**Instituto Politecnico Nacional**

**ESCOM “ESCUELA SUPERIOR DE CÓMPUTO”**

*ANÁLISIS DE ALGORITMOS*

*EJERCICIO 5: ALGORITMOS RECURSIVOS*

PROFE: Edgardo Adrián Franco Martínez

ALUMMNO: Rojas Alvarado Luis Enrique

GRUPO: 3CM4

# ANÁLISIS DE ALGORITMOS RECURSIVOS

1. Calcular la cota de complejidad para el algoritmo de la siguiente función recursiva

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

Cuando entra en el primer if cuando comprueba que el número es igual con cero solo hace un return, por lo que el caso base es T(0) = 2 después tomando en cuenta que si ésta primera condición no se cumple, se tiene que cumplir el segundo if, que comprueba que el numero sea 1 y e cuenta una asignación, un ciclo for, una multiplicación dentro de él y un return por lo que se suman 2 pruebas anteriores, más 4 operaciones internas dentro del if queda el segundo caso base como: T(1) = 6. Por último si no es ninguna de las anteriores se cuenta las 2 pruebas anteriores, un return, una operación matemática y las funciones recursivas, por lo que la recursión queda como: 4 + T(n-1) + T(n-2).

$$T(n) = \begin{cases} T(0) = 2 & n = 0 \text{ Primer caso base.} \\ T(1) = 6 & n = 1 \text{ Segundo caso base.} \\ 4 + T(n-1) + T(n-2) & n > 1 \text{ Recursión lineal no homogénea.} \end{cases}$$

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

$K = 2;\ p(n) = 4;\ b = 1;\ d = 0 \Rightarrow b^n\, p(n) = 1^n\,(4)$

Haciendo $x^k \Rightarrow (x^2 - x - 1)(x - b)^{d+1} = 0$

$(x^2 - x - 1)(x - 1) = 0 \Rightarrow$ Factorizando:

$r_1 = 1;\ r = \frac{-b \pm \sqrt{b^2-4ac}}{2a} = \frac{-(-1) \pm \sqrt{(-1)^2-4(1)(-1)}}{2(1)} = \frac{1 \pm \sqrt{5}}{2} \Rightarrow r_2 = \frac{1+\sqrt{5}}{2}\ ;\ r_3 = \frac{1-\sqrt{5}}{2}$

$T(n) = C_1\,(1^n) + C_2(\frac{1+\sqrt{5}}{2})^n + C_3(\frac{1-\sqrt{5}}{2})^n$

Si $C_2 \neq 0 \Rightarrow T(n) \in O(\frac{1+\sqrt{5}}{2})^n$

2. Calcular la complejidad de implementación recursiva del producto.

```
int Producto( int a, int b)
{

        if (b==0)
                return 0;
        else
                return a + Producto(a,b-1);
}
```

Primero comprueba si b es igual con cero, y posteriormente hace un return, así que el caso base queda como: T(0) = 2. Tomando en cuenta que si no se cumple ésta primer condición, tiene que entrar al else, por lo que hace un return, una suma y la función es llamada a sí misma, quedando la recurrencia cómo: 4 + T(n-1).

$$T(n) = \begin{cases} T(0) = 2 & n = 0 \text{ Primer caso base.} \\ 4 + T(n-1) & n > 1 \text{ Recursión lineal no homogénea.} \end{cases}$$

$T(n) – T(n-1) = 4$

$K = 1;\ p(n) = 4;\ b = 1;\ d = 0 \Rightarrow b^n\, p(n) = 1^n\,(4)$

Haciendo $x^k \Rightarrow (x – 1)(x – b)^{d+1} = 0$

$(x – 1)(x – 1) = 0 \Rightarrow (x – 1)^2 = 0 \Rightarrow r_{1,\,2} = 1$

$T(n) = C_1\,(1^n) + C_2\,n(1^n)$

Si $C_2 \neq 0 \Rightarrow T(n) \in O(n1^n)$

3. Calcular el costo de un recorrido In-Orden de un árbol binario completamente lleno.

```
void TraverseInorder (TreeNode root)
{
        if(root!=null)
        {
                TraverseInorder(root.getLeft());
                process (root.getValue());
                TraverseInorder(root.getRight());
        }
}
```

Está comprobando que la raíz sea diferente de null, por lo que si es null simplemente queda como T(0) = 1, si no es null la función se llama así misma 2 veces y hace una llamada a una función externa, quedando la recurrencia como: 1 + 2T(n-1).

$$T(n) = \begin{cases} T(0) = 1 & n = null \quad \text{Primer caso base.} \\ 1 + 2T(n-1) & n \; != null \quad \text{Recursión lineal no homogénea.} \end{cases}$$

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

$K = 1;\ p(n) = 1;\ b = 1;\ d = 0 \Rightarrow b^n\, p(n) = 1^n\,(4)$

Haciendo $x^k \Rightarrow (x - 2)(x - b)^{d+1} = 0$

$(x - 2)(x - 1) = 0;\ r_1 = 2;\ r_2 = 1$

$T(n) = C_1(2^n) + C_2(1^n)$

Si $C_1 \neq 0 \Rightarrow T(n) \in O(2^n)$

4. Calcular la cota de complejidad del algoritmo de búsqueda ternaria.

```
double BusquedaTernaria(double f[] ,int l ,int r ,double absolutePrecision )
{
            if(r-l <=  absolutePrecision)
            {
                        return  ( l  +  r ) / 2.0;
            }
            else
            {
                int m1  =  ( 2 * l  +  r ) / 3;
                           int m2  =  ( l  +  2 * r ) / 3;

                if(f[m1]< f[m2])
                {
                return  BusquedaTernaria( f ,  m1 ,  r ,  absolutePrecision );
                           }
            else
                {
                                                return  BusquedaTernaria ( f ,  l ,  m2 , absolutePrecision );
                           }
            }
}
```

Primero hace una comparación y dentro de la condición ésta una resta (2), cuando se entra al primer caso base hace una división, una suma y un return (3). Por lo que el costo del primer caso base es de T(n-1) = 5. Si no se cumple esa condición hace primero una asignación de 3 operaciones aritméticas en la cual r se multiplica r por 2 y se divide entre 3 en 2 variables (8). Posteriormente hace una validación, la cual retorna la función recursiva, pero como anteriormente se partió la variable r. La recursión queda como $T(n) = T(\frac{2n}{3}) + 10$

$$T(n) = \begin{cases} T(n-1) = 5 & \text{Primer caso base.} \\ T(\frac{2n}{3}) + 10 & \text{Modelo recursivo. (Teorema maestro).} \end{cases}$$

Resolviendo por teorema maestro. $T(n) = T((\frac{2n}{3}) + 10$

a = 1; b = $(\frac{3}{2})$; f(n) = 10 => comprobamos $a = b^c$ => $1 = (\frac{3}{2})^0$ => $\Theta(n^c \ln(n))$

$T(n) = \Theta(n^0 \ln(n)) = \Theta(\log n)$

5. Calcular la cota de complejidad del algoritmo de ordenamiento QuickSort.

```
QuickSort(lista, inf, sup)
{
    elem_div = lista[sup];
    i = inf - 1;
    j = sup;
    cont = 1;

            if (inf >= sup)
                    return;

    while (cont)
            {
                    while (lista[++i] < elem_div);
                    while (lista[--j] > elem_div);
                    if (i < j)
                    {
                            temp = lista[i];
                            lista[i] = lista[j];
                            lista[j] = temp;
                    }
                    else
                    {
                            cont = 0;
                    }
            }

            temp = lista[i];
            lista[i] = lista[sup];
            lista[sup] = temp;

            QuickSort (lista, inf, i - 1);
            QuickSort (lista, i + 1, sup);
}
```

Primero se toma el primer elemento como pivote, posteriormente se hace la búsqueda por la izquierda (i) y después por la izquierda (j), mientras las búsquedas no se crucen busca el elemento mayor que pivote y busca el elemento menor que pivote, si no se han cruzado los intercambia. Posteriormente se coloca el pivote en el lugar en donde tendremos los menores a la izquierda y los mayores a la derecha. La función se llama así misma para ordenar el subarray izquierdo y luego se vuelve a llamar para acomodar el subarray derecho [2T(n)]. Debido a que el ordenamiento divide una lista y cada sublista genera en promedio dos sublistas más: $(\frac{n}{2})$ haciendo esto n veces del array quedando la recurrencia como $2T((\frac{n}{2}) + n$. El caso base queda como T(1) = 2, puesto que solo es una comparación y un return.

$$T(n) = \begin{cases} T(1) = 2 & \text{Primer caso base.} \\ 2T(\frac{n}{2}) + n & \text{Modelo recursivo. (Teorema maestro).} \end{cases}$$

Resolviendo por teorema maestro: $T(n) = 2T(\frac{n}{2}) + n$

$a = 2;\ b = 2;\ f(n) = n$ => comprobamos $a = b^c$ => $2 = 2^1$ => $\Theta(n^c \ln(n))$

$T(n) = \Theta(n^1 \ln(n)) = \Theta(n \log n)$

6. Resolver las siguientes ecuaciones y dar su orden de complejidad:

**6.1) T(n)= 3T(n-1) + 4T(n-2) => n>1; T(0)=0; T(1)=1**

Ésta ecuación se tratará como una homogénea.

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

Reescribiendo en términos de $x^k$.

$x^2 - 3x - 4 = 0$ Por lo que la factorización queda como $(x - 4)(x + 1)=0$ y por lo tanto sus raíces son: $r_1 = 4$ y $r_2 = -1$

Para que al final la ecuación quede como: $C_1(4^n) + C_2(-1)^n$

Por condiciones iniciales tenemos que:

$T(0) = 0$ => $C_1(4^{(0)}) + C_2(-1)^{(0)} = 0$ => $C_1 + C_2 = 0$
$T(1) = 1$ => $C_1(4^{(1)}) + C_2(-1)^{(1)} = 1$ => $4C_1 - C_2 = 1$

Resolviendo el sistema de ecuaciones por método de Gauss-Jordan:

$$\left(\begin{array}{cc|c} 1 & 1 & 0 \\ 4 & -1 & 1 \end{array}\right) \times(-4) \xrightarrow{F_2 - 4 \times F_1 \to F_2} \left(\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & -5 & 1 \end{array}\right) \times\left(\frac{-1}{5}\right) \xrightarrow{F_2/(-5) \to F_2}$$

$$\left(\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 1 & \frac{-1}{5} \end{array}\right) \times(-1) \xrightarrow{F_1 - 1 \times F_2 \to F_1} \left(\begin{array}{cc|c} 1 & 0 & \frac{1}{5} \\ 0 & 1 & \frac{-1}{5} \end{array}\right)$$

Por lo que podemos concluir que $C_1 = \frac{1}{5}$; $C_2 = \frac{-1}{5}$ y sustituyendo en la ecuación original nos queda:

$T(n) = C_1(4^n) + C_2(-1)^n = \frac{1}{5}(4^n) + \frac{-1}{5}(-1)^n = \frac{1}{5}(4^n - (-1)^n)$ => $T(n) \in O(4^n)$

**6.2) $T(n)= 3T(n-1) + 4T(n-2) + (n+5)2^n$ => n>1; T(0)=5 T(1)=27**

Ésta ecuación tiene 1 parte homogénea y una no homogénea. Primero se resuelve la parte homogénea.

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

Haciendo $x^k$ => $x^2 – 3x – 4 = 0$ Quedando como factorización $(x – 4)(x + 1)=0$ y siendo las raíces cómo $r_1 = 4$ y $r_2 = -1$ y la ecuación queda como:

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

Resolviendo la parte no homogénea se propone $T_{p(n)}= (An + B)2^n$ y sustituimos en la ecuación original:

$(An + B)2^n = 3[A(n - 1) + B]2^{n-1} + 4[A(n - 2) + B]2^{n-2} + (n+5)2^n$

Tenemos que dejar todo en términos de $2^n$ multiplicando todo por (2)(2) Para poder eliminar el exponente que la n se resta con una constante por leyes de los exponentes.

$(An + B)2^n(2)(2) = 3[A(n - 1) + B]2^{n-1}(2)(2) + 4[A(n - 2) + B]2^{n-2}(2)(2) + (n+5)2^n(2)(2)$

$4(An + B)2^n = 6[A(n - 1) + B]2^n + 4[A(n - 2) + B]2^n + 4(n+5)2^n$

$4(An + B)2^n = \{6[A(n - 1) + B] + 4[A(n - 2) + B] + 4(n+5)\} 2^n$ => Factorizando.

$4(An + B) = 6[A(n - 1) + B] + 4[A(n - 2) + B] + 4(n+5)$ => Se elimina $2^n$.

$4An + 4B = 6An – 6A + 6B + 4An – 8A + 4B + 4n + 20$ => Sumando términos semj.

$4An + 4B = (10A + 4)n – 14A + 10B + 20$

Igualando el término lineal.

$4A = (10A + 4)$ => $4A – 10A = 4$ => $-6A = 4$ => $A=-\frac{2}{3}$

Igualando el término independiente.

$4B = – 14A + 10B + 20$ => $4B – 10B = -14(-\frac{2}{3}) + 20$ => $-6B = \frac{88}{3}$ => $B = -\frac{44}{9}$

Entonces la ecuación queda como:

$T_{p(n)}= (An + B)2^n = (-\frac{2}{3}n + -\frac{44}{9}) 2^n$

Por último. La recurrencia es la suma de las 2 soluciones:

$T(n) = C_1(4^n) + C_2(-1)^n + (-\frac{2}{3}n + -\frac{44}{9})\ 2^n$

De las condiciones iniciales:

$T(0) = 5 \Rightarrow C_1(4^0) + C_2(-1)^0 + (-\frac{2}{3}(0) + -\frac{44}{9})\ 2^0 = 5 \Rightarrow C_1 + C_2 - \frac{44}{9} = 5$

$T(1) = 27 \Rightarrow C_1(4^1) + C_2(-1)^1 + (-\frac{2}{3}(1) + -\frac{44}{9})\ 2^1 = 27 \Rightarrow 4C_1 - C_2 - \frac{100}{9} = 27$

Resolviendo el sistema de ecuaciones por método de Gauss-Jordan.

$C_1 + C_2 = 5 + \frac{44}{9} \Rightarrow C_1 + C_2 = \frac{89}{9}$

$4C_1 - C_2 = 27 + \frac{100}{9} \Rightarrow 4C_1 - C_2 = \frac{343}{9}$

$$\left(\begin{array}{cc|c} 1 & 1 & \frac{89}{9} \\ 4 & -1 & \frac{343}{9} \end{array}\right) \times(-4) \quad F_2 - 4 \times F_1 \to F_2 \quad \left(\begin{array}{cc|c} 1 & 1 & \frac{89}{9} \\ 0 & -5 & \frac{-13}{9} \end{array}\right) \times\left(\frac{-1}{5}\right) \quad F_2/(-5) \to F_2$$

$$\left(\begin{array}{cc|c} 1 & 1 & \frac{89}{9} \\ 0 & 1 & \frac{13}{45} \end{array}\right) \times(-1) \quad F_1 - 1 \times F_2 \to F_1 \quad \left(\begin{array}{cc|c} 1 & 0 & \frac{48}{5} \\ 0 & 1 & \frac{13}{45} \end{array}\right)$$

Podemos decir que $C_1 = \frac{48}{5}$ ; $C_2 = \frac{13}{45}$ y sustituyendo en la ecuación original.

$T(n) = C_1(4^n) + C_2(-1)^n + (-\frac{2}{3}n + -\frac{44}{9})\ 2^n = \frac{48}{5}(4^n) + \frac{13}{45}(-1)^n + (-\frac{2}{3}n + -\frac{44}{9})$

Por lo tanto el orden de complejidad del algoritmo: Escriba aquí la ecuación.$T(n) \in O(4^n)$

**6.3) $T(n) - 2T(n-1) = 3^n \quad \Rightarrow \quad n>=2; \quad T(0)=0; \quad T(1)=1.$**

De igual manera que el problema anterior, se puede dividir en una parte homogénea y una no homogénea. Siendo la que se resolverá primero la parte homogénea.

$T(n) - 2T(n-1) = 0 \Rightarrow$ Haciendo $x^k \Rightarrow x - 2 = 0$ Por lo que la única raíz es $x = 2$ y esto hace que $T(n) = C_1(2^n)$

Resolviendo la parte no homogénea. Se propone $T_{p(n)} = A(3^n)$ y sustituimos en la ec.

$A(3^n) = 2A(3)^{n-1} + 3^n$Entonces se deja todo en términos de $3^n$ multiplicando por 3 la ec

$3A(3^n) = 2A(3)^{n-1}(3) + 3(3)^n$

$3A(3^n) = 2A(3)^n + 3(3)^n \Rightarrow$Se factoriza por término común $(3)^n$

$3A(3^n) = [2A + 3](3)^n$

$3A = [2A + 3] \Rightarrow 3A - 2A = 3 \Rightarrow A = 3$

Por lo tanto $T_{p(n)} = A(3^n) \Rightarrow T_{p(n)} = 3(3^n)$

Se suman las 2 soluciones de las partes homogénea y la no homogénea.

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

Por las condiciones iniciales. $T(0) = 0$ Se tiene que cumplir que $C_1 + 3 = 0$ Por lo que despejando queda como: $C_1 = -3$

Por lo tanto: $T(n) = C_1(2^n) + 3(3^n) = T(n) = (-3)(2^n) + 3(3^n)$ Esto hace el orden de la ecuación recurrente como $T(n) \epsilon O(3^n)$

Las 2 condiciones iniciales no se pueden cumplir simultáneamente. Si se elige la condición inicial de $T(1) = 1$ La ecuación sale de tipo $T(n) = 3^{n+1} - 2^{n+2}$

7. Calcular la cota de complejidad que tendrían los algoritmos con los siguientes modelos recurrentes.

   Para resolver los siguientes modelos recurrentes se utilizó la estrategia proporcionada por el profesor.

$$T(n) = \begin{cases} d & n = 1 \\ aT(n/b) + n^c & n > 1 \end{cases} \quad T(n) = \begin{cases} O(n^c) & a < b^c \\ \Theta(n^c \lg n) & a = b^c \\ \Theta(n^{\log_b a}) & a > b^c \end{cases}$$

**7.1) $T(n)= 3T(\frac{n}{3}) + 4T(\frac{n}{2}) + 2n^2 + n$**

El problema se puede dividir en dos ecuaciones para su resolución.

$T(n)= 3T(\frac{n}{3}) + n$ Aplicamos teorema maestro.

$a = 3;\ b = 3;\ f(n) = n \Rightarrow$ comprobamos $a = b^c \Rightarrow 3 = 3^1 \Rightarrow \Theta(n^c \ln(n))$

Por lo tanto $T(n) = (n \ln(n))$

$T(n) = 4T(\frac{n}{2}) + 2n^2$

$a = 4;\ b = 2;\ f(n) = 2n^2 \Rightarrow$ comprobamos $a = b^c \Rightarrow 4 = 2^2 \Rightarrow \Theta(n^c \ln(n))$

Por lo tanto T(n) = ($n^2$ ln(n)) Y entonces podemos decir que la cota de complejidad de éste algoritmo recursivo es T(n) ϵ O($n^2$ log n)

**7.2) T(n)= T(n-1) + T(n-2) + T($\frac{n}{2}$) si n>1;  T(0)=1;  T(1)=1**

Éste algoritmo se Puede dividir en 2 partes, una se resuelve de manera no homogénea, y la segunda con el teorema maestro.

Resolviendo primero por teorema maestro: T($\frac{n}{2}$)

a = 1;  b = 2;  f(n) = 0 => comprobamos a = $b^c$ => 1 = $2^0$ => ɵ($n^c$ ln(n))

T(n) = ɵ($n^0$ ln(n)) = ɵ(ln(n))

Resolviendo la parte no homogénea.

T(n) - T(n-1) - T(n-2) = 0 Pasando a $x^k$ => $x^2$ – x – 1 = 0 => Por fórmula general:

$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a} = \frac{-(-1) \pm \sqrt{(-1)^2-4(1)(-1)}}{2(1)} = \frac{1 \pm \sqrt{5}}{2}$ => $X_1 = \frac{1+\sqrt{5}}{2}$ ; $X_2 = \frac{1-\sqrt{5}}{2}$

T(n) = $C_1(\frac{1+\sqrt{5}}{2})^n + C_2(\frac{1-\sqrt{5}}{2})^n$

Si $C_1 \neq 0$ T(n) ϵ O($\frac{1+\sqrt{5}}{2}$)$^n$

Se toma la complejidad más alta siendo la última de orden constante elevado a la n que es más eficiente que log n.

**7.3) T(n)= T($\frac{n}{2}$) +2T($\frac{n}{4}$) + 2**

Se puede dividir en 2 ecuaciones a resolver por el teorema maestro.

T(n)= T($\frac{n}{2}$)

a = 1;  b = 2;  f(n) = 0 => se comprueba a = $b^c$ => 1 = $2^0$ => ɵ($n^c$ ln(n))

T(n) = ɵ($n^0$ ln(n)) = ɵ(ln(n))

T(n)= 2T($\frac{n}{4}$) + 2

a = 2;  b = 4;  f(n) = 2 => se comprueba a > $b^c$ => 2 > $4^0$ => ɵ($n^{\log_b a}$)

$T(n) = \theta(n^{\log_4 2}) = \theta(n^{1/2})$

Por lo tanto la complejidad de la ecuación recurrente es: $T(n) \in O(n^{1/2})$

**7.4) $T(n)= 2T(\frac{n}{2}) + 4T(\frac{n}{4}) +10n^2 + 5n$**

Se divide en dos ecuaciones a resolver por teorema maestro.

$T(n)= 2T(\frac{n}{2}) + 5n$

$a = 2;\ b = 2;\ f(n) = 5n$ => se comprueba $a = b^c$ => $2 = 2^1$ => $\theta(n^c \ln(n))$

$T(n) = \theta(n^1 \ln(n)) = \theta(n \ln(n))$

$T(n)= 4T(\frac{n}{4}) +10n^2$

$a = 4;\ b = 4;\ f(n) = 10n^2$ => se comprueba $a < b^c$ => $4 < 4^2$ => $\theta(n^c)$

$T(n) = \theta(n^2)$

Por lo que la complejidad de la ecuación recurrente es: $T(n) \in O(n^2)$