Programación I

Parcial (turno mañana) – 13/11/2019

Ejercicio 1 *(20 pts. - 5 pts. cada item.)*

Discutir la veracidad de las siguientes afirmaciones, justificando su respuesta.

a) Si tenemos el siguiente código:
```
Point[] pts = new Point[3];
pts[0] = new Point(5,1);
pts[1] = pts[0];
pts[1] = null;
```
El Garbage Collector libera la memoria ocupada por el punto x=5 e y=1;

b) Para que no se produzca una excepción de StackOverflow en un algoritmo recursivo sobre enteros se debe hacer siempre la llamada recursiva con un número menor.

c) Los algoritmos de ordenamiento Quicksort y MergeSort tienen complejidad O(n*log(n)) en peor caso.

d) La única implementación posible del TAD Lista es con una lista enlazada.

Ejercicio 2 *(20 pts.)*

Escribir una función recursiva `String separarVocales(String s)` que toma un `String s` y retorna una nueva cadena en la cual las consonantes de `s` aparecen al principio y las vocales al final. El orden entre las vocales y entre las consonantes no importa, pero se debe cumplir que todas las consonantes estén al principio y todas las vocales al final.

Por ejemplo:

- `separarVocales("murcielago")` puede devolver `"mrclgoaeiu"`.
- `separarVocales("manzana")` puede devolver `"mnznaaa"`.
- `separarVocales("llamado")` puede devolver `"llmdoaa"`.
- `separarVocales("el")` debe devolver `"le"`.

Observación: puede haber varias formas de ordenar las letras de `s`, siempre que las consonantes estén al comienzo y las vocales al final. Por ejemplo, en el primero de los casos, otros posibles resultados podrían ser:

- "mrclgaeiou"
- "clgmraeiou"
- "mrclguieao"
- etc.

Se pide resolver **utilizando recursión**. Se pueden dar por hecha la función `String resto(String s)` que devuelve una cadena igual a `s` pero sin su primer caracter.

Ejercicio 3 *(35 pts. - item a. 15 pts. - item b. 20 pts.)*

Consideremos las siguientes clases:

```
public class Tienda {
        public Venta[] ventas;
        ...
}

public class Venta {
        public Ticket ticket;
        public Producto[] productos;
        public AsesorDeVenta[] asesores;
        ...
}

public class Producto {
        public String marca;
        public double precio;
        ...
}
```

```
public class Ticket {
        public String nombreCliente;
        public String apellidoCliente;
        public String direccionCliente;
        public double iva;
        ...
}

public class AsesorDeVenta {
        public int legajo;
        public String nombre;
        public String apellido;
        public double promedioDeVentasMensuales;
        ...
}
```

Estas clases modelan una Tienda que comercializa productos químicos. En la tienda tenemos un arreglo de ventas. Cada venta tiene un ticket con los datos del cliente, un arreglo con los productos vendidos en esa transacción, y dada la peligrosidad de los productos químicos, podemos tener uno o varios asesores de venta, almacenados en un arreglo, que brindaron asesoramiento en dicha venta. Para la clase Tienda, se pide:

a) Escribir el método **public int** ventasGrandes(**int** p, **int** a) que calcule la cantidad de ventas de la tienda en las que se vendieron más de p productos, y que participaron más de a asesores de ventas.
b) En las ventas que sólo involucran productos de una misma marca suelen tener promociones, por eso nos interesa conocer este tipo de ventas. Escribir el método **public** Set<Venta> ventasDeMarcaUnica() que devuelve el conjunto de ventas de la tienda que hayan involucrado solo a productos de la misma marca.
c) (*bonus track* 20 pts.) Escribir el método **public** AsesorDeVenta asesorMasVendedor() que devuelve el asesor de ventas que haya participado en la mayor cantidad de ventas de la tienda. En caso de existir más de un asesor con estas características se puede devolver cualquiera de ellos.

Ejercicio 4 *(25 pts.)*

Dadas las clases:

```
public class NodoInt {
    int elemento;
    NodoInt siguiente;
}
```

```
public class ListaInt {
    NodoInt primero;
    ...
}
```

Se pide para la clase ListaInt escribir el método de instancia **public void** todosMultiplosDe(**int** k) que modifica la lista **this** de modo que cada elemento que no es divisible por k quede multiplicado por k, cada vez que se realice una multiplicacion por k se agrega un nodo con 0 como elemento, y al final de la lista se agrega un nodo que tiene como elemento la cantidad de multiplicaciones que se realizaron.

Por ejemplo:

- Si la lista es [1,2,3,4,5,6] y k=2, la lista debe quedar [2,0,2,6,0,4,10,0,6,3].
- Si la lista es [1,9,9,9] y k=3, la lista debe quedar [3,0,9,9,9,1].
- Si la lista es [9,3,6,3] y k=3, la lista debe quedar [9,3,6,3,0].
- Si la lista es [] y k=5, la lista debe quedar [0].

Se pide además que el método implementado **sea de orden lineal**, es decir, $O(n)$ donde n es la cantidad de elementos de la lista. Justificar la complejidad del mismo.