Pregunta 1. 2.5 puntos

Se tiene una ListaConPI genérica cuyos elementos están ordenados ascendentemente. En dicha lista, algunos elementos pueden aparecer repetidos dos veces. Se pide diseñar un método que elimine los elementos repetidos de la ListaConPI y los guarde en una Pila.

Por ejemplo, si el contenido de la ListaConPI es: 

Tras la llamada al método la ListaConPI quedará: 

y se devolverá la siguiente Pila: 

.

Completa los huecos del siguiente método con una de las opciones (indicando la **letra** de la opción) que figuran a la derecha. Una opción puede aparecer en más de un hueco.

```
public static <E> Pila<E> eliminarRepetidos(ListaConPI<E> l) {
    Pila<E> p = new [hueco 1]<E>();
    if ([hueco 2]()) return p;
    [hueco 3]();
    E e1 = [hueco 4]();
    do {
        [hueco 5]();
        if ([hueco 6]()) {
            E e2 = [hueco 7]();
            if ([hueco 8]) {
                [hueco 9](e2);
                [hueco 10]();
            }
            e1 = e2;
        }
    } while ([hueco 11]());
    return p;
}
```

OPCIONES	
A	Pila
B	ArrayPila
C	LEGPila
D	p.apilar
E	p.esVacia
F	l.inicio
G	l.siguiente
H	l.esFin
I	!l.esFin
J	l.esVacia
K	!l.esVacia
L	l.recuperar
M	l.eliminar
N	e1.equals(e2)
O	e1.compareTo(e2) == 0

hueco 1: **B** | **C**
hueco 2: **J**
hueco 3: **F**
hueco 4: **L**
hueco 5: **G**
hueco 6: **I**
hueco 7: **L**
hueco 8: **N**
hueco 9: **D**
hueco 10: **M**
hueco 11: **I**

Pregunta 2. 2.5 puntos

1.- Completar la implementación del siguiente método.

El método devuelve el número de Strings de una longitud dada, **x**, en un array de Strings, **v**, ordenado ascendentemente por longitud de String.

Se pide completar los huecos, de modo que la implementación sea la más eficiente posible.

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

public static int deLongitudX(String[] v, int x, int i, int j) {
    if (i > j) return 0;
    int m = /** hueco 1 **/;
    int l = v[m].length();
    if (l == x) {
        return 1 + /** hueco 2 **/;
    } else if (l < x) {
        return /** hueco 3 **/;
    } else {
        return /** hueco 4 **/;
    }
}
```

hueco 1: **(i + j) / 2**
hueco 2: **deLongitudX(v, x, i, m - 1) + deLongitudX(v, x, m + 1, j)**
hueco 3: **deLongitudX(v, x, m + 1, j)**
hueco 4: **deLongitudX(v, x, i, m - 1)**

2.- Obtener el coste Temporal Asintótico del método.

Sea **x** la talla del problema que indica el tamaño del subarray de cada llamada.

Completa los huecos con aquellas de las siguientes opciones que consideres adecuadas: **Theta, Omega, O, (1), (logx), (x), (x*x), (x*logx) o (2^x).**

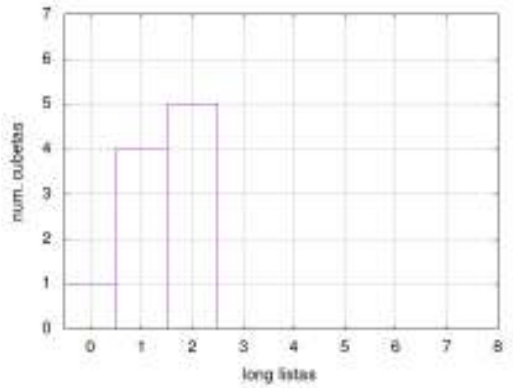
Se debe respetar la notación, incluidos los paréntesis que aparecen en algunas de las opciones.

****Como mínimo (cota inferior), T(x) ε Omega(log x)**

****Como máximo (cota superior), T(x) ε O(x)**

Pregunta 3. 2 puntos

Cuestión 1. Dado el histograma de ocupación de una tabla hash, contesta a las siguientes preguntas:



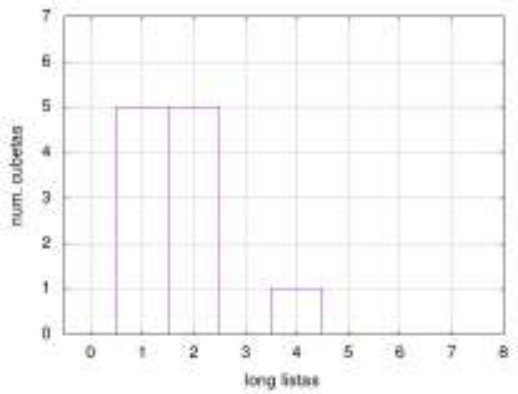
a) ¿Cuántas cubetas tiene la tabla? **10**
b) ¿Cuántos elementos tiene la tabla? **14**
c) ¿Cuál es su factor de carga? **1,4**

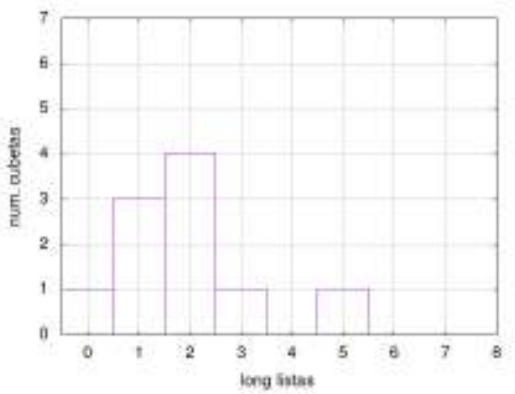
Cuestión 2. Dibuja una tabla hash que pueda haber producido el histograma anterior. Dicha tabla hash almacenará enteros en el rango [10..99] (elige los enteros que desees para crearla, pero que estén dentro de ese rango). La función de dispersión usada es H(x) = x % elArray.length. Para dibujar la tabla, usa el siguiente formato: cada una de las filas representa una cubeta de la tabla hash, la primera columna será el índice de cubeta, y, a continuación, los elementos almacenados en esa cubeta, separados por comas. Usa tantas filas como necesites.

Cubeta Elementos

0	
1	**11**
2	**12**
3	**13**
4	**14**
5	**15, 25**
6	**16, 26**
7	**17, 27**
8	**18, 28**
9	**19, 29**

Cuestión 3. Si se insertan 5 elementos distintos que no estaban en la tabla, ¿qué histograma(s) podría(n) corresponder al resultado? Marca con verdadero ("V") o falso ("F") las siguientes afirmaciones.





F El histograma de la izquierda.
V El histograma de la derecha.

Pregunta 4. 3 puntos

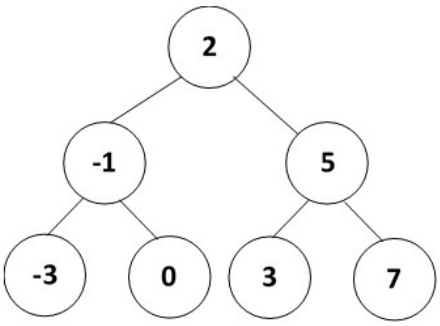
Sea **ABBDeInteger** una subclase de **ABB** que representa un **ABB** de **Integer.**

public class ABBDeInteger extends ABB<Integer> ...

Se pide completar la implementación de los siguientes métodos de instancia de **ABBDeInteger** para que proporcionen una **ListaConPI** de enteros, ordenada ascendentemente, con los valores del ABB mayores que el valor recibido como argumento.

La implementación debe tener el menor coste posible (ser la más eficiente). En el código a añadir, no se permite declarar variables locales.

Ejemplo. Sea **ab**, el **ABBDeInteger** de la siguiente figura, la llamada **ab.numerosMayores(-1)** devuelve la lista: 0, 2, 3, 5, 7.




```
public ListaConPI<Integer> numerosMayores(Integer n) {
    ListaConPI<Integer> ls = new LEGListaConPI<Integer>();
    /** hueco 1 **/;
    return ls;
}
protected void numerosMayores(Integer n, ListaConPI<Integer> ls, /** hueco 2 **/ nodo) {
    if (/** hueco 3 **/) return; // caso base
    // caso general recursión:
    /** hueco 4 **/
    if (/** hueco 5 **/) {
        ls.inicio();
        /** hueco 6 **/
        /** hueco 7 **/
    }
}
```

Indica el código a completar en cada hueco a continuación:

hueco 1: **numerosMayores(n, ls, this.raiz)**

hueco 2: **NodoABB**

hueco 3: **nodo == null**

hueco 4: **numerosMayores(n, ls, nodo.der);**

hueco 5: **nodo.dato.compareTo(n) > 0**

hueco 6: **ls.insertar(nodo.dato);**

hueco 7: **numerosMayores(n, ls, nodo.izq);**