Estructuras de Datos y Algoritmos **14-04-2025**

Mesa de Estudio 1

Autor: *Diego Banda* (diego.banda@mail.udp.cl)

1. Recursión

1.1. Cotas superiores

Para las siguientes funciones determine las cotas superiores utilizando notación Big O.

1. $5n + 3$
2. $4n^2 + 100n + 25$
3. $2^n + n^3$
4. $\log n + 100$
5. $3n^3 + 2n \log n + 7$
6. $n! + 2^n$
7. $\sqrt{n} + \log n$
8. $2^n + 3^n$
9. $n + n^2 + n^3 + ...n^k$
10. $\frac{n^3}{1000} + 999n$

1.2. Algoritmos

Para los siguientes algoritmos, analice el tiempo de ejecución en términos de Big O (**O(f(n))**).

1. $f1$

```java
public static void f1(int N){
    for(int i = 0 ; i < N ; i++){
        //  Body
    }
}
```

Listing 1: f1

2. $f2$

```java
public static void f2(){
    for(int i = 0 ; i < 1024 ; i++){
        //  Body
    }
}
```

Listing 2: f2

3. $f3$

```java
public static void f3(int N){
    for(int i = 0 ; i < N*N ; i++){
        //  Body
```

```
    }
}
```

Listing 3: f3

4. Tiempo de ejecución del método *f5*

```
public static void f4(int N){
    for(int i = 0 ; i < N ; i++){
        System.out.println(i);
    }
}

public static void f5(){
    for(int i = 0 ; i < 1000 ; i++){
        f4(i);
    }
}
```

Listing 4: f4 y f5

5. $f6$

```
public static int f6(int N){
    if(N == 0) return 0;
    return f6(N-1) + 1;
}
```

Listing 5: f6

6. $f7$

```
public static void f7(int N){
    if(N == 0) return 0;
    f7(N - 1);
    f7(N - 1);
}
```

Listing 6: f7

7. $f8$

```
public static int f6(int N){
    if(N == 0) return 0;
    return f6(N / 2) + 1;
}
```

Listing 7: f8

2. Compilador a papel

2.1. Algoritmo Misterioso: Postfix

Usted recibe una carta con un algoritmo misterioso en ella. Dado que usted es un experto en la materia, responderá las siguientes preguntas acerca de este misterioso algoritmo:

```
static int AlgoritmoMisterioso(String expr){
    Stack <Integer> stack = new Stack<>();

    for (int i = 0 ; i < expr.length() ; i++){
        char c = expr.charAt(i);

        if (c == ' '){
            continue;
        }

        if (Character.isDigit(c)){
            stack.push(c - '0');
        else {
            int v1 = stack.pop();
            int v2 = stack.pop();

            switch (c){
            case '+':
                stack.push(v2 + v1);
                break;
            case '-':
                stack.push(v2 - v1);
                break;
            case '/':
                stack.push(v2 / v1);
                break;
            case '*':
                stack.push(v2 * v1);
                break;
            }
        }
    }
    return stack.pop();
}
```

Listing 8: Algoritmo Misterioso

Pista: La expresión c - '0'tiene como resultado el valor del número c. Ejemplos: '0'- '0'= 0, '4'- '0'= 4.

1. Para cada uno de los siguientes casos de prueba, indique el valor retornado por el método *AlgoritmoMisterioso* al recibir el caso como argumento:
   - 5 5 +
   - 6 6 / 9 *
   - 2 4 * 2 2 * +
2. En base a sus observaciones del punto anterior, explique en no más de **3 lineas**, ¿De qué forma el algoritmo modifica los datos que recibe?
3. Analice el algoritmo en cuestión y describa su tiempo de ejecución en términos de **O(f(n))**, dónde **n** es el largo del arreglo recibido como argumento y **f** es una función matemática propuesta por usted. Fundamente su respuesta.

2.2. Algoritmo Misterioso 2

Se define una lista enlazada de la siguiente manera:

```
    class linkedList {
        class Node {
            int value;
            Node next;
        }
        Node head;
    }
```

Listing 9: Implementación LinkedList

Analice con detenimiento el siguiente algoritmo y responda las siguientes preguntas.

```
static Node algoritmoMisterioso(Node head){
    if(head == null || head.next == null)
        return head;

    Node temp = algoritmoMisterioso(head.next);

    head.next.next = head;

    head.next = null;

    return temp;
}
```

Listing 10: Algoritmo Misterioso 2

3. A programar

3.1. Suma de dos listas

Se define una lista enlazada de la siguiente manera:

```
    class linkedList {
        class Node {
            int value;
            Node next;
        }
        Node head;
    }
```

Listing 11: Implementación LinkedList

Considere el uso de esta lista enlazada para representar números enteros: con un dígito por nodo, en orden inverso de los dígitos (el último nodo representa el dígito más significativo). Por ejemplo, la lista 2 -> 9 -> null representa el número 92. La lista 1 -> 3 -> 9 -> null representa el número 931. Considerando lo anterior se le solicita que:

1. Defina un método *suma*, el cual recibe como argumentos los head de dos listas que representan números y retorna el head de una nueva lista, la cual representa la suma de los dos números recibidos. Como ejemplo:

   - Caso 1:

     Input:

     - L1: 8 -> 3 -> 2 -> null
     - L2: 3 -> 4 -> 3 -> null

     Output:

- suma: 1 -> 8 -> 5 -> null

- Caso 2:

  Input:

  - L1: 7 -> 1 -> 9 -> null
  - L2: 5 -> 9 -> null

  Output:

  - suma: 2 -> 1 -> 0 -> 1 -> null

2. Analice el algoritmo diseñado y describa su tiempo de ejecución en términos de **O(f(n, m))**, donde **n** y **m** son la cantidad de datos en la primera y segunda lista, respectivamente, y **f** es una función matemática propuesta por usted. Fundamente su respuesta.

3.2. Eliminar duplicados de una lista enlazada ordenada

Se define una lista enlazada de la siguiente manera:

```
class linkedList {
    class Node {
        int value;
        Node next;
    }
    Node head;
}
```

Listing 12: Implementación LinkedList

Usted es un apasionado coleccionista que le gusta tener todo ordenado y sin repetir nada, su colección es de una lista enlazada, la cual siempre tiene números ordenados pero a veces se tienen duplicados sin darse cuenta. Se le pide a usted todo un experto en listas enlazadas, que realice un algoritmo que recibe el head de una lista enlazada ordenada y elimina los duplicados.

1. Programe una función llamada *eliminarRepetidos* que recibe el head de una lista y retorna la lista sin duplicados. Como ejemplo:

   - Caso 1:

     Input:

     - Head: 1 -> 2 -> 2 -> 3 -> 4 -> null

     Output:

     - sinDuplicados: 1 -> 2 -> 3 -> 4 -> null

   - Caso 2:

     Input:

     - Head: 1 -> 1 -> 1 -> 1 -> null

     Output:

     - sinDuplicados: 1 -> null

2. Determine el tiempo de ejecución utilizando la notación **O(f(n))**.