Departamento de Lenguajes y Sistemas Informáticos

Material permitido: NINGUNO. **Duración:** 2 horas

Entregue este folio con sus datos consignados

Alumno:	**Identificación:**
C. Asociado en que realizó la Práctica Obligatoria:	

P1 (1'5 puntos) **Práctica**. Describa cómo modificaría el algoritmo de búsqueda de la mejor plaza para un vehículo si se permitiese a) ocupar plaza de familiar a uno normal pero no a la inversa y b) se prefiriese que un vehículo normal ocupase plaza de normal. Explique con precisión qué consideraciones adicionales debería realizar para añadir esta variación.

1. (2 puntos) Implementar en JAVA una función recursiva cuyo perfil sea el siguiente:

   ```
   ListIF<ListIF<Integer>> split (ListIF<Integer> L, int x)
   ```

   que genere dos listas de manera que la primera contenga los elementos de la lista L menores o iguales que x y la segunda contenga los elementos de L mayores que x. Calcular el coste asintótico temporal en el caso peor de la función pedida.

2. (6'5 puntos) El bingo es un juego de azar en el que salen de un bombo una serie de bolas numeradas del 1 al 90 sin repeticiones. Los jugadores tienen cartones consistentes en tres *líneas* con cinco números:

   - En la primera línea hay números del 1 al 30 ordenados de menor a mayor sin repeticiones
   - En la segunda línea hay números del 31 al 60 ordenados de menor a mayor sin repeticiones
   - En la tercera línea hay números del 61 al 90 ordenados de menor a mayor sin repeticiones

   Durante el juego, van saliendo del bombo las diferentes bolas numeradas, de manera que los jugadores marcan en sus cartones aquellas casillas que contengan los números de las bolas que van saliendo. Un jugador canta línea cuando ha marcado todos los números contenidos en alguna línea de su cartón. Cuando un jugador ha marcado todos los números de su cartón, canta *bingo*, y el juego se termina. Consideraremos dos clases de interés para resolver el problema, que se llamarán respectivamente `BingoCard` y `BingoGame`. Las operaciones que pueden aplicarse sobre cartones de bingo y una partida de bingo aparecen definidas en las siguientes clases:

   BingoCard

```
//Representa un cartón de bingo en su versión de 90 bolas
public class BingoCard {

/* Actualiza el cartón de bingo adecuadamente cuando ha salido del */
/* bombo la bola dada como parámetro. [1 punto]                    */
/* @param el número que sale del bombo (bola)                      */
public void updateCard(int bola);

/* Indica si el cartón ha completado una línea                     */
/* Cada línea sólo se puede declarar una única vez                 */
/* @return true si y sólo si el cartón completa una línea          */
public boolean isLine();

/* Indica si al salir la bola dada como parámetro se canta bingo   */
/* en el cartón. [1 punto]                                         */
/* @return true si y sólo si el cartón completa el bingo           */
public boolean isBingo();
}
```

BingoGame

```
//Gestiona un juego de bingo en su versión de 90 bolas

public class BingoGame {

/* Saca una bola que no haya salido con anterioridad. [1 punto]     */
/* @return el número de la bola extraída del bombo                  */
public int drawNumber();

/* Indica si algún cartón de la partida ha conseguido línea         */
/* @return true si algún cartón ha completado una línea [0'5 puntos]*/
public boolean checkLine();

/* Indica si algún cartón de la partida ha conseguido bingo         */
/* @return true si algún cartón ha completado bingo [0'5 puntos]    */
public boolean checkBingo();

}
```

Se pide:

a) (0'5 puntos) Describa detalladamente cómo realizaría la representación interna de la clase **BingoCard**, que representa el estado de un cartón de bingo (usando los TAD estudiados en la asignatura). Justifique su elección y detalle el constructor de una clase que implemente esta interfaz.

b) (0'5 puntos) Describa detalladamente cómo realizaría la representación interna de la clase **BingoGame**, que representa el estado actual de un juego de bingo, esto es: el conjunto de cartones de la partida y la lógica de control de los números que van saliendo del bombo. Justifique su elección y detalle el constructor de una clase que implemente esta interfaz.

c) (5 puntos) Basándose en las respuestas anteriores, implemente todos los métodos de las clases **BingoCard** y **BingoGame**. Se valorará que detalle los contratos (pre y postcondiciones) de las operaciones o que comente, al menos, las restricciones que deben aplicarse a los parámetros de entrada. (**Nota:** las puntuaciones asignadas a cada método se indican en la especificación de dicho método en la interfaz del tipo).

d) (0'5 puntos) Calcule el coste asintótico temporal en el caso peor del método `checkBingo ()` 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 ();
}
```