INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

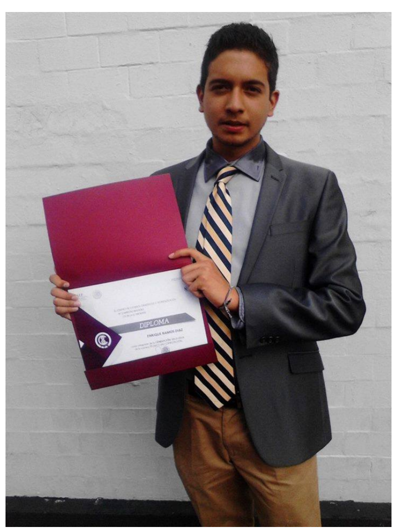
Ejercicio 05: Análisis de algoritmos recursivos

Unidad de aprendizaje: Análisis de Algoritmos

Grupo: 3CM3

Alumno:
Ramos Diaz Enrique

Profesor(a):
Franco Martínez Edgardo Adrián



23 de Octubre 2018

Índice

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);
}
```

Operaciones básicas para el análisis de la complejidad: aritméticas, retornos.

1.1. Ecuaciones de recurrencia

$T(0) = 1 \Longleftarrow$ 1 retorno

$T(1) = 3 \Longleftarrow$ 2 aritméticas y 1 retorno

$T(n) = 1 + T(n-1) + T(n-2) \Longleftarrow$ 1 aritmética y 2 llamadas recursivas

$T(2) = 1 + T(1) + T(0) = 5$

1.2. Análisis del orden de complejidad

Reordenando términos

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

Recurrencia lineal no homogénea. Haciendo el cambio $x^k = T(n)$ con k = 2, b = 1, d = 0 obtenemos su ecuación característica:

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

Calculamos sus raíces:

Raíces distintas $\begin{cases} r_1 = & \frac{1+\sqrt{5}}{2} \\ r_2 = & \frac{1-\sqrt{5}}{2} \\ r_3 = & 1 \end{cases}$

NOTA:

Raíces iguales: $\displaystyle\sum_{i=1}^{k+1} c_i n^{i-1} r_i^n$

Raíces distintas: $\sum_{i=1}^{k+1} c_i r_i^n$

Sustituimos:

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

Creamos el sistema de ecuaciones, sustituyendo n

$T(0) = 1 = C_1 + C_2 + C_3$

$T(1) = 3 = C_1 \left(\frac{1+\sqrt{5}}{2}\right) + C_2 \left(\frac{1-\sqrt{5}}{2}\right) + C_3$

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

Resolviendo:

$C_1 = 2.34$, $C_2 = -0.35$, $C_3 = -1$

Finalmente obtenemos la cota de complejidad del algoritmo:

$T(n) = 2.34 \left(\frac{1+\sqrt{5}}{2}\right)^n - 0.35 \left(\frac{1-\sqrt{5}}{2}\right)^n - 1 \in O(1.61803^n)$

2. Calcular la complejidad de la implementación recursiva del producto

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

Operaciones básicas para el análisis de la complejidad: retornos, aritméticas.

2.1. Ecuaciones de recurrencia

$T(0) = 1 \Longleftarrow$ 1 retorno

$T(n) = 1 + T(n-1) \Longleftarrow$ 1 aritmética y 1 llamada recursiva

$T(1) = 1 + T(0) = 2$

2.2. Análisis del orden de complejidad

Reordenando términos

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

Recurrencia lineal no homogénea. Haciendo el cambio $x^k = T(n)$ con k = 1, b = 1, d = 0 obtenemos su ecuación característica:

$(x-1)(x-1)=0$

Calculamos sus raíces:

Raíces iguales $\begin{cases} r_1 = 1 \\ r_2 = 1 \end{cases}$

Sustituimos:

$T(n) = C_1 + C_2 n$

Creamos el sistema de ecuaciones, sustituyendo n

$T(0) = 1 = C_1$

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

Resolviendo:

$C_1 = 1, C_2 = 1$

Finalmente obtenemos la cota de complejidad del algoritmo:

$T(n) = 1 + n \in O(n)$

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());
   }
}
```

Operaciones básicas para el análisis de la complejidad: obtener raíz

3.1. Ecuaciones de recurrencia

$T(0) = 0$

$T(n) = 1 + 2T(\frac{n}{2}) \Longleftarrow$ 1 raíz y 2 llamadas recursivas con las mitades del árbol

3.2. Análisis del orden de complejidad

Recurrencia no lineal. Utilizamos el Teorema maestro

$T(n) = 2T(\frac{n}{2}) + 1$

En donde a = 2, b = 2, $f(n) = 1 \in O(1)$

- Primer caso

  $O(n^{\log_2 2-\varepsilon})$ con $\varepsilon = 1$

  $= O(n^{\log_2 2-1}) = O(n^{1-1}) = O(n^0) = O(1) = f(n) \therefore T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$

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

```
double BusquedaTernaria(double f[], int l, int r, double absolutePrecision){
   if(r-1 <= absolutePrecision){
      return (1 + 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);
      }
   }
}
```

Operaciones básicas para el análisis de la complejidad: comparaciones con la variable r

4.1. Ecuaciones de recurrencia

$T(1) = 1 \Longleftarrow$ 1 comparación

$T(n) = 1 + T(\frac{2n}{3}) \Longleftarrow$ 1 comparación y 2 llamadas recursivas [m1 = (2*l + r) / 3]

4.2. Análisis del orden de complejidad

Recurrencia no lineal. Utilizamos el Teorema maestro

$T(n) = T(\frac{2n}{3}) + 1$

En donde a = 1, b = $\frac{3}{2}$, $f(n) = 1 \in O(1)$

- Primer caso

  $O(n^{\log_{1.5} 1-\varepsilon})$ No nos sirve para ningún ε...

- Segundo caso

  $O(n^{\log_{1.5} 1}) = O(n^0) = O(1) = f(n)$

  $\therefore T(n) = \Theta(n^{\log_{1.5} 1} \log n) = \Theta(\log n)$

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);
}
```

Operaciones básicas para el análisis de la complejidad: comparaciones entre las variables inf y sup.

5.1. Ecuaciones de recurrencia (Peor caso)

$T(0) = 0$

$T(1) = 1 \Longleftarrow$ 1 comparación

$T(n) = n + T(n-1) \Longleftarrow$ n comparaciones y el pivote no separa nada, recorriendo el arreglo hasta la última posición.

$T(2) = 3$

5.2. Análisis del orden de complejidad

Reordenando términos

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

Recurrencia lineal no homogénea. Haciendo el cambio $x^k = T(n)$ con k = 1, b = 1, d = 1, P(n) = n obtenemos su ecuación característica:

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

Calculamos sus raíces:

Raíces iguales $\begin{cases} r_1 = 1 \\ r_2 = 1 \\ r_3 = 1 \end{cases}$

Sustituimos:

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

Creamos el sistema de ecuaciones, sustituyendo n

$T(0) = 0 = C_1$

$T(1) = 1 = C_1 + C_2 + C_3$

$T(2) = 3 = C_1 + 2C_2 + 4C_3$

Resolviendo:

$C_1 = 0$, $C_2 = 0.5$, $C_3 = 0.5$

Finalmente obtenemos la cota de complejidad del algoritmo:

$T(n) = 0 + 0.5n + 0.5n^2 \in O(n^2)$

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

6.1. Ecuación 1

$T(n) = 3T(n-1) + 4T(n-2) \Rightarrow n > 1$

$T(0) = 0$

$T(1) = 1$

Reordenando términos

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

Recurrencia lineal homogénea. Haciendo el cambio $x^k = T(n)$ obtenemos su ecuación característica:

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

Calculamos sus raíces:

Raíces distintas $\begin{cases} r_1 = -1 \\ r_2 = 4 \end{cases}$

Sustituimos:

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

Creamos el sistema de ecuaciones, sustituyendo n

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

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

Resolviendo:

$C_1 = -0.2$, $C_2 = 0.2$

Finalmente obtenemos la cota de complejidad del algoritmo:

$T(n) = (-0.2)(-1)^n + (0.2)(4)^n \in O(4^n)$

6.2. Ecuación 2

$T(n) = 3T(n-1) + 4T(n-2) + (n+5)2^n \Rightarrow n > 1$

$T(0) = 5$

$T(1) = 27$

Reordenando términos

$T(n) - 3T(n-1) - 4T(n-2) = (n+5)2^n$

Recurrencia lineal no homogénea. Haciendo el cambio $x^k = T(n)$ con k = 2, b = 2^x, d = 1, P(n) = n+5 obtenemos su ecuación característica:

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

Calculamos sus raíces:

Raíces distintas $\begin{cases} r_1 = -1 \\ r_2 = 4 \end{cases}$

Sustituimos (hay que sumar b a la ecuación de concurrencia):

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

Creamos el sistema de ecuaciones, sustituyendo n

$T(0) = 5 = C_1 + C_2$

$T(1) = 27 = -C_1 + 4C_2 + 2$

Resolviendo:

$C_1 = -1, C_2 = 6$

Finalmente obtenemos la cota de complejidad del algoritmo:

$T(n) = (-1)(-1)^n + (6)(4)^n + 2^n \in O(4^n)$

6.3. Ecuación 3

$T(n) - 2T(n-1) = 3^n \Rightarrow n\geq 2$

$T(0) = 0$

$T(1) = 1$

Recurrencia lineal no homogénea. Haciendo el cambio $x^k = T(n)$ con k = 1, b = 3^x, d = 0, P(n) = 1 obtenemos su ecuación característica:

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

Calculamos sus raíces:

$r_1 = 2$

Sustituimos (hay que sumar b a la ecuación de concurrencia):

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

Para encontrar el valor de C_1, utilizamos el caso en donde n = 1

$T(1) = 1 = 2C_1 + 3$

Resolviendo:

$C_1 = -1$

Finalmente obtenemos la cota de complejidad del algoritmo:

$T(n) = (-1)(2)^n + 3^n \in O(3^n)$

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

7.1. Ecuación 1

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

Recurrencia no lineal. Utilizamos el Teorema maestro.

Dividimos la ecuación en dos partes:

Parte 1:

$T(n) = 3T(\frac{n}{3})$

En donde a = 3, b = 3, $f(n) = 0 \in O(1)$

- Primer caso

  $O(n^{\log_3 3-\varepsilon})$ con $\varepsilon = 1$

  $= O(n^{\log_3 3-1}) = O(n^{1-1}) = O(n^0) = O(1) = f(n)$

  $\therefore T(n) = \Theta(n^{\log_3 3}) = \Theta(n)$

Parte 2:

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

En donde a = 4, b = 2, $f(n) = 2n^2 + n \in O(n^2)$

- Primer caso

  $O(n^{\log_2 4-\varepsilon})$ No nos sirve para ningún ε...

- Segundo caso

  $O(n^{\log_2 4}) = O(n^2) = f(n)$

  $\therefore T(n) = \Theta(n^{\log_2 4} \log n) = \Theta(n^2 \log n)$

Finalmente, unimos ambas cotas de complejidad:

$T(n) = \Theta(n) + \Theta(n^2 \log n) \in \Theta(n^2 \log n)$

7.2. Ecuación 2

$T(n) = T(n-1) + T(n-2) + T(\frac{n}{2}) \Rightarrow n > 1$

$T(0) = 1$

$T(1) = 1$

Dividimos la ecuación en dos partes:

Parte 1:

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

Recurrencia lineal homogénea. Nos damos cuenta que es la misma ecuación que la del Algoritmo recursivo 1 (FuncionRecursiva) pero homogénea, por ende, sus raíces serán las mismas, excepto x = 1.

Raíces distintas $\begin{cases} r_1 = & \frac{1+\sqrt{5}}{2} \\ r_2 = & \frac{1-\sqrt{5}}{2} \end{cases}$

Sustituimos:

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

Creamos el sistema de ecuaciones, sustituyendo n

$T(0) = 1 = C_1 + C_2$

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

Resolviendo:

$C_1 = 0.72, C_2 = 0.27$

Finalmente obtenemos la cota de complejidad:

$T(n) = 0.72\left(\frac{1+\sqrt{5}}{2}\right)^n + 0.27\left(\frac{1-\sqrt{5}}{2}\right)^n \in O(1.61803^n)$

Parte 2:

Recurrencia no lineal. Utilizamos el Teorema maestro.

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

En donde a = 1, b = 2, $f(n) = 0 \in O(1)$

- Primer caso

  $O(n^{\log_2 1-\varepsilon})$ No nos sirve para ningún ε...

- Segundo caso

  $O(n^{\log_2 1}) = O(n^0) = O(1) = f(n)$

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

Finalmente, unimos ambas cotas de complejidad:

$T(n) = O(1.61803^n) + \Theta(\log n) \in \Theta(1.61803^n)$

7.3. Ecuación 3

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

Recurrencia no lineal. Utilizamos el Teorema maestro.

Dividimos la ecuación en dos partes:

Parte 1:

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

Esta cota ya la calculamos en la Ecuación 2: $T(n) = \Theta(n^{\log_2 1}\log n) = \Theta(\log n)$

Parte 2:

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

En donde a = 2, b = 4, $f(n) = 2 \in O(1)$

- Primer caso
  $O(n^{\log_4 2-\varepsilon})$ con $\varepsilon = 0.5$
  $= O(n^{\log_4 2-0.5}) = O(n^{0.5-0.5}) = O(n^0) = O(1) = f(n)$
  $\therefore T(n) = \Theta(n^{\log_4 2}) = \Theta(n^{0.5})$

Finalmente, unimos ambas cotas de complejidad:

$T(n) = \Theta(\log n) + \Theta(n^{0.5}) \in \Theta(n^{0.5})$

7.4. Ecuación 4

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

Recurrencia no lineal. Utilizamos el Teorema maestro.

Dividimos la ecuación en dos partes:

Parte 1:

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

Esta cota ya la calculamos en el algoritmo del recorrido In-orden de un Árbol Binario completamente lleno: $T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$

Parte 2:

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

En donde a = 4, b = 4, $f(n) = 10n^2 + 5n \in O(n^2)$

- Primer caso
  $O(n^{\log_4 4-\varepsilon})$ No nos sirve para ningún ε...
- Segundo caso
  $O(n^{\log_4 4}) = O(n^1) = O(n) \neq f(n)$
- Tercer caso
  $O(n^{\log_4 4+\varepsilon})$ con $\varepsilon = 1$
  $= O(n^{\log_4 4+1}) = O(n^{1+1}) = O(n^2) = f(n)$
  Comprobamos $f(\frac{n}{b}) \leq cf(n)$

$$\left(\frac{n}{4}\right)^2 \leq c(n^2)$$

$$\therefore T(n) = \Theta(n^2)$$

Finalmente, unimos ambas cotas de complejidad:

$T(n) = \Theta(n) + \Theta(n^2) \in \Theta(n^2)$