Eficiencia de Algoritmia

Leticia Blanco

Departamento de Informática - Sistemas
UMSS

21 de agosto de 2024

Contenido

Antecedentes

Análisis de eficiencia

Tiempo de ejecución
- Definición
- Ejemplos
- Resumen

Complejidad
- Definición
- Propiedades

Antecedentes

Correctitud
Eficiencia

La solución que se presenta debe ser eficiente.
Por lo que, las preguntas son:

- Cuánto tardan?
- Es posible medir la lógica de pensamiento?

Análisis de eficiencia

Por qué hacer análisis de eficiencia?

- Hay varias *soluciones* para un problema.
- Cuál es mejor?
- Cómo elegimos?

Uso de recursos: *procesador* / memoria
Presunción del *mundo ideal*

Análisis de eficiencia

- Tiempo de ejecución
- Complejidad

Tiempo de ejecución

Cuenta cada paso primitivo de un algoritmo.
$T_A(n)$: tiempo de ejecución del algoritmo A para una entrada de tamaño n

- Tiempo de ejecución: $T_A(n) \equiv f(n)$
- $T_A(n) \equiv \sum_{i=1}^{k} T_{p_i}(n)$
- $T_{p_i}(n) \equiv$ tiempo de ejecución del paso p_i

Expresado como una función matemática, para fines de comparación.

Tiempo de ejecución

Medir:

- Expresiones(calcular/comparar)
- Instrucciones (pasos)

Tiempo de ejecución - Expresiones

$$
E = \begin{cases}
var & => \quad T_E(n) = 1 \\
ctte & => \quad T_E(n) = 1 \\
E_1 \ op \ E_2 & => \quad T_E(n) = T_{E1}(n) + T_{E2}(n) + 1 \\
op \ E_1 & => \quad T_E(n) = T_{E1}(n) + 1 \\
(E_1) & => \quad T_E(n) = T_{E1}(n) \\
fn() & => \quad T_E(n) = T_{fn()}(n)
\end{cases}
$$

Tiempo de ejecución - Instrucciones

Declaración: **tipo var**

$$T(n) \equiv 0$$

Tiempo de ejecución - Instrucciones

Lectura: **leer var**

$$T_{in}(n) \equiv 1$$

Tiempo de ejecución - Instrucciones

Escritura: **mostrar E**

$$T_{out}(n) \equiv T_E(n)$$

Tiempo de ejecución - Instrucciones

Retorno: **return E**

$$T_{ret}(n) \equiv T_E(n)$$

Tiempo de ejecución - Instrucciones

Asignación: **var = E**

$$T_{asig}(n) \equiv T_E(n) + 1 \equiv T_E(n)$$

Tiempo de ejecución - Instrucciones

Condicional: **if_then_else**

```
1  if(expBool){
2      cuerpoSI
3  }else{
4      cuerpoNO
5  }
```

$$T_{if}(n) \equiv T_{expBool}(n) + max(T_{cuerpoSI}(n), T_{cuerpoNO}(n))$$

Tiempo de ejecución - Instrucciones

Repetición por conteo: **for**

```
for(cont = valorIni; cont <= valorFin; cont+=k){
    cuerpo
}
```

$$valorIni \leq cont \leq valorFin; +k$$

$$T_{for}(n) \equiv \frac{valorFin - valorIni}{k} + 2 + \left(\frac{valorFin - valorIni}{k} + 1\right) * T_{cuerpo}(n)$$

Tiempo de ejecución - Instrucciones

Repetición por condición: **while_do**

```
while(expBool){
    cuerpo
}
```

m ≡ numero de veces que la expBool se hace verdadera

$$T_{while}(n) \equiv (m+1) * T_{expBool}(n) + m * T_{cuerpo}(n)$$

Tiempo de ejecución - Instrucciones

Repetición por condición: **do_while**

```
do{
    cuerpo
}while(expBool);
```

m ≡ numero de veces que la expBool se hace verdadera

$$T_{do}(n) \equiv (m+1) * (T_{expBool}(n) + T_{cuerpo}(n))$$

Ejemplos

```
1 int mayor(int a, int b){
2   int c, d;
3   c = |a-b|;
4   d = a+b;
5   d = (c+d)/2;
6   return d;
7 }
```

Cuál es el T(n)?

Ejemplos

```
int mayor(int a, int b){
 int c, d;
 c = |a-b|;
 d = a+b;
 d = (c+d)/2;
 return d;
}
```

c = |a-b|;	4
d = a+b;	3
d = (c+d)/2;	5
return d;	1
	13

$T(n) \equiv 13$; es una función constante, independiente de n

Ejemplos

```
int menor(int a, int b, int c){
 int m;
 if(a<b){
    if(a<c){
       m = a;
    }else{
       m = c;
    }
 }else{
    if(b<c){
       m = b;
    }else{
       m = c;
    }
 }
 return m;
}
```

Cuál es el T(n)?

Ejemplos

```
1  int menor(int a, int b, int c){
2   int m;
3   if(a<b){                                          3
4      if(a<c){                          3
5         m = a;        1                +
6      }else{                   >
7         m = c;        1          1
8      }                            4
9   }else{                                   >        +
10     if(b<c){                       3
11        m = b;        1            +
12     }else{                   >                     4
13        m = c;        1          1
14     }                            4
15  }                                                    7
16  return m;                                            1
17 }                                                     8
```

$T(n) \equiv 8$; es una función constante, independiente de n

Ejemplos

```
int sumarDig(int n){
 int s;
 s = 0;
 while(n > 0){
    s = s + n%10;
    n = n/10;
 }
 return s;
}
```

Cuál es el T(n)?

Ejemplos

```
int sumarDig(int n){
 int s;
 s = 0;                                  1
 while(n > 0){                      (m+1)*3
    s = s + n%10;        5 |        |    +
    n = n/10;            3 | 8 * m  |  8*m
 }                                  11m+3
 return s;                               1
}                                   11m + 5
```

$T(n) \equiv 11m + 5$; pero que es m?

Ejemplos

```
while(n > 0){
    s = s + n%10;
    n = n/10;
}
```

n	$n > 0$	nro veces True
n	True	1
$n/10^1$	True	2
$n/10^2$	True	3
...	...	...
$n/10^{m-1}$	True	m
$n/10^m$	*False*	*m+1*

$$n/10^{m-1} = 1$$
$$n = 10^{m-1}$$
$$log_{10} n = m - 1$$
$$\log_{10} n + 1 = m$$

Ejemplos

```
int sumarDig(int n){
 int s;
 s = 0;                                  1
 while(n > 0){                      (m+1)*3
    s = s + n%10;        5 |        |    +
    n = n/10;            3 | 8 * m  |  8*m
 }                                  11m+3
 return s;                               1
}                                   11m + 5
```

$T(n) \equiv 11m + 5$

$T(n) \equiv 11 * (log_{10} n + 1) + 5$

$T(n) \equiv 11 * log_{10} n + 16$

$T(n)$ es una función logarítmica, dependiente de n

Ejemplos

```
1  int sumarDig(int n){
2    int s;
3    s = 0;                                  1
4    while(n > 0){                       (m+1)*3
5      s = s + n%10;        5 |          |    +
6      n = n/10;            3 | 8 * m    |  8*m
7    }                                  11m+3
8    return s;                               1
9  }                                     11m + 5
```

$T(n) \equiv 11m + 5$

$T(n) \equiv 11 * (log_{10} n + 1) + 5$

$T(n) \equiv 11 * log_{10} n + 16$

$T(n)$ es una función logarítmica, dependiente de n

$$\mathrm{T(n)} = \begin{cases} 11 * log_{10} n + 16 & <=> \quad n > 0 \\ 5 & <=> \quad caso\ contrario \end{cases}$$

Ejemplos

```
1 int sumar(int[] a){
2   int n, s;
3   n = a.length;
4   s = 0;
5   for(int i = 0; i < n; i++){
6       s = s + a[i];
7   }
8   return s;
9 }
```

Cuál es el T(n)?

Ejemplos

```
boolean sumar(int[] a){
 int n, s;
 n = a.length;                                  1
 s = 0;                                         1
 for(int i = 0; i <= n-1; i++){      | n-1-0+2 | n+1 |
     s = s + a[i];                 3 | *n      | 3n  |
 }                                       4n+1
 return s;                                      1
}                                          4 + 4n
```

$T(n) \equiv 4 + 4n$; es una función lineal, es dependiente del tamaño de entrada

Ejemplos

```
boolean capicua(int x){
 int c = 1;
 boolean res = true;
 while( x > 9){
  c++;
  x = x/10;
 }
 for(int i = 1; i <= c/2; i++){
  res = res && (x/Math.pow(10,i-1))%10
          == (x/Math.pow(10,c-i))%10;
 }
 return res;
}
```

Ejemplos

```
1   boolean capicua(int x){
2    int c = 1;                                                    1
3    boolean res = true;                                           1
4    while( x > 9){                                  |(m+1)*3     |
5     c++;                                   3|      |   +        |
6     x = x/10;                              3|*m    | 6m         |
7    }                                                 ========= 9m+3
8    for(int i = 1; i <= c/2; i++){                  |c/2-1+2     |
9     res = res&&(x/pow(10,i-1))%10                  |    +       |
10        ==(x/pow(10,c-i))%10;              13|*c/2-1+1| 13c/2    |
11   }                                                 ========= 7c+1
12   return res;                                                   1
13  }
```

Ejemplos

```
boolean capicua(int x){
 int c = 1;                                                  1
 boolean res = true;                                         1
 while( x > 9){                                   |(m+1)*3  |
  c++;                                   3|       |   +     |
  x = x/10;                              3|*m     |  6m     |
 }                                                 ========= 9m+3
 for(int i = 1; i <= c/2; i++){                   |c/2-1+2  |
  res = res&&(x/pow(10,i-1))%10                   |   +     |
      ==(x/pow(10,c-i))%10;          13|*c/2-1+1| 13c/2    |
 }                                                 ========= 7c+1
 return res;                                                 1
}
```

$$T_{capicua}(x) \equiv 1 + 1 + 9m + 3 + 7c + 1 + 1$$
$$\equiv 7 + 9m + 7c$$

Cuánto es m? Cuánto es c?

```
int c = 1;
while( x > 9){
  c++;
  x = x/10;
}
```

x	$x > 9$	nro veces True	c	
x	True	1	2	$x/10^{m-1} = 10$
$x/10^1$	True	2	3	$x = 10 * 10^{m-1}$
$x/10^2$	True	3	4	$x = 10^m$
...	...	...	...	$\log_{10} x = m$
$x/10^{m-1}$	True	m	m+1	
$x/10^m$	*False*	*m+1*		

$$m = log_{10}x; c = log_{10}x + 1$$

Ejemplos

$$T_{capicua}(x) \equiv 1 + 1 + 9m + 3 + 7c + 1 + 1$$
$$\equiv 7 + 9m + 7c \;\; T_{capicua}(x) \equiv 7 + 9(log_{10}x) + 7(log_{10}x + 1)$$
$$T_{capicua}(x) \equiv 14 + 16 * log_{10}x$$
$$\mathrm{T(n)} = \begin{cases} 14 + 16 * log_{10}x & <=> \quad x > 0 \\ 7 & <=> \quad caso\ contrario \end{cases}$$

Ejemplos

```
1 int sumaLenta (int a, int b]{
2   while(a > 0){
3     a= a-1;
4     b= b+1
5   }
6   return b;
7 }
```

Tiempo de ejecución - Tarea

```
1 void swap(int a, int b]{
2     int aux;
3     aux = a;
4     a = b;
5     b = aux;
6 }
```

Tiempo de ejecución - Tarea

```
1  int triangulares(int n){
2    int suma;
3    int contador;
4    suma= 0;
5    contador= 1;
6    while(contador <= n){
7      suma= suma + contador;
8      contador= contador + 1;
9    }
10   return suma;
11 }
```

Tiempo de ejecución

Todo proceso puede ser *"medido"* mediante una función matemática que determina el *Tiempo de Ejecución* del proceso.

Pero, es necesario encontrar el $T(n)$?, por lo menos a este nivel de detalle?

Pues no, lo único que interesa saber es cuán complejo es el proceso.

Complejidad

- Abstracción para representar funciones.
- Abstacción asintótica.
- Función representativa.

Complejidad

O

Para c, n_0 constantes positivas

$O(g(n)) \equiv \{f(n)/0 \leq f(n) \leq c * g(n), \forall n \geq n_0\}$

$f(n) = O(g(n)) \;\simeq\; f(n) \leq g(n)$

Ω

Para c, n_0 constantes positivas

$\Omega(g(n)) \equiv \{f(n)/0 \leq c * g(n) \leq f(n), \forall n \geq n_0\}$

$f(n) = \Omega(g(n)) \;\simeq\; g(n) \leq f(n)$

Θ

Para c_1, c_2, n_0 constantes positivas

$\Theta(g(n)) \equiv \{f(n)/0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n), \forall n \geq n_0\}$

$f(n) = \Theta(g(n)) \;\simeq\; g(n) < f(n) < g(n)$

Propiedades

- $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$
- $f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$

Teorema

$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$

- $f(n) = f_1(n) + f_2(n) \simeq f(n) = f_1(n) + \Theta(g(n))$;
  dado que $f_2(n) = \Theta(g(n))$
- $f(n) = f_1(n) + f_2(n) \simeq f(n) = \Theta(max(g_1(n), g_2(n)))$;
  dado que $f_1(n) = \Theta(g_1(n)) \wedge f_2(n) = \Theta(g_2(n))$

Propiedades

- *Transitividad*
  - $f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  - $f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
  - $f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$
- *Reflexiva*
  - $f(n) = \Theta(f(n))$
  - $f(n) = O(f(n))$
  - $f(n) = \Omega(f(n))$
- *Simetria*
  - $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

Propiedades

- *Simetria transpuesta*
  - $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$
- *Relación orden*
  - $f(n) = O(g(n))$; $a \leq b$
  - $f(n) = \Theta(g(n))$; $a = b$
  - $f(n) = \Omega(g(n))$; $a \geq b$

Complejidad

Para fines de comparación y determinación de eficiencia, usualmente se considera O, para determinar la optimalidad se utiliza Θ

Cómo determinar la función $g(n)$ dado que se tiene $f(n)$?

Complejidad

Si la función f(n) tiene la forma:

$$f(n) = t_1 + t_2 + t_3 + ...t_k$$

Se debe identificar el t_j, que sea el mayor de todos los términos de la $f(n)$.

Éste término es el candidato a ser $g(n)$.

Complejidad

Example

Si $f(n) = \frac{9n}{10} + 13$, el término más pesado es: $\frac{9n}{10}$.
Éste término es el más pesado por la presencia de n, por lo que $\frac{9}{10}$ es un factor constante, por lo que al quitar este coeficiente el termino sigue siendo el más pesado - entonces n es nuestro candidato. Por lo tanto, $g(n) = n$

Complejidad

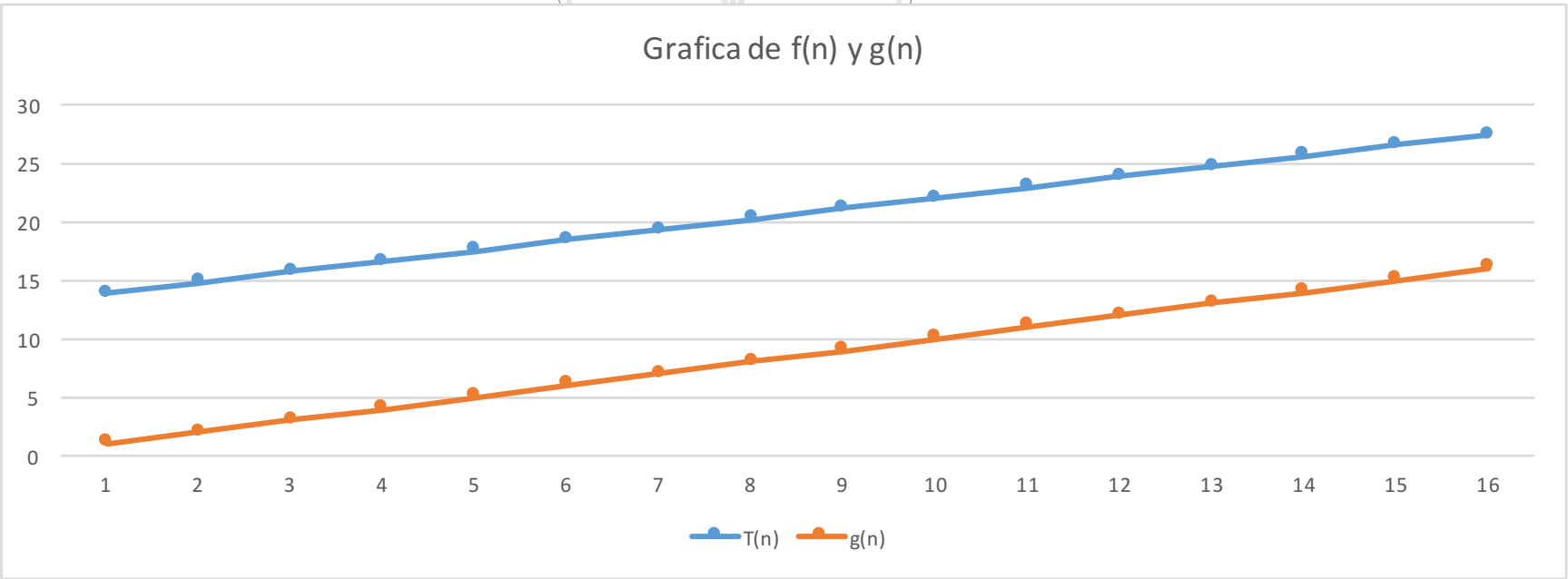
Por el concepto de complejidad: $f(n) = \frac{9n}{10} + 13$, es acotado maximalmente por $g(n) = n$.

Veamos gráficamente las funciones $f(n)$ y $g(n)$

Complejidad

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T(n)	13,9	14,8	15,7	16,6	17,5	18,4	19,3	20,2	21,1	22	22,9	23,8	24,7	25,6	26,5	27,4
g(n)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

$T(n) = f(n) = 9n/10 + 13$ $g(n) = n$



Complejidad

Retomemos: $f(n) = \frac{9n}{10} + 13$, es acotado maximalmente por $g(n) = n$.

Cómo es esto posible?, claramente la función $f(n)$ está más arriba que $g(n)$

Recordemos la definición de O:

$$f(n) = O(g(n)) \simeq f(n) \leq g(n)$$

Que se lee:

g(n) acota maximalmente a f(n)

Complejidad

$$f(n) = O(g(n)) \simeq f(n) \leq g(n)$$

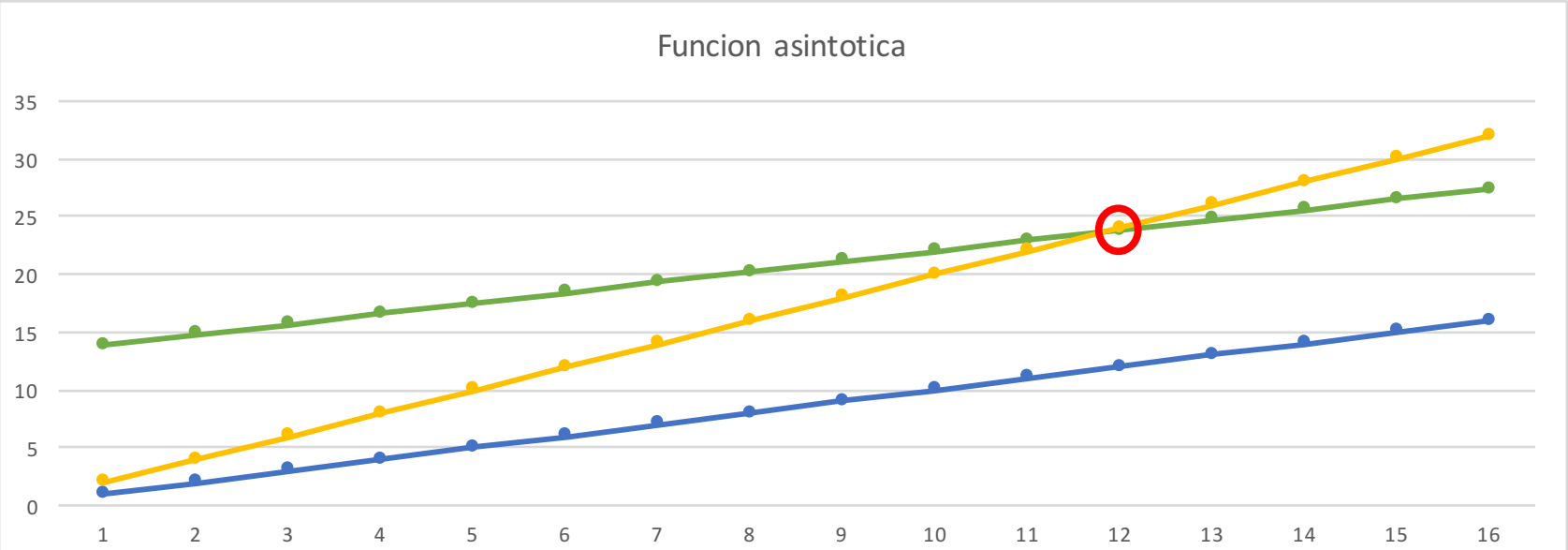
Para que esta relación se cumpla, se debe conseguir un n_0 a partir del cual la función $g(n)$ acota maximalmente a $f(n)$. Y a su vez es posible desplazar la función $g(n)$ en un valor constante c. Entonces se tiene que:

$$f(n) = O(g(n))$$
ssi: $\exists c, n_0$,tal que
$\forall n \geq n_0$, se cumple
$$f(n) \leq c * g(n)$$

Complejidad

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T(n)	13,9	14,8	15,7	16,6	17,5	18,4	19,3	20,2	21,1	22	22,9	23,8	24,7	25,6	26,5	27,4
g(n)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
c*g(n)	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

T(n) = f(n) = 9n/10 + 13 **g(n) = n** **c*g(n) = 2 * n** **n0 = 12**



Complejidad

Por lo tanto, se ha demostrado que:

$$f(n) = O(g(n))$$

para valores

$$c = 2$$
$$n_0 = 12$$

Complejidad

Consideremos ahora el $T_{capicua}(x)$ y definamos la función que acota maximalmente a este tiempo de ejecución, para determinar la complejidad de la solución.

$$T_{capicua}(x) = 14 + 16 * log_{10}\ x$$

El termino que define la función es

$$16 * log_{10}\ x,$$

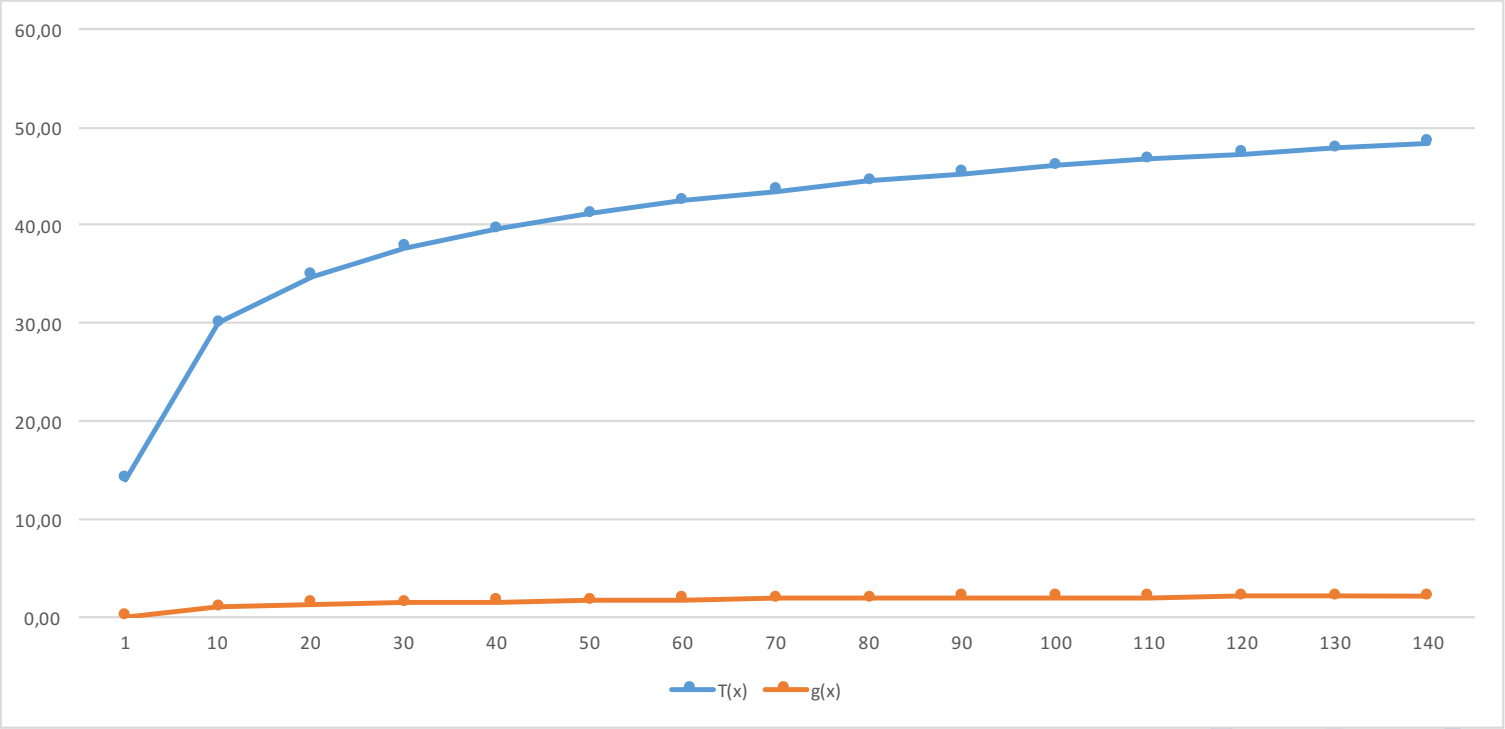
por lo que la función candidata a acotar es:

$$g(x) = log_{10}\ x$$

Complejidad

x	1	10	20	30	40	50	60	70	80	90	100	110	120	130	140
T(x)	14,00	30,00	34,82	37,63	39,63	41,18	42,45	43,52	44,45	45,27	46,00	46,66	47,27	47,82	48,34
g(x)	0,00	1,00	1,30	1,48	1,60	1,70	1,78	1,85	1,90	1,95	2,00	2,04	2,08	2,11	2,15

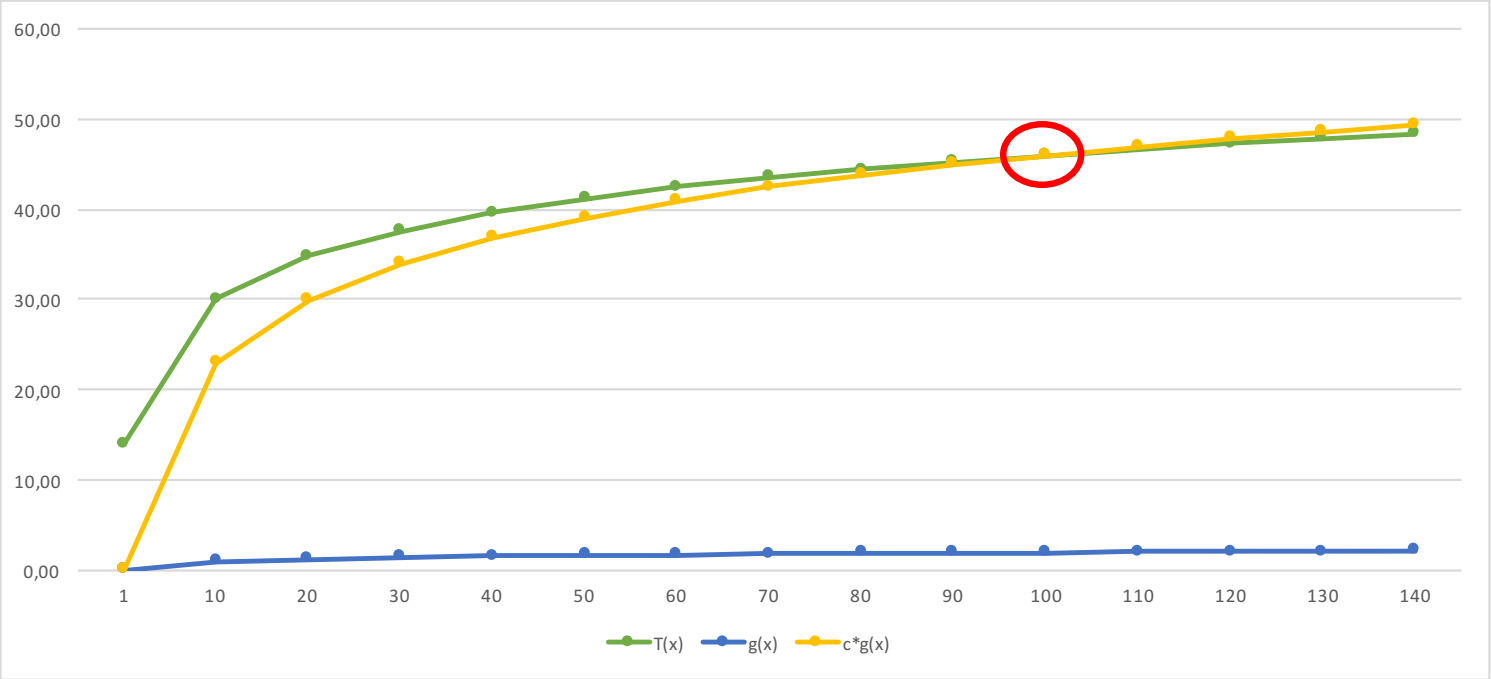
T(x) = f(x) = 14 + 16* log x **g(x) = log x**



Complejidad

x	1	10	20	30	40	50	60	70	80	90	100	110	120	130	140
T(x)	14,00	30,00	34,82	37,63	39,63	41,18	42,45	43,52	44,45	45,27	46,00	46,66	47,27	47,82	48,34
g(x)	0,00	1,00	1,30	1,48	1,60	1,70	1,78	1,85	1,90	1,95	2,00	2,04	2,08	2,11	2,15
c*g(x)	0,00	23,00	29,92	33,97	36,85	39,08	40,90	42,44	43,77	44,95	46,00	46,95	47,82	48,62	49,36

T(x) = f(x) = 14 + 16* log x g(x) = log x c*g(x) = 23*log x n0 = 100



Complejidad

Por lo tanto, se ha demostrado que:

$$14 + 16 * log\ x\ = O(log\ n)$$

para valores

$$c = 23$$
$$n_0 = 100$$

Funciones

La eficiencia se representa con funciones, entonces entenderlas es importante; algunas propiedades son:

- *Monotonicidad*
  - $f(n)$ es monotonicamente incremental ssi $m \leq n \Rightarrow f(m) \leq f(n)$
  - $f(n)$ es monotonicamente decremental ssi $m \geq n \Rightarrow f(m) \geq f(n)$
  - $f(n)$ es estrictamente incremental ssi $m < n \Rightarrow f(m) < f(n)$
  - $f(n)$ es estrictamente decremental ssi $m > n \Rightarrow f(m) > f(n)$

Funciones

- *Piso/Techo*

$$x - 1 \leq \lfloor x \rfloor \leq x \leq \lceil x \rceil \leq x + 1$$
$$\forall n \in \mathbb{Z}, \lfloor n/2 \rfloor + \lceil n/2 \rceil = n$$
$$\forall n \in \mathbb{R}_0^+, \wedge a, b \in \mathbb{Z}^+:$$
$$\begin{array}{lcl} \lceil \lceil n/a \rceil / b \rceil & = & \lceil n/ab \rceil \\ \lfloor \lfloor n/a \rfloor / b \rfloor & = & \lfloor n/ab \rfloor \\ \lceil a/b \rceil & \leq & (a + (b-1))/b \\ \lfloor a/b \rfloor & \geq & (a - (b-1))/b \end{array}$$

Funciones

- *Polinomial*

  Dado un polinomio de grado d, con $d \in \mathbb{Z}^+$, p(n):
  $p(n) = \sum_{i=0}^{d} a_i * n^i$, a_i es coeficiente
  $\Rightarrow p(n) = \Theta(n^d) \Rightarrow O(n^d)$

- *Exponencial*

  $\forall a \in \mathbb{R}^+$

$$\begin{aligned} a^0 &= 1 \\ a^1 &= a \\ a^{-1} &= 1/a \\ (a^m)^n &= a^{mn} \\ (a^m)^n &= (a^n)^m \\ a^m * a^n &= a^{m+n} \end{aligned}$$

  $\forall n \wedge a \geq 1 \Rightarrow a^n$ es monotonicamente incremental en n

Funciones

- *Logaritmos*

$\forall a \in \mathbb{R}, b > 0, c > 0$ y con las bases > 1

$$\begin{aligned}
a &= b^{log_b a} \\
log_c(ab) &= log_c a + log_c b \\
log_b a^n &= n * log_b a \\
log_b a &= \frac{log_c a}{log_c b} \\
log_b \frac{1}{a} &= -log_b a \\
a^{log_b c} &= c^{log_b a}
\end{aligned}$$

$\frac{x}{1+x} \leq ln(1+x) \leq x$

$\forall b > 1 \wedge n > 0 \Rightarrow log_b n$ es estrictamente incremental en n

Complejidad

Indica si $f(n) = O(g)$ o $f = \Omega(g)$ o si son ambos:

$f(n)$	$g(n)$
$n - 100$	$n - 200$
$n^{1/2}$	$n^{2/3}$
$100n + log\ n$	$n + (log\ n)^2$
$nlog\ n$	$10nlog10n$
$log\ 2n$	$log\ 3n$
$10log\ n$	$log(n^2)$
$n^{1,01}$	$nlog^2n$
$n^2/log\ n$	$n(log\ n)^2$
$n^{0,1}$	$(log\ n)^{10}$

Referencias

THOMAS H. CORMEN CHARLES E. LEISERSON RONALD L. RIVEST CLIFFORD STEIN (2009), *INTRODUCTION TO ALGORITHMS THIRD EDITION*, *The MIT Press*