



Instituto Politécnico Nacional
Escuela Superior de Cómputo



❖ UNIDAD DE APRENDIZAJE: ANÁLISIS DE ALGORITMOS

❖ PROFESOR: EDGARDO ADRIÁN FRANCO MARTÍNEZ

❖ ALUMNO:

BARRERA PÉREZ CARLOS TONATIHU



❖ Ejercicio 5: Análisis de algoritmos recursivos

❖ GRUPO: 3CM3

Calcular la complejidad para el algoritmo para el cálculo del factorial recursivo.

```

- int funcionRecursiva(int num) {
    if (num == 0)
        return 1;
    else if (num < 3) {
        resultado = 0;
        for (i = 0; i < num * num; i++)
            resultado += num;
        return resultado;
    } else
        return funcionRecursiva(num-1) * funcionRecursiva(num-2) / funcionRecursiva(num-3);
}

```

$$T(n) = \begin{cases} 0, & n=0 \\ n^2, & 0 < n < 3 \\ T(n-1) + T(n-2) + T(n-3) + 5 \end{cases}$$

Considerando las operaciones aritméticas

$$T(n) = T(n-1) + T(n-2) + T(n-3) + 5$$

$$T(n) - T(n-1) - T(n-2) - T(n-3) = 5$$

$$(x^3 - x^2 - x - 1)(x-1) = 0$$

$$r_1 = 1.8392$$

$$r_2 = -0.4196 + 0.6062i$$

$$r_3 = -0.4196 - 0.6062i$$

$$r_4 = 1$$

$$T(n) = c_1 r_1^n + c_2 r_2^n + c_3 r_3^n + c_4 r_4^n$$

$$O(c_1 r_1^n) \text{ mientras que } c_1 > 0$$

Calcular la complejidad de la implementación recursiva del termino n de la serie de Tribonacci (0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1705, ...).

```

int Tribonacci(int num) {
    if (num == 0)
        return 0;
    else if (num == 1 || num == 2)
        return 1;
    else
        return tribonacci(num-1) + tribonacci(num-2) + tribonacci(num-3);
}

```

$$T(n) = \begin{cases} 1, & n=0 \\ 2, & n=1, 2 \\ T(n-1) + T(n-2) + T(n-3) + 7, & n \geq 3 \end{cases}$$

Considerando las comparaciones y las operaciones aritméticas

$$T(n) = T(n-1) + T(n-2) + T(n-3) + 7$$

$$T(n) - T(n-1) - T(n-2) - T(n-3) = 7$$

$$(x^3 - x^2 - x - 1)(x-1) = 0$$

Haciendo un cambio de variable $x^3 = T(n)$, $a=0$
2 $b=1$

$$r_1 = 1.8392$$

$$r_2 = -0.4196 + 0.6062i$$

$$r_3 = -0.4196 - 0.6062i$$

$$r_4 = 1$$

$$\Rightarrow T(n) = C_1 r_1^n + C_2 r_2^n + C_3 r_3^n + C_4 r_4^n$$

$$O(C_1 r_1^n) \text{ mientras que } C_1 > 0$$

Resolver las siguientes ecuaciones y dar su orden de complejidad.

Problema 1

• $T(n) = 3T(n-1) + 4T(n-2)$ si $n > 1$; $T(0) = 0$; $T(1) = 1$

Resolviendo la recurrencia homogénea

$$T(n) - 3T(n-1) - 4T(n-2) = 0$$

Realizamos un cambio de variable $k=2$

$$x^2 - 3x - 4 = 0$$

$$(x-4)(x+1) = 0 \Rightarrow r_1 = 4 \text{ y } r_2 = -1$$

$$T(n) = C_1 4^n + C_2 (-1)^n \Rightarrow T(0) = 0 = C_1 + C_2$$

$$T(1) = 1 = C_1(4) - C_2$$

tenemos

$$\begin{aligned} C_1 + C_2 &= 0 \\ 4C_1 - C_2 &= 1 \end{aligned}$$

$$\frac{5C_1}{5} = 1 \Rightarrow C_1 = \frac{1}{5} \text{ y } C_2 = -\frac{1}{5}$$

$$T(n) = \frac{1}{5} 4^n - \frac{1}{5} (-1)^n \in O(4^n)$$

Problema 2

• $T(n) = 2T(n-1) - (n+5)3^n$ si $n > 0$; $T(0) = 0$

Resolviendo la recurrencia no homogénea

$$T(n) - 2T(n-1) = -(n+5)3^n$$

Realizamos un cambio de variable con $x' = T(n)$ $b=3$ y $d=1$

$$T(n) - 2T(n-1) = -n3^n - 5 \cdot 3^n$$

$$(x-2)(x-3)^2 = 0 \Rightarrow r_1 = 2, r_2 = 3 \text{ y } r_3 = 3$$

$$T(n) = C_1 2^n + C_2 3^n + C_3 n 3^n$$

$$T(0) = 0 = C_1 + C_2$$

$$T(1) = 2T(0) - (1+5)3 = 2(0) - (6)3 = -18$$

$$T(1) = -18 = C_1 2 + C_2 3 + C_3 3$$

$$T(2) = 2T(1) - (2+5)3^2 = 2(-18) - (7)3^2 = -57$$

$$T(2) = -57 = C_1 4 + C_2 9 + C_3 18$$

Obtenemos las siguientes ecuaciones

$$\begin{aligned} 4C_1 + 9C_2 + 18C_3 &= -57 \\ C_1 + C_2 &= 0 \\ 2C_1 + 3C_2 + 3C_3 &= -18 \end{aligned} \Rightarrow \begin{aligned} C_1 &= 51 \\ C_2 &= -51 \\ C_3 &= 11 \end{aligned}$$

$$T(n) = (51)2^n - (51)3^n + 11n3^n \in (51)2^n - (51)3^n < (11)n3^n$$

∴ $O(n3^n)$

Problema 3

• $T(n) = 3T(n-1) + 4T(n-2) + (n+5)2^n$ si $n > 1$ $T(0) = 0$, $T(1) = 100$

$T(n) - 3T(n-1) - 4T(n-2) = (n+5)2^n$ Resolviendo con un cambio de variable $x^2 = T(n)$, $b=2$ $d=1$

$$(x^2 - 3x - 4)(x-2)^2 = 0$$

$$(x-4)(x+1)(x-2)^2 = 0 \quad r_1 = 4, r_2 = -1, r_3 = 2 \quad \& \quad r_4 = 2$$

$$T(n) = C_1 4^n + C_2 (-1)^n + C_3 2^n + C_4 n 2^n$$

$T(0) = 0 = C_1 + C_2 + C_3$ $T(1) = 100 = 4C_1 - C_2 + 2C_3 + 2C_4$

$$T(2) = 3T(1) + 4T(0) + (2+5)2^2 = 7(4) + 300 = 328$$

$$T(2) = 328 = 16C_1 + C_2 + 4C_3 + 8C_4$$

$$T(3) = 3T(2) + 4T(1) + (3+5)2^3 = 3(328) + 4(100) + 64 = 1448$$

$$T(3) = 1448 = C_1 4^3 + C_2 + 8C_3 + 74C_4 = 64C_1 - C_2 + 8C_3 + 74C_4$$

$$C_1 = 23.2 \quad C_2 = -18.3 \quad C_3 = -4.8 \quad C_4 = -0.66$$

$$T(n) = 23.2(4^n) - 18.3(-1)^n - 4.8(2^n) - 0.66n2^n \in O(4^n)$$

Problema 4

• $T(n) - 2T(n-1) = 3^n$ si $n \geq 2$, $T(0) = 0$, $T(1) = 1$

Realizamos un cambio de variable $x = T(n)$ $a = 0$ $b = 3$

$(x - 2)(x - 3) = 0 \Rightarrow r_1 = 2$ & $r_2 = 3$

Sustituyendo estos valores tenemos

$T(n) = C_1 2^n + C_2 3^n$ | Resolviendo el sistema de ec.

$T(0) = 0 = C_1 + C_2$ | $C_2 = 1$ & $C_1 = -1$

$T(1) = 1 = 2C_1 + 3C_2$ | $\Rightarrow T(n) = 2^n + 3^n \in O(3^n)$

Calcular la cota de complejidad del algoritmo de búsqueda binaria recursiva (Ejemplo 04).

```
int BusquedaBinaria(int num_buscado, int numeros[], int inicio, int centro, int final) {  
    if (inicio > final)  
        return -1;  
    else if (num_buscado == numeros[centro])  
        return centro;  
    else if (num_buscado < numeros[centro])  
        return BusquedaBinaria(num_buscado, numeros, inicio, (int)((inicio+centro-1)/2), centro-1);  
    else  
        return BusquedaBinaria(num_buscado, numeros, centro+1, (int)((final+centro+1)/2), final);  
}
```

El análisis de este algoritmo es el siguiente.

• Para la búsqueda binaria tenemos lo siguiente

$T(n) = \begin{cases} 0, & n=0 \\ 3 + T(\frac{n}{2}), & n>0 \end{cases}$

Utilizamos el teorema maestro probando $f(n) = 3$, $a = 1$ & $b = 2$

Para el caso 2:
 $f(n) = 3 = \Theta(n^{\log_2 1}) = \Theta(n^0) = \Theta(1)$
 $\Theta(1)$ es constante así como $f(n)$, por lo tanto

$T(n) = \Theta(n^{\log_2 1} \log n) = \Theta(\log n)$

Calcular la cota de complejidad del algoritmo Merge-Sort recursivo (Ejemplo 05).

En este caso a es un arreglo de n elementos y p & r son los índices del rango a ordenar.

```
void MergeSort(a, p, r) {  
    if (p < r) {  
        q = parteEntera((q+r)/2);  
        MergeSort(a, p, q);  
        MergeSort(a, q+1, r);  
        Merge(a, p, q, r);  
    }  
}
```

El análisis para este problema es el siguiente.

Para el algoritmo Merge-Sort tenemos lo siguiente

$$T(n) = \begin{cases} 1, & n=1 \\ 2T(\frac{n}{2}) + n, & n>1 \end{cases}$$

Utilizamos el teorema maestro con $f(n)=n$, $a=2$ & $b=2$
para el caso 2:
 $f(n) = n = \Theta(n^{\log_2 2}) = \Theta(n)$
Por lo tanto $T(n) = \Theta(n^{\log_2 2} \log n) = \Theta(n \log n)$