# Estructura de datos y algoritmos

Rodrigo Alvarez

rodrigo.alvarez2@mail.udp.cl

## **Verdadero y Falso**

ullet \_\_\_\_ Mientras mejor CPU y memoria tenga un computador, mejor será la complejidad de tiempo  $\mathcal{O}(f(n))$  de un algoritmo.

• F Mientras mejor CPU y memoria tenga un computador, mejor será la complejidad de tiempo  $\mathcal{O}(f(n))$  de un algoritmo.

Mejorar el hardware, como la CPU y la memoria, puede acelerar la **ejecución real** del algoritmo, pero no cambia la complejidad asintótica  $\mathcal{O}(f(n))$ , que está relacionada con el crecimiento del número de operaciones a medida que aumenta n.

• \_\_\_\_ Un stack mantiene un orden LIFO (último que entra es el primero que sale)

ullet Un stack mantiene un orden LIFO (último que entra es el primero que sale)

• \_\_\_ Memoization es una técnica que almacena los resultados de ciertas operaciones de un algoritmo para evitar realizarlas de manera repetida.

 $\bullet$   $\underline{V}$  Memoization es una técnica que almacena los resultados de ciertas operaciones de un algoritmo para evitar realizarlas de manera repetida.

 $\bullet$  \_\_\_\_ Obtener el valor del elemento en la posición i de un arreglo tiene complejidad de tiempo  $\mathcal{O}(1).$ 

ullet Obtener el valor del elemento en la posición i de un arreglo tiene complejidad de tiempo  $\mathcal{O}(1)$ .

ullet \_\_\_\_ Agregar un elemento al final de una cola y obtener el primer elemento en la cola tiene complejidad  $\mathcal{O}(1)$ .

ullet Agregar un elemento al final de una cola y obtener el primer elemento en la cola tiene complejidad  $\mathcal{O}(1)$ .

• \_\_\_\_ Agregar un elemento a un stack y remover el elemento en la cima(top o peek) tiene complejidad  $\mathcal{O}(N)$ .

• F Agregar un elemento a un stack y remover el elemento en la cima (top o peek) tiene complejidad  $\mathcal{O}(N)$ .

Agregar un elemento a un stack y remover el elemento en la cima tienen complejidad  $\mathcal{O}(1)$ . Estas operaciones no dependen del tamaño del stack N, ya que solo se accede al último elemento añadido o se agrega uno al final, lo que ocurre en tiempo constante.

14

# Compilador de papel

#### Dado el siguiente algoritmo, responda las preguntas:

```
public static int binToInt(String bin) {
   int decimal = 0;
   int length = bin.length();
   for (int i = 0; i < length; i++) {
      char bit = bin.charAt(length - 1 - i);
      if (bit == '1') {
        decimal += Math.pow(2, i);
      }
   }
  return decimal;
}</pre>
```

Calcule el valor para las siguientes entradas:

- 11101
- 10001011
- 10101

16

#### Dado el siguiente algoritmo, responda las preguntas:

```
public static int binToInt(String bin) {
   int decimal = 0;
   int length = bin.length();
   for (int i = 0; i < length; i++) {
      char bit = bin.charAt(length - 1 - i);
      if (bit == '1') {
        decimal += Math.pow(2, i);
      }
   }
  return decimal;
}</pre>
```

Calcule el valor para las siguientes entradas:

• 11101: 29

10001011: 139

• 10101: 21

```
public static int binToInt(String bin) {
   int decimal = 0;
   int length = bin.length();
   for (int i = 0; i < length; i++) {
      char bit = bin.charAt(length - 1 - i);
      if (bit == '1') {
        decimal += Math.pow(2, i);
      }
   }
  return decimal;
}</pre>
```

Caracterice el tiempo de ejecución utilizando la notación  $\mathcal{O}(f(n))$ 

```
public static int binToInt(String bin) {
   int decimal = 0;
   int length = bin.length();
   for (int i = 0; i < length; i++) {
      char bit = bin.charAt(length - 1 - i);
      if (bit == '1') {
        decimal += Math.pow(2, i);
      }
   }
  return decimal;
}</pre>
```

Caracterice el tiempo de ejecución utilizando la notación  $\mathcal{O}(f(n))$ :  $\overline{\mathcal{O}(n)}$ 

## Análisis

```
public static int f1(int N) {
   if (N == 1 || N == 0) {
      return 1;
   }
   return f1(N - 1) + N;
}
```

Determine la complejidad temporal, describiendola con la notación Big O

```
public static int f1(int N) {
   if (N == 1 || N == 0) {
      return 1;
   }
   return f1(N - 1) + N;
}
```

Determine la complejidad temporal, describiendola con la notación Big O:  $\mathcal{O}(n)$ 

```
public static boolean f2(int N) {
    for(int i = 1; i*i < N; ++i) {
        if (N % i == 0) {
            return true;
        }
    }
    return false;
}</pre>
```

Determine la complejidad temporal, describiendola con la notación Big O

```
public static boolean f2(int N) {
    for(int i = 1; i*i < N; ++i) {
        if (N % i == 0) {
            return true;
        }
    }
    return false;
}</pre>
```

Determine la complejidad temporal, describiendola con la notación Big O:  $\mathcal{O}(\sqrt{n})$ 

```
public static int f3(int N) {
    int r = 0;
    while(N > 1) {
        ++r;
        N/=2;
    }
    return r;
}
```

Determine la complejidad temporal, describiendola con la notación Big O

```
public static int f3(int N) {
    int r = 0;
    while(N > 1) {
        ++r;
        N/=2;
    }
    return r;
}
```

Determine la complejidad temporal, describiendola con la notación Big O:  $\mathcal{O}(\log n)$ 

• Calculadora de listas enlazadas

27