemento
    * @param e2 el segundo elemento */
    public boolean isLess (T e1, T e2);
    /* Devuelve: cierto si un elemento es igual que otro
    * @param e1 el primer elemento
    * @param e2 el segundo elemento */
    public boolean isEqual (T e1, T e2);
    /* Devuelve: cierto si un elemento es mayor que otro
    * @param e1 el primer elemento
    * @param e2 el segundo elemento*/
    public boolean isGreater (T e1, T e2);
}

```

IteratorIF

```

/* Representa un iterador sobre una abstraccion de
datos */
public interface IteratorIF<T>{
    /* Devuelve: el siguiente elemento de la
    iteracion */
    public T getNext ();
    /* Devuelve: cierto si existen mas elementos en
    el iterador */
    public boolean hasNext ();
    /* Restablece el iterador para volver a recorrer
    la estructura */
    public void reset ();
}

```