***1. Invertir lista (0,5 puntos)***

Tenemos los siguientes algoritmos para invertir los valores de una lista:

a)

```
public ArrayList<T> invertir(ArrayList<T> l) {
// post: el resultado contiene los elementos de la lista original en
//       orden inverso

       ArrayList<T> resultado = new ArrayList<T>();

       Iterator<T> it = l.iterator();

       while it.hasNext(){
          T s = it.next();
          resultado.addFirst(s);
       }

       return resultado;
 }
```

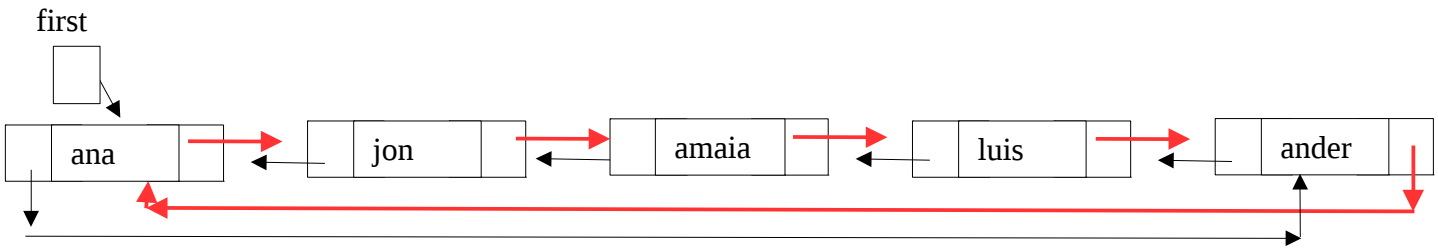
b)

```
public ArrayList<T> invertir(ArrayList<T> l) {
// post: el resultado contiene los elementos de la lista original en
//       orden inverso

       ArrayList<T> resultado = new ArrayList<T>();
       Stack<T> aux = new Stack<T>();

       Iterator<T> it = l.iterator();

       while it.hasNext(){
          T s = it.next();
          aux.push(s);
       }

       while (!aux.isEmpty()){
          T s = aux.pop();
          resultado.addLast(s);
       }

       return resultado;
 }
```

a) Calcular el coste de esos dos algoritmos, **de manera razonada**

b) Decir, cuál de las dos soluciones propuestas es más eficiente

***1. (1 punto) Tenemos una lista doblemente enlazada circular:***




Queremos implementar el siguiente método:

```
public class DoubleNode<T> {
   T data;
   DoubleNode<T> next;
   DoubleNode<T> prev;
}

public class DoubleLinkedList<T> {
   DoubleNode<T> first;

   public DoubleLinkedList<T> addAfter(T elem, T target)
   // Precondición:
   // Postcondición: se ha insertado el elemento “elem” detrás del elemento “target”,
   //                 En caso de que no existiera un elemento igual a “target”, se
   //                 insertará el nuevo elemento al principio de la lista

}
```

Por ejemplo, addAfter(“peio”, “amaia”) insertaría a “peio” entre “amaia” y “luis”.

Se pide:

- Implementar el algoritmo
- Calcular el coste de manera razonada