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.

<pre>public void invertir() {</pre>		
ListaConPI <e> lpi = ();</e>	① this.encolar	
while (!this.esVacia()) {	② lpi.insertar	
(());	③ lpi.eliminar	
}	<pre>④ lpi.inicio</pre>	
<pre>while (!lpi.esVacia()) {</pre>	<pre>⑤ new ListaConPI<e></e></pre>	
(());	<pre>⑥ new LEGListaConPI<e></e></pre>	
(); }	⑦ this .desencolar	
}	<pre>8 lpi.recuperar</pre>	
<pre>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 = (0.2 puntos) b) Indica si hay instancias significativas para una talla dada y por qué. En caso afirmativo, descríbelas. (0.6 puntos)</pre>		
c) En base a tu apartado b), escribe la(s) Relación(es) de Rec	urrencia que expresa(n) el coste del método.	(0.6 puntos)
En el caso general, cuando x >,		
d) Resuelve la(s) Relación(es) de Recurrencia del aparta escribiendo el coste Temporal del método en notación asint		que usas y (0.6 puntos)

- 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:
 - Devolver un (nuevo) Map que contenga únicamente las Entradas de **m** que corresponden a alumnos aprobados (con nota mayor o igual que 5.0).
 - Fliminar del Man **m** todas las Entradas que corresponden a alumnos aprobados. Es decir, al terminar la ejecución

<pre>public static Map<alumno, double=""> obtenerAprobados(Map<alumno, double)<="" pre=""></alumno,></alumno,></pre>	e> m) {
}	
b) Suponiendo que el Map m se ha implementado eficientemente mediante una TablaHa diseñado: la talla del problema x que resuelve, en función de sus parámetros; las instancias si las hubiera; su coste Temporal en notación asintótica (O y Ω o bien Θ).	
4 (3 puntos) El Ascendiente Común "Más Bajo", o <i>Lowest Common Ancestor</i> (<i>LCA</i>), 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 <i>LCA</i> 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.	8 22
En la clase ABB , implementa un método público 1CA que, en tiempo lineal con su altura, devuelva el <i>LCA</i> 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 <i>LCA</i> siempre existe, y que e1 es menor que e2 .	10 14

ANEXO

Las interfaces Map y ListaConPI del paquete modelos.

```
public interface ListaConPI<E> {
public interface Map<C, V> {
    V insertar(C c, V v);
                                                       void insertar(E e);
    V eliminar(C c);
                                                       void eliminar();
    V recuperar(C c);
                                                       void inicio();
    boolean esvacio();
                                                       void siguiente();
    int talla():
                                                       void fin();
    ListaConPI<C> claves();
                                                       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 \ge 1

• si a = 1, f(x) \in \Theta(x);

• si a > 1, f(x) \in \Theta(a^{x/c});

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

• si a > 1, f(x) \in \Theta(a^{x/c});

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

• si a > 1, f(x) \in \Theta(\log_c x);

• si a < c, f(x) \in \Theta(x);

• si a < c, f(x) \in \Theta(x);

• si a < c, f(x) \in \Theta(x);

• si a > c, f(x) \in \Theta(x);

• si a > c, f(x) \in \Theta(x)
```

Teoremas maestros:

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

```
    T(x) ∈ O(x<sup>log</sup><sub>b</sub><sup>a</sup>) si a>b<sup>k</sup>;
    T(x) ∈ O(x<sup>k</sup>·log x) si a=b<sup>k</sup>;
    T(x) ∈ O(x<sup>k</sup>) si a<b/>b<sup>k</sup>;
```

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

```
    T(x) ∈ Θ(x<sup>k</sup>) si a<1;</li>
    T(x) ∈ Θ (x<sup>k+1</sup>) si a=1;
```

• $T(x) \in \Theta$ ($a^{x/c}$) si a>1;