Resolución del examen de Recuperación del Primer Parcial de EDA (12 de Junio de 2019)

1.- ***(3 puntos)*** En la clase **ABB**, implementa un método público que, con el menor coste temporal posible, elimine el nodo que contiene el máximo de un ABB.

```
public void eliminarMax() {
    if (this.raiz != null) { this.raiz = eliminarMax(this.raiz); }
}
//SII actual != null: devuelve el nodo actual tras eliminar su máximo
protected NodoABB<E> eliminarMax(NodoABB<E> actual) {
    NodoABB<E> res = actual;
    if (actual.der != null) {
        res.der =  eliminarMax(actual.der);
        res.talla--; // o res.talla = 1 + talla(res.izq) + talla(res.der);
    }
    else { res = actual.izq; }
    return res;
}
```

Una vez diseñado el método, y suponiendo que se aplica sobre un ABB Equilibrado de N nodos y altura H, ...

a) Indica la talla del problema que resuelve **x =** N, o bien H = $\log_2$ N ***(0.1 puntos)***

b) Indica si hay instancias significativas para una talla dada y por qué. En caso afirmativo, descríbelas. ***(0.25 puntos)***

No hay instancias significativas, pues el método realiza siempre un **Recorrido** en Pre-Orden del Camino más a la derecha del ABB hasta alcanzar su último nodo, lo que siempre tiene el mismo coste en un ABB equilibrado: del orden de la altura del ABB, o del logaritmo de su talla. Además, el borrado del nodo más a la derecha del ABB no añade coste ni casos, pues como solo puede ser una Hoja o un nodo con hijo Izquierdo, se realizará en tiempo constante.

c) Indica su coste Temporal utilizando la notación asintótica (O y Ω o bien Θ). ***(0.4 puntos)***

$T_{eliminarMax}(N) \in \Theta(\log_2 N)$, o bien $T_{eliminarMax}(H) \in \Theta(H = \log_2 N)$

2.- ***(3 puntos)*** Escribe un método estático Divide y Vencerás que, dados un array **v** de **`int`** ordenado ascendentemente y sin elementos repetidos y un **`int x`**, devuelva el elemento de **v** con valor más cercano al de **x**. Para ello...

- Puedes suponer que **v** tiene como mínimo tres elementos y que, de ellos, el elemento de valor más cercano al de **x** no está NI en la primera posición de **v** NI en la última.
- Puedes usar en tu código un método **`comparar`** que, como su nombre indica y en el orden de una constante, compara con **x** dos elementos **eV1** y **eV2** de **v** y devuelve aquel de ellos con valor más cercano al de **x**. Su perfil es: **`int comparar(int[] v, int eV1, int eV2, int x)`**.

```
public static int buscarMasCercano(int[] v, int x) {
    return buscarMasCercano(v, x, 0, v.length - 1);
}
protected static int buscarMasCercano(int[] v, int x, int ini, int fin) {
    int m = (ini + fin) / 2;
    if (v[m] == x) { return v[m]; }
    if (v[m] < x) {
        if (v[m + 1] > x) { return comparar(v, v[mitad], v[mitad + 1], x); }
        else { return buscarMasCercano(v, x, m + 1, fin); }
    }
    else if (v[m - 1] < x) { return comparar(v, v[m - 1], v[m], x); }
    else { return buscarMasCercano(v, x, ini, m - 1); }
}
```

Una vez diseñado el método, estudia su coste temporal del método recursivo que lanza. En concreto:

a) Expresa la talla del problema **x** en función de sus parámetros **x =** `fin - ini + 1` ***(0.2 puntos)***

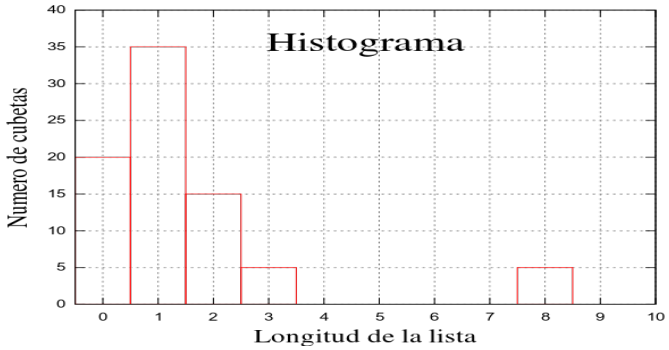
b) Escribe la(s) Relación(es) de Recurrencia que expresa(n) su coste. ***(0.4 puntos)***

En el caso general, **cuando x >** **2** , $T_{buscarMasCercano}^{M}(x) = k$ y $T_{buscarMasCercano}^{P}(x) = 1 * T_{buscarMasCercano}^{P}(x / 2) + k'$.

c) Resuelve la(s) Relación(es) de Recurrencia del apartado **b)**, indicando el(los) Teoremas de Coste que usas y escribiendo el coste Temporal del método en notación asintótica (O y Ω o bien Θ). ***(0.4 puntos)***

$T_{buscarMasCercano}(x) \in \Omega(1)$ y, por Teorema 3 con a = 1 y c = 2, $T_{buscarMasCercano}(x) \in O(\log_2 x)$.

3.- ***(2 puntos)*** Dado el siguiente histograma de ocupación de una Tabla Hash (con Hashing Enlazado):



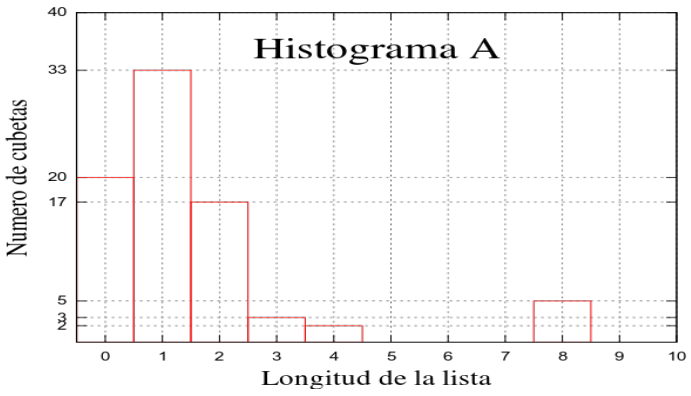
a) Indica el número de cubetas y el de elementos que tiene la Tabla, dejando indicadas las operaciones que has realizado para calcular dichos valores. ***(0.5 puntos)***

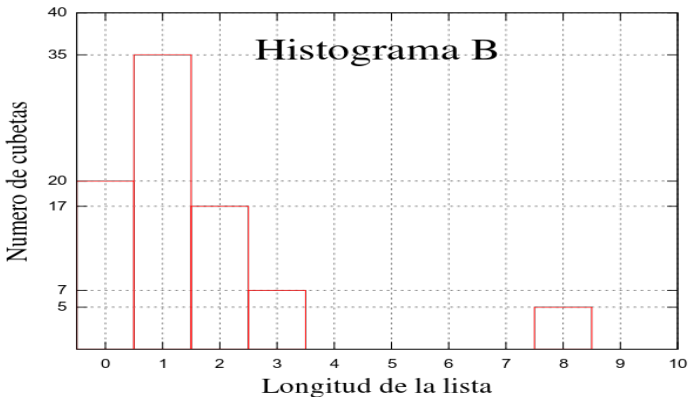
Número de cubetas, o `elArray.length` = 20 + 35 + 15 + 5 + 5 = 80.
Número de elementos de la Tabla, o `talla` = 20 * 0 + 35 * 1 + 15 * 2 + 5 * 3 + 5 * 8 = 120.

b) Calcula el Factor de Carga de la Tabla y la desviación típica de las longitudes de sus cubetas, dejando indicadas las operaciones que has realizado para calcular dichos valores. ***(0.5 puntos)***

FC = `talla / elArray.length` = 120 / 80 = 1.5.
σ = sqrt((20 * (0-1.5)² + 35 * (1-1.5)² + 15 * (2-1.5)² + 5 * (3-1.5)² + 5 * (8-1.5)²) / 80).

c) Supón que se insertan cuatro elementos distintos y que no estaban en la Tabla. Indica cuál de los siguientes es el histograma de ocupación de la Tabla tras estas inserciones. Es imprescindible que razones tu respuesta. ***(1 punto)***





El histograma de ocupación resultante es el A porque en él hay dos cubetas más de 2 elementos que en el anterior (tras insertar 2 nuevos elementos en 2 de sus cubetas de longitud uno) y, consecuentemente, dos cubetas menos de 1 elemento que en la anterior. Además, también hay en él dos cubetas más de 4 elementos que en el anterior (tras insertar 2 nuevos elementos en 2 de sus cubetas de longitud tres).
Por otra parte, el histograma resultante no puede ser el B porque tras insertar nuevos elementos en la Tabla no puede aumentar el número de cubetas de 1 elemento SIN disminuir el número de las que tienen 0 o 1.

4.- ***(2 puntos)*** El siguiente método de la clase **`ArrayColaExt<E extends Comparable<E>>`**, que extiende de **`ArrayCola<E>`**, ordena ascendentemente los elementos de una Cola usando una **`ListaConPI`** como estructura auxiliar. Escribe en cada recuadro el número de la opción (ver listado a la derecha) que le corresponde.

```
public void ordenar() {
  [6] lpi = new [5] ();
  while (!this.esVacia()) {
    E e = [2] ();
    for (lpi.inicio(); [3] && [1] ; [8] );
      [7] (e);
  }
  lpi.inicio();
  while (!lpi.esVacia()) {
    [10] ( [9] ());
    [4] ();
  }
}
```

① `lpi.recuperar().compareTo(e) <= 0`
② `this.desencolar`
③ `!lpi.esFin()`
④ `lpi.eliminar`
⑤ `LEGListaConPI<E>`
⑥ `ListaConPI<E>`
⑦ `lpi.insertar`
⑧ `lpi.siguiente()`
⑨ `lpi.recuperar`
⑩ `this.encolar`

ANEXO

Las clases `NodoABB` y `ABB` del paquete `jerarquicos`.

```
class NodoABB<E> {
    E dato;
    NodoABB<E> izq, der;
    int talla;
    NodoABB(E dato) {…}
}
```

```
public class ABB<E extends Comparable<E>> {
    protected NodoABB<E> raiz;
    protected int talla;
    public ABB() {…}
    …
}
```

Teoremas de coste:

Teorema 1: $f(x) = a \cdot f(x - c) + b$, con $b \geq 1$

- si $a=1$, $f(x) \in \Theta(x)$;
- si $a>1$, $f(x) \in \Theta(a^{x/c})$;

Teorema 2: $f(x) = a \cdot f(x - c) + b \cdot x + d$, con b y $d \geq 1$

- si $a=1$, $f(x) \in \Theta(x^2)$;
- si $a>1$, $f(x) \in \Theta(a^{x/c})$;

Teorema 3: $f(x) = a \cdot f(x/c) + b$, con $b \geq 1$

- si $a=1$, $f(x) \in \Theta(\log_c x)$;
- si $a>1$, $f(x) \in \Theta(x^{\log_c a})$;

Teorema 4: $f(x) = a \cdot f(x/c) + b \cdot x + d$, con b y $d \geq 1$

- si $a<c$, $f(x) \in \Theta(x)$;
- si $a=c$, $f(x) \in \Theta(x \cdot \log_c x)$;
- si $a>c$, $f(x) \in \Theta(x^{\log_c a})$;

Teoremas maestros:

Teorema para recurrencia divisora: la solución a la ecuación $T(x) = a \cdot T(x/b) + \Theta(x^k)$, con $a \geq 1$ y $b>1$ es:

- $T(x) \in O(x^{\log_b a})$ si $a>b^k$;
- $T(x) \in O(x^k \cdot \log x)$ si $a=b^k$;
- $T(x) \in O(x^k)$ si $a<b^k$;

Teorema para recurrencia sustractora: la solución a la ecuación $T(x) = a \cdot T(x-c) + \Theta(x^k)$ es:

- $T(x) \in \Theta(x^k)$ si $a<1$;
- $T(x) \in \Theta(x^{k+1})$ si $a=1$;
- $T(x) \in \Theta(a^{x/c})$ si $a>1$;