**Primer Parcial de Estructuras de Datos y Algoritmos (EDA) – 31 de Marzo de 2021 - Duración: 1h. 30m.**

| APELLIDOS, NOMBRE | GRUPO |
|---|---|
| | |

**1.-** ***(3,00 puntos)*** El siguiente método recibe dos **`ListaConPI`** genéricas, ambas sin elementos repetidos y ordenados ascendentemente. El método debe eliminar de dichas listas todos los elementos que tengan en común y devolverlos almacenados en una **`Pila`**.

Por ejemplo, si la primera lista es [1, 3, 5, 7, 9, 10] y la segunda es [1, 2, 4, 9, 11], tras la llamada al método quedarán como [3, 5, 7, 10] y [2, 4, 11], respectivamente, y la pila contendrá los dos elementos eliminados: el 9 en su tope y el 1 en su base.

Escribe en cada recuadro el número de la opción (ver listado a la derecha) que le corresponde. Una opción puede no aparecer en ningún recuadro, o puede usarse para rellenar uno o varios recuadros.

```
public static <[2]> Pila<[1]> metodo1(
  ListaConPI<[1]> l1, ListaConPI<[1]> l2) {
      Pila<E> p = [9]();
      l1.[6]();
      l2.[6]();
      while (!l1.[4]() && !l2.[4]()) {
          E e1 = l1.[7]();
          E e2 = l2.[7]();
          int cmp = e1.[14](e2);
          if (cmp [13] 0) {
              p.apilar(e1);
              l1.[8]();
              l2.[8]();
          } else if (cmp [12] 0) {
              l1.[5]();
          } else {
              l2.[5]();
          }
      }
      return p;
  }
```

① `E`

② `E extends Comparable<E>`

③ `esVacia`

④ `esFin`

⑤ `siguiente`

⑥ `inicio`

⑦ `recuperar`

⑧ `eliminar`

⑨ **`new`** `ArrayPila<E>`

⑩ **`new`** `Pila<E>`

⑪ `>`

⑫ `<`

⑬ `==`

⑭ `compareTo`

**2.-** Aplicación de la estrategia **Divide y Vencerás**.

**2.1.-** ***(2,70 puntos).*** Implementa un método recursivo eficiente que, dado un array de enteros, no vacío, ordenado ascendentemente y sin elementos repetidos, devuelva el primer valor omitido que haría que sus elementos fueran consecutivos. En el caso de que no hubiera valor omitido, se devolverá el siguiente valor en secuencia. Implementa, además, el método lanzadera que invoque el método recursivo.

La siguiente tabla muestra el resultado correcto para varias entradas, con un comentario que explica cada resultado. Por ejemplo, para el array [0, 1, 2, 4, 5, 7] el método devuelve 3, el 1º valor omitido. Observa que podrían haber otros valores omitidos (en ese ejemplo, también falta el 6). Se pide devolver el 1º omitido, excepto si no hay ninguno, como en el array [0, 1, 2, 3, 4], y entonces se devuelve el que sería siguiente, 5 en este caso.

| *Array de enteros* | *Resultado* | *Comentario* |
|---|---|---|
| [0, 1, 2, 4, 5, 7] | 3 | v[3] = 4 ≠ v[0] + 3 = 0 + 3 = 3 |
| [3, 4, 5, 7, 8, 9, 10, 12] | 6 | v[3] = 7 ≠ v[0] + 3 = 3 + 3 = 6 |
| [-2, -1, 0, 1, 2, 3, 5, 6, 9] | 4 | v[6] = 5 ≠ v[0] + 6 = -2 + 6 = 4 |
| [0, 1, 2, 3, 4] | 5 | ∀i v[i] = v[0] + i |
| [7] | 8 | |

```
public static int metodo2(int[] v) {
   int u = v.length - 1;
   if (v[u] == v[0] + u) return v[u] + 1;
   return metodo2(v, 0, u);
}

private static int metodo2(int[] v, int i, int f) {
   if (i > f || v[i] != v[0] + i) return v[0] + i;
   int m = (i + f) / 2;
   if (v[m] == v[0] + m) return metodo2(v, m + 1, f);
   else return metodo2(v, i, m - 1);
}
```

→ ***Nota. Existen otras implementaciones correctas y eficientes (es decir, con el mismo coste temporal).***

**2.2.-** Estudia el coste Temporal del método recursivo que has implementado:

**a)** ***(0,10 puntos).*** Expresa la talla del problema **x** en función de los parámetros del método. **x = f – i + 1** .

**b)** ***(0,40 puntos).*** Escribe la(s) Relación(es) de Recurrencia que expresa(n) el coste del método.

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

$T^m(x) = k$ → ***Nota. El caso mejor tiene coste constante, no hay relación de recurrencia en ese caso.***

$T^p(x) = T^p(x/2) + k$

**c)** ***(0,30 puntos).*** 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).

**Aplicando el teorema 3 con a=1 y c=2,** $T^p(x) \in \Theta(\log x)$. Por tanto, $T(x) \in \Omega(1)$ y $T(x) \in O(\log x)$.

**3.-** ***(3,50 puntos)*** Implementa un método eficiente que, dado un map **m1**, de tipo **Map<C,V>**, devuelva otro map **m2**, cuyo tipo sea **Map<V,ListaConPI<C>>**, y que almacene la misma información de **m1** pero invirtiendo claves y valores. En concreto:

- Las claves de **m2** serán los valores de **m1.**
- Los valores de **m2** serán listas con PI con todas las claves que, en **m1**, tenían ese mismo valor.

Ejemplo. Sea **m1** un **Map<String,Integer>**, donde las claves son nombres de jugadores de baloncesto y los valores son los puntos anotados (en una jornada, temporada, etc.). El resultado de invocar el método será un **Map<Integer,ListaConPI<String>>** donde para cada número de puntos se tendrá la lista de jugadores que han anotado dicha cantidad. Las siguientes tablas muestran unas instancias de este ejemplo:

**m1**

| | |
|---|---|
| "Larry Bird" | 258 |
| "Michael Jordan" | 362 |
| "Magic Johnson" | 264 |
| "Patrick Ewing" | 258 |
| "Shaquille O'Neal" | 264 |
| "David Robinson" | 243 |

**m2**

| | |
|---|---|
| 258 | "Larry Bird", "Patrick Ewing" |
| 362 | "Michael Jordan" |
| 264 | "Magic Johnson", "Shaquille O'Neal" |
| 243 | "David Robinson" |

```
public static <C,V> Map<V,ListaConPI<C>> metodo3(Map<C,V> m1) {
    Map<V,ListaConPI<C>> m2 = new TablaHash<>(m1.talla()); // m2, reverse map
    ListaConPI<C> lc = m1.claves(); // claves (directas) de m1
    for (lc.inicio(); !lc.esFin(); lc.siguiente()) {
        C c = lc.recuperar(); // clave en m1
        V v = m1.recuperar(c); // valor en m1, clave en m2
        ListaConPI<C> lv = m2.recuperar(v); // valor en m2
        if (lv == null) lv = new LEGListaConPI<C>();
        lv.insertar(c);
        m2.insertar(v, lv);
    }
    return m2;
}
```

# 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();
}
```

## 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:

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

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

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