Primer Parcial Grupos 81-82
Estructura de Datos y Algoritmos, 1º, Curso 2018---2019
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

13 de Marzo de 2019

Nombre y Apellidos:

Problema 1

```
public class SNode {
	public Integer elem;
	public SNode next;

	public SNode(Character e) {
		elem = e;
	}
}
public class SList {
	public SNode first;
	public SNode last;
	int size;
}
```

A (0,40) SList es una clase que implementa una lista simplemente enlazada de enteros. Añadir un método (no estático) isOrdered () que devuelva un boolean que determine si los enteros (contenidos en el objeto lista) están ordenados en orden ascendente (entero anterior<= entero).

Ejemplo:

- si la lista es {2,4,1,5} devolvería false (porque 1 no es mayor que 4)
- si la lista es {2,3,7,9} devolvería true
- si la lista es {2,3,7,7,9} devolvería true

```
public boolean isOrdered() {
	if (isEmpty() || size==1) {
		System.out.println("la lista es vacía o tiene un elemento y está ordenada");
		return true;
	}
	boolean ordened = true;
	SNode previousNode= first;
	for (SNode nodeIt = first.next; nodeIt != null && ordened; nodeIt = nodeIt.next) {
		if (previousNode.elem.compareTo(nodeIt.elem)<=0)  previousNode = nodeIt;
		else  ordened=false;
	}
	return ordened;

	}
```

B (0,40) Añadir un método (no estático) removeNoOrder() que transforme el objeto lista en una lista ordenada (los enteros que contiene estén ordenados). El método eliminará los nodos que no sigan la pauta de estar ordenados.

Ejemplo:

- la lista {2,4,1,5} se transformará en {2,4,5} (el 1 ha sido eliminado) porque no estaba ordenado.
- la lista {2,3,8,10} se transformará en la misma lista ya que está ordenada.

No se pueden utilizar llamadas a métodos como removeAt (), getIndexOf(), …

```
public void removeNoOrder() {
        if (isOrdered()) return;
        SNode previousNode= first;
        for (SNode nodeIt = first.next; nodeIt != null; nodeIt = nodeIt.next) {
                if (previousNode.elem.compareTo(nodeIt.elem)<=0)  previousNode = nodeIt;
                else {
                        if (nodeIt == last) {
                                previousNode.next=null;
                                last=previousNode;
                                size--;
                        } else {
                        previousNode.next = nodeIt.next;
                        size--;
                        }
                        }
                }

        }
```

C. (0,50) Haz el método removeNoOrder() del apartado anterior pero en este caso utilizando listas doblemente enlazadas. Escribir la clase dNode, atributos y constructor de la clase dList y el método removeNoOrder().

No se pueden utilizar llamadas a métodos como removeAt (), getIndexOf(), …

```
public class DNode {

        public Integer elem;
        public DNode prev;
        public DNode next;

        public DNode(Integer elem) {
                this.elem = elem;
        }
}
public class DList  {

        DNode header;
        DNode trailer;
        int size=0;

        public DList() {
                header = new DNode(null);
                trailer = new DNode(null);
                header.next = trailer;
                trailer.prev= header;

        }
```

```
public void removeNoOrder() {
        if (isEmpty() || size==1) {
                System.out.println("la lista es vacía o tiene un elemento y está ordenada");
                return;
        }
        for (DNode nodeIt = header.next.next; nodeIt != trailer; nodeIt = nodeIt.next) {
                if (nodeIt.prev.elem.compareTo(nodeIt.elem)<=0);
                else {
                        nodeIt.prev.next = nodeIt.next;
                        nodeIt.next.prev= nodeIt.prev;
                        size--;
                        }
        }

}
```

D. (0,20 pto.) Explicar brevemente la complejidad (Big-O) de los métodos de los apartados A, B y C. No tienes que calcular T(n).

Solución:

- isOrdered (): En el peor de los casos, el método tendrá que comprobar que la lista está ordenada y visitar nodo por nodo para comparar con su siguiente (comparas los elementos), siendo la complejidad del bucle O(n). Por tanto, la complejidad final es O(n), lineal
- removeNoOrder() con lista simplemente enlazada: el método tendrá que visitar nodo por nodo para comparar con su siguiente (comparas los elementos), y eliminar nodo en el caso de no estar ordenado, siendo la complejidad del bucle O(n). Por tanto, la complejidad final es O(n), lineal
- removeNoOrder() con lista doblemente enlazada: el método tendrá que visitar nodo por nodo para comparar con su siguiente (comparas los elementos), y eliminar nodo en el caso de no estar ordenado, siendo la complejidad del bucle O(n). Por tanto, la complejidad final es O(n), lineal

Problema2

A (0,20 pto.) Escribir un método recursivo que dado un número n>0, devuelva el producto de los triples de los números desde 1 a n (ambos incluidos).

Ejemplo:

- productoTriple(2)=(1*3) * (2*3) = 3*6 =18
- productoTriple(3)= (1*3) * (2*3) * (3*3) = 3*6*9=162

Solución:

```
public static int productoTriple(int n) { //n>0
        if (n==1) return 3;
        else return 3*n * productoTriple(n-1);

        }
```

B (0,30 pto.) Escribir un método recursivo que tome un array de enteros y compruebe si el array está ordenado (orden ascendente).

Solución:

```
public static boolean checkSort(int a[]) {
            if (a==null || a.length<0) return true;
```

```
        return checkSort(a,0);
}

private static boolean checkSort(int a[], int pos) {
        if (pos==a.length-1) return true;
        if (a[pos+1]<a[pos]) return false
        else return checkSort(a,pos+1);
```