Exámen 2012, preguntas y respuestas - Resolucion del Primer Parcial

Estructuras de datos y algoritmos (Universitat Politecnica de Valencia)

Resolución del Primer parcial de EDA (12/3/2012)

1.- La siguiente interfaz extiende la funcionalidad de una Lista con Punto de Interés para añadir un nuevo método, *borrarTodos*:

```
public interface ListaConPIPlus<E> extends ListaConPI<E> {
intborrarTodos(Ex);
}
```

Este método elimina todos los elementos de la lista que sean iguales a *x*, y devuelve el número de elementos que se han borrado.**Se pide** completar el método *borrarTodos* en la siguiente clase, haciendo uso únicamente de los métodos de la interfaz *ListaConPI*: **(2puntos)**

```
public classImplementacionListaConPIPlus<E>
extends ImplementacionListaConPI<E> implements ListaConPIPlus<E> {
   public intborrarTodos(Ex) {
      int n = 0;
      inicio();
      while ( !esFin() )
            if ( recuperar().equals(x) ){
                  eliminar();
                  n++;
            }
            else siguiente();
      return n;
   }
}
```

```
public interface
ListaConPI<E> {
voidinsertar(E e);
voideliminar();
 E recuperar();
voidinicio();
voidsiguiente();
booleanesFin();
booleanesVacia();
voidfin();
inttalla();
}
```

2.-La clase *LEGCola* es una implementación de la interfaz *Cola* mediante una lista enlazada. Se ha diseñado una nueva clase, *LEGColaDeEnteros*, a la que se quiere añadir un nuevo método, *borrarPrimero*, que elimine el primer elemento de la cola sí y sólo sí el valor del primer elemento es mayor que la suma de todos los demás. El método devuelve *true* si se ha realizado el borrado y *false* en caso contrario.

Se pide completar el método *borrarPrimero* asumiendo que hay, al menos, dos elementos en la cola: **(2 puntos)**

```
public class LEGColaDeEnteros extends LEGCola<Integer> {
  public booleanborrarPrimero(){
      int suma = 0;
      NodoLEG<Integer> aux = primero.siguiente;
      while ( aux!=null ){
            suma += aux.dato;
            aux = aux.siguiente;
      }
      if ( primero.dato.compareTo(suma)>0 ){
            primero = primero.siguiente;
            return true;
      }
      else return false;
  }
}
```

```
publicclassLEGCola<E>
implements Cola<E>{
protectedNodoLEG<E>
primero, fin;
 …
}
classNodoLEG<E>{
 E dato;
NodoLEG<E>siguiente;
 …
}
```

3.- Se desea analizar el coste temporal del siguiente método recursivo: **(3 puntos)**

```
public int metodo(int v[], int i, intj){
      if ( i>j )return 0;
      else{
            int medio = ( i + j )/2;
            int n1 = metodo(v, i, medio - 1);
            int n2 = metodo(v, medio + 1, j);
            return n1 + v[medio] + n2;
      }
}
```

Para ello **se pide**:

a) Expresar la talla del problema en función de los parámetros del método: **(0.5 puntos)**

x = j–i+1

b) Indica de forma justificada si existen instancias significativas para una talla dada**:** **(0.5 puntos)**

No hay instancias significativas pues, en cualquier caso, se debe recorrer el (sub) array v[i…j]

c) Escribir las Relaciones de Recurrencia que definan el coste del método: **(1punto)**

$$T_{metodo}(x) = \begin{cases} k' & si\ x = 0 \\ T_{metodo}\left(\frac{x}{2}\right) + k & si\ x > 0 \end{cases}$$

d) Resuelve las relaciones de recurrencia del apartado anterior y escribe el coste temporal asintótico del método: **(1 punto)**

Por Teorema 3 con a =c=2, $T_{metodo}(x) \in \Theta(x)$

4.- Dado un array *v* de componentes de tipo *int*, ordenado de forma creciente y sin elementos repetidos, se quiere determinar si existe alguna componente de *v* que represente el mismo valor que el de su posición en *v* (y obtener dicha posición). En el caso de que no haya ninguna, se devolverá -1. **(3 puntos)**

Ejemplo:

0	1	2	3	4	5	6
-5	-4	-2	1	4	7	8

Para ello **se pide**: completar el método recursivo *igualAPosicion* haciendo uso de la estrategia Divide y Vencerás para obtener una implementación lo más eficiente posible.

```
public static int igualAPosicion(int v[]){
      return igualAPosicion(v, 0, v.length-1);
}

private static int igualAPosicion(int v[], int izq, int der){
      if ( izq <= der){
          int mitad = (izq + der)/2;
          if ( v[mitad]==mitad ) return mitad;
          if ( v[mitad]< mitad ) return igualAPosicion(v, mitad+1, der);
          else return igualAPosicion(v, izq, mitad-1);
      }
      else return -1;
}
```