# Resolución del Primer Parcial de EDA (26 de Marzo de 2019)

**1.-** ***(2 puntos)*** El siguiente método de la clase **`ArrayCola`** invierte el orden de los elementos de una Cola, usando una **`ListaConPI`** auxiliar. Escribe en cada recuadro el número de la opción (ver listado a la derecha) que le corresponde.

```
public void invertir() {
    ListaConPI<E> lpi = [6] ();
    while ( !this.esVacia() ) {
        [2] ( [7] () );
        [4] ();
    }
    while ( !lpi.esVacia() ) {
        [1] ( [8] () );
        [3] ();
    }
}
```

① `this.encolar`

② `lpi.insertar`

③ `lpi.eliminar`

④ `lpi.inicio`

⑤ `new ListaConPI<E>`

⑥ `new LEGListaConPI<E>`

⑦ `this.desencolar`

⑧ `lpi.recuperar`

**2.-** ***(2 puntos)*** Estudia el coste Temporal del siguiente método:

```
/* Precondición: v ordenado ascendentemente AND a < b */
private static int metodo(int[] v, int ini, int fin, int a, int b) {
    if (ini > fin) { return 0; }
    int mitad = (ini + fin) / 2, res = 0;
    if (v[mitad] >= a) { res += metodo(v, mitad + 1, fin, a, b); }
    if (v[mitad] <= b) { res += metodo(v, ini, mitad - 1, a, b); }
    if (v[mitad] >= a && v[mitad] <= b) { res++; }
    return res;
}
```

**a)** Expresa la talla del problema **x** en función de los parámetros del método. **x =** fin – ini + 1 ***(0.2 puntos)***

**b)** Indica si hay instancias significativas para una talla dada y por qué. En caso afirmativo, descríbelas. ***(0.6 puntos)***

Sí hay instancias significativas:

- **Mejor caso:** todos los elementos de *v* son menores que *a* (o mayores que *b*)
- **Peor caso:** todos los elementos de *v* están en el intervalo [a, b]

**c)** En base a tu apartado **b)**, escribe la(s) Relación(es) de Recurrencia que expresa(n) el coste del método. ***(0.6 puntos)***

En el caso general, **cuando x >** **0** **,**

$T_{metodo}^{M}(x) = 1 * T_{metodo}^{M}(x / 2) + k$

$T_{metodo}^{P}(x) = 2 * T_{metodo}^{P}(x / 2) + k$

**d)** Resuelve la(s) Relación(es) de Recurrencia del apartado **c)**, indicando el(los) Teoremas de Coste que usas y escribiendo el coste Temporal del método en notación asintótica (O y Ω o bien Θ). ***(0.6 puntos)***

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

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

**3.-** ***(3 puntos)*** Sea una aplicación de gestión de notas en la que se usa un **`Map<Alumno, Double>`**, cada una de cuyas Entradas representa a un alumno y la nota que este ha obtenido en una asignatura.

**a)** Implementa un método cuyo perfil sea el mostrado en el siguiente recuadro, donde el parámetro **`m`** es el Map de las notas de todos los alumnos de una asignatura. El método debe realizar las siguientes acciones: ***(2.25 puntos)***

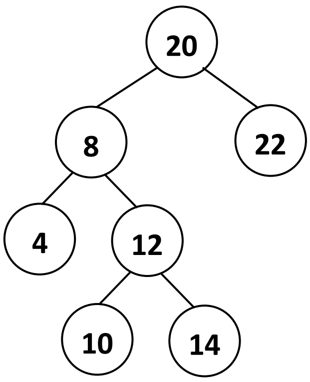
- Devolver un (nuevo) Map que contenga únicamente las Entradas de **`m`** que corresponden a alumnos aprobados (con nota mayor o igual que 5.0).
- Eliminar del Map **`m`** todas las Entradas que corresponden a alumnos aprobados. Es decir, al terminar la ejecución del método, **`m`** debe contener únicamente las Entradas correspondientes a alumnos suspendidos.

```
public static Map<Alumno, Double> obtenerAprobados(Map<Alumno, Double> m) {
    Map<Alumno, Double> aprobados = new TablaHash<Alumno, Double>(m.talla());
    ListaConPI<Alumno> l = m.claves();
    for (l.inicio(); !l.esFin(); l.siguiente()) {
        Alumno alumno = l.recuperar();
        Double nota = m.recuperar(alumno);
        if (nota >= 5.0) {
            aprobados.insertar(alumno, nota);
            m.eliminar(alumno);
        }
    }
    return aprobados;
}
```

**b)** Suponiendo que el Map **`m`** se ha implementado eficientemente mediante una **`TablaHash`**, indica para el método diseñado: la talla del problema **x** que resuelve, en función de sus parámetros; las instancias significativas que presenta, si las hubiera; su coste Temporal en notación asintótica (O y Ω o bien Θ). ***(0.75 puntos)***

- Talla del problema, en función de los parámetros del método: **x** = `m.talla()`.
- Instancias significativas: no hay, pues se trata de un método de Recorrido.
- Utilizando la notación asintótica, $T_{obtenerAprobados}(x) \in \Theta(x)$.

**4.-** ***(3 puntos)*** El Ascendiente Común "Más Bajo", o *Lowest Common Ancestor* (*LCA*), de dos elementos **e1** y **e2** de un Árbol se define como el elemento situado en el nodo "más bajo" (a mayor profundidad o distancia de la raíz) del que los nodos que contienen a **e1** y **e2** son descendientes, pudiendo ser un nodo descendiente de él mismo. Así, por ejemplo, en el ABB de la figura a tu derecha: el *LCA* de 10 y 14 es 12, el de 8 y 14 es 8, el de 10 y 22 es 20 y el de 8 y 22 es 20.




En la clase **ABB**, implementa un método público **`lCA`** que, en tiempo lineal con su altura, devuelva el *LCA* de **e1** y **e2** en un ABB Equilibrado, no vacío y sin elementos repetidos. Asume además que **e1** y **e2** son dos elementos del ABB, por lo que su *LCA* siempre existe, y que **e1** es menor que **e2**.

```
/ ** SII ABB no es vacío, e1 y e2 están en el ABB y e1 < e2 */
public E lCA(E e1, E e2) { return lCA(this.raiz, e1, e2).dato; }

// Búsqueda con garantía de éxito del Nodo del LCA de e1 y e2
protected NodoABB<E> lCA(NodoABB<E> n, E e1, E e2) {

    // Como e1 < e2, si n.dato > e2, también n.dato > e1
    if (n.dato.compareTo(e2) > 0) { return lCA(n.izq, e1, e2); }

    // Sino, si n.dato < e1, también n.dato < e2
   if (n.dato.compareTo(e1) < 0) { return lCA(n.der, e1, e2); }

   // Sino, o n.dato = e1, o n.dato = e2, o n.dato > e1 y n.dato < e2
   return n;

}
```

# ANEXO

## Las interfaces `Map` y `ListaConPI` del paquete `modelos`.

```
public interface Map<C, V> {
    V insertar(C c, V v);
    V eliminar(C c);
    V recuperar(C c);
    boolean esVacio();
    int talla();
    ListaConPI<C> claves();
}
```

```
public interface ListaConPI<E> {
   void insertar(E e);
   void eliminar();
   void inicio();
   void siguiente();
   void fin();
   E recuperar();
   boolean esFin();
   boolean esVacia();
   int talla();
}
```

## 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;