Analizar el algoritmo

Calcular su eficiencia o complejidad algorítmica

Complejidad temporal o eficiencia en tiempo: Número de operaciones o pasos ejecutados para resolver un problema con una entrada de tamaño n. Se denota por $T(n)$.

Complejidad espacial o eficiencia en espaico: Qué tantos recursos de memoria utiliza el algoritmo para resolver un problema de tamaño n. Se denota por $S(n)$.

Análisis a Priori:

Búsqueda Secuencial.

Entrada: $A[0, \ldots, n-1]$, elemento a buscar x.

Salida: Si x se encuentra en A
regresar la posición
en caso contrario
regresa -1.

proceso:

```
for i <- 0 to i <- n-1
    if A[i] == x
        return i
return -1
```

Mejor Caso: x se encuentra en la primera posición, luego $T(n) = c \in \Omega(1)$

Peor Caso:

regresa -1.

proceso:

$\underbrace{i=0,1,\dots,n-1}_{n}$ n $(n+1)$

proceso	costos	#pasos ejecutados
① for $i \leftarrow 0$ to $i \leftarrow n-1$	c_1	$n+1$
② if $A[i] == x$	c_2	n
③ return i	c_3	0
④ return -1	c_4	1

... en la primera

Peor Caso: x no se encuentra en el arreglo, luego

$$T(n) = c_1(n+1) + c_2 n + c_3 \cdot 0 + c_4 \cdot 1$$
$$= c_1 n + c_1 + c_2 n + c_4$$
$$= \underbrace{(c_1 + c_2)}_{a} n + \underbrace{(c_1 + c_4)}_{b}$$
$$= an + b$$

$\therefore T(n) \in \Theta(n)$

miércoles, 25 de agosto de 2021 14:09

Suma Secuencial arreglo de enteros.

Entrada: $A[0, \dots, n-1]$

Salida: Suma de los valores de A.

Proceso:

```
suma = 0
for i = 0 to i ≤ n-1
    suma = suma + A[i]

return suma
```

Costos es el tiempo de ejecución

Proceso:

			ejecutados
1	suma = 0	C_1	1
2	for i = 0 to i ≤ n-1	C_2	n+1
3	suma = suma + A[i]	C_3	n
4	return suma	C_4	1

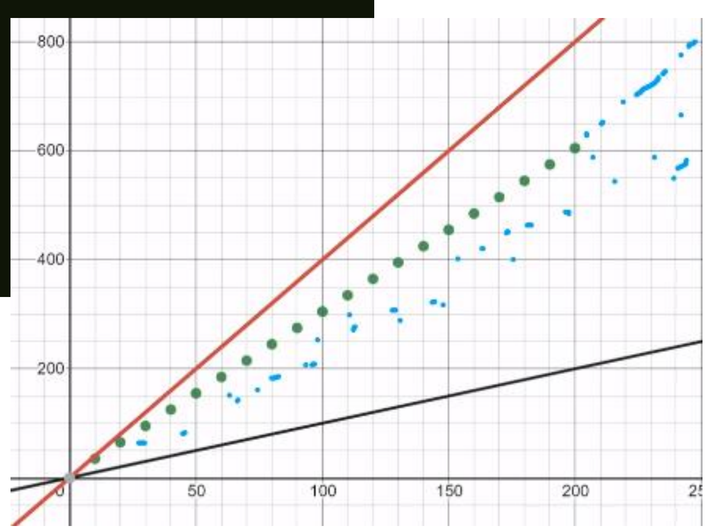
Mejor caso = Peor caso.

Luego $T(n) = C_1 + C_2(n+1) + C_3 n + C_4$

$$= \underbrace{(C_2 + C_3)}_{a} n + \underbrace{(C_1 + C_2 + C_4)}_{b}$$

$$= an + b$$

$$\therefore T(n) \in \Theta(n)$$



viernes, 27 de agosto de 2021 14:29

Análisis a Priori (Teórico): Cálculo de la complejidad algorítmica mediante conceptos teóricos. Se obtiene una función que acota el tiempo de ejecución del algoritmo.

Análisis a Posteri (experimental): Se recogen estadísticas de tiempo consumidas por el algoritmo mientras se ejecuta.

TAM=n=8

-1	0	1	2	3	4	5	6	7
	7	6	0	1	9	4	2	5
	j	i						

$-1 \leq j \leq i-1$ $\quad$ $1 \leq i \leq n-1$

$Key = A[i] = 6$ $\qquad$ $A[j] > Key$

Insertion Sort

Entrada: $A[0,\dots,n-1]$

Salida: El arreglo ordenado (ascendente).

Proceso:

		costo	# pasos ejecutados
①	for $i=1$ to $n-1$	C_1	n
②	$Key = A[i]$	C_2	$n-1$
③	$j = i-1$	C_3	$n-1$
④	while $j \geq 0$ and $A[j] > Key$	C_4	$\sum_{i=1}^{n-1} t_i$
⑤	$A[j+1] = A[j]$	C_5	$\sum_{i=1}^{n-1} (t_i - 1)$
⑥	$j = j-1$	C_6	$\sum_{i=1}^{n-1} (t_i - 1)$
⑦	$A[j+1] = Key$	C_7	$n-1$

Mejor caso:

i	t_i
1	1
2	1
3	1
⋮	⋮
i	1

– obs – Si i es la i-ésima línea del pseudo-código, asumimos que C_i es la cantidad constante requerida para ejecutarse.

$i=1 \longrightarrow t_1$

$i=2 \longrightarrow t_2$

$i=3 \longrightarrow t_3$

$\vdots$

$i=n-1 \longrightarrow t_{n-1}$

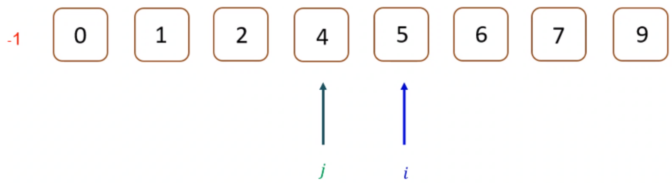
$\sum_{i=1}^{n-1} t_i$

$i = n-1 \longrightarrow t_n$

– obs: – Para $i=1,\dots,n-1$, sea t_i el número de veces que el ciclo while es ejecutado para cada valor de i.

Mejor Caso

Arreglo Ordenado Ascendentemente



$i \to t_i = 1$

$i = 4$

$t_4 = 1$

$Key = A[i] = 5$ $\quad$ $while\ j \geq 0\ and\ A[j] > Key\ do$

Se tiene:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=1}^{n-1} t_i + c_5 \sum_{i=1}^{n-1}(t_i - 1)$$
$$+ c_6 \sum_{i=1}^{n-1}(t_i - 1) + c_7(n-1).$$

-obs - El tiempo de ejecución depende del tamaño de la entrada y de como esten ordenado los datos.

Mejor caso: Ocurre cuando el arreglo se encuentra ordenado (ascendente), en tal caso, para cada $i = 1, \ldots, n-1$, se tiene $A[i] \leq key$, luego $t_i = 1 \quad \forall\, i = 1, \ldots, n-1$.

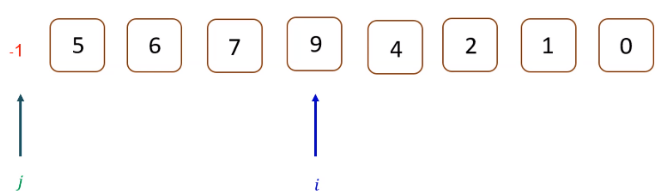
Luego:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=1}^{n-1} 1 + c_7(n-1)$$
$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
$$= (\underbrace{c_1 + c_2 + c_3 + c_4 + c_7}_{a})n + (\underbrace{-c_2 - c_3 - c_4 - c_7}_{b})$$
$$= an + b$$
$$\therefore\ T(n) \in \Omega(n)$$

Peor caso

i	t_i
1	2
2	3
$\vdots$	
i	$i+1$

Peor Caso



$i \to t_i = i + 1$

$i = 3$

$t_3 = 4$

$Key = A[i] = 5$ $\quad$ $while\ j \geq 0\ and\ A[j] > Key\ do$

Peor caso: Ocurre cuando el arreglo se encuentra ordenado descendentemente. En tal caso, se tiene que $t_i = i+1 \ \forall \ i=1,\dots,n-1$.

F. Gauss
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Luego:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{i=1}^{n-1}(i+1) + c_5\sum_{i=1}^{n-1} i + c_6\sum_{i=1}^{n-1} i + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left[\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1\right] + c_5\frac{(n-1)n}{2} + c_6\frac{(n-1)(n)}{2} + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left[\frac{(n-1)n}{2} + n-1\right] + c_5\frac{n(n-1)}{2} + c_6\frac{n(n-1)}{2} + c_7(n-1)$$

$$\vdots$$

$$= an^2 + bn + c$$ con $a = (c_4+c_5+c_6)/2$, $b = c_1+c_2+c_3+c_4/2+c_7-c_6/2-c_5/2$, $c = -c_2-c_3-c_4-c_7$

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

Insertion-Sort

Mejor caso: $T(n) \in \Omega(n)$

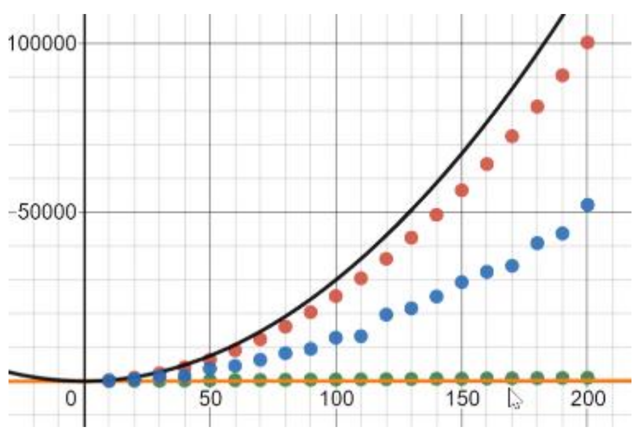
Peor caso: $T(n) \in \Theta(n^2)$

```
void insertionSort(int *A, int n, int *ct)
{
    int i; (*ct)++;
    int j; (*ct)++;
    int key; (*ct)++;

    (*ct)++;
    for(i=1; i<n; i++)
      {
        (*ct)++;
        key=A[i]; (*ct)++;
        j=i-1; (*ct)++;

        while(j>=0 && A[j]>key)
            {
             (*ct)++;
             (*ct)++;
             (*ct)++;
             A[j+1]=A[j]; (*ct)++;
             j--; (*ct)++;
            }
        (*ct)++;
        (*ct)++;


        A[j+1]=key; (*ct)++;
      }
}
```



producto.c

```c
#include <stdio.h>
int main(void)
{
  int m, n;
  scanf("%d", &n);
  m = n * n;
  printf("%d\n", m);
  return 0;
}
```

Línea	costo	# pasos ejecutados
int m, n;	C_1	1
scanf("%d", &n);	C_2	1
m = n * n;	C_3	1
printf("%d\n", m);	C_4	1
return 0;	C_5	1

Mejor Caso = Peor Caso

Luego $T(n) = C_1 + C_2 + C_3 + C_4 + C_5$

$\therefore \quad T(n) \in \Theta(1)$

suma.c

```c
#include <stdio.h>
int main(void)
{
  int m, n, i;
  scanf("%d", &n);
  m = 0;
  for (i=0; i<n; i++)
    m = m + n;
  printf("%d\n", m);
  return 0;
}
```

$i=0, i=1, \ldots, i=n-1 \quad i=n$

Línea	costo	# pasos ejecutados
int m, n, i;	C_1	1
scanf("%d", &n);	C_2	1
m = 0;	C_3	1
for (i=0; i<n; i++)	C_4	$n+1$
m = m + n;	C_5	n
printf("%d\n", m);	C_6	1
return 0;	C_7	1

Mejor Caso = Peor Caso

Luego $T(n) = C_1 + C_2 + C_3 + C_4(n+1) + C_5 n + C_6 + C_7$

$= \underbrace{(C_4 + C_5)}_{a} n + \underbrace{(C_1 + C_2 + C_3 + C_4 + C_6 + C_7)}_{b}$

$= an + b$

$\therefore \quad T(n) \in \Theta(n)$

incremento.c

```c
#include <stdio.h>
int main(void)
{
  int m, n, i, j;
  scanf("%d", &n);
  m = 0;
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      m++;
  printf("%d\n", m);
  return 0;
}
```

Línea	costo	# pasos ejecutados
int m, n, i, j;	C_1	1
scanf("%d", &n);	C_2	1
m = 0;	C_3	1
for (i=0; i<n; i++)	C_4	$n+1$
for (j=0; j<n; j++)	C_5	$\sum_{i=0}^{n-1} t_i$
m++;	C_6	$\sum_{i=0}^{n-1} (t_i - 1)$
printf("%d\n", m);	C_7	1
return 0;	C_8	1

Luego

$$T(n) = c_1 + c_2 + c_3 + c_4(n+1) + c_5 \sum_{i=0}^{n-1} t_i + c_6 \sum_{i=0}^{n-1} (t_i - 1) + c_7 + c_8$$

Mejor Caso = Peor Caso.

Se tiene que $t_i = n+1 \quad \forall\, i = 0, \dots, n-1$

Luego

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4(n+1) + c_5 \sum_{i=0}^{n-1} (n+1) + c_6 \sum_{i=0}^{n-1} n + c_7 + c_8 \\ &= c_1 + c_2 + c_3 + c_4(n+1) + c_5 \left[\sum_{i=0}^{n-1} n + \sum_{i=0}^{n-1} 1 \right] + c_6 n^2 + c_7 + c_8 \\ &= c_1 + c_2 + c_3 + c_4(n+1) + c_5 \left[n^2 + n \right] + c_6 n^2 + c_7 + c_8 \\ &= \underbrace{(c_5 + c_6)}_{a} n^2 + \underbrace{(c_4 + c_5)}_{b} n + (c_1 + c_2 + c_3 \\ &= an^2 + bn + c \end{aligned}$$

$$\therefore \; T(n) \in \Theta(n^2) \;\; /\!/$$

Mayor (m, n)	costo	#pasos ejecutados
if $(m \geq n)$	c_1	1
return m	c_2	1
	c_3	0
else return n	c_4	0

Supongamos que $m \geq n$.

Se tiene que $T(m) = c_1 + c_2 \in \Theta(1)$

Sin pérdida de generalidad supongamos que el mínimo se encuentra en la primera posición, es decir min = A [0].

	Costos	# pasos ejecutados
min = A[0]	C1	1
for i= 0 to i ≤ n-1	C2	n+1
if(A[i] < min)	C4	n
min = A[i]	C3	0
return min	C5	1

miércoles, 1 de septiembre de 2021 14:11

Fundamentos para el análisis de la eficiencia de algoritmos.

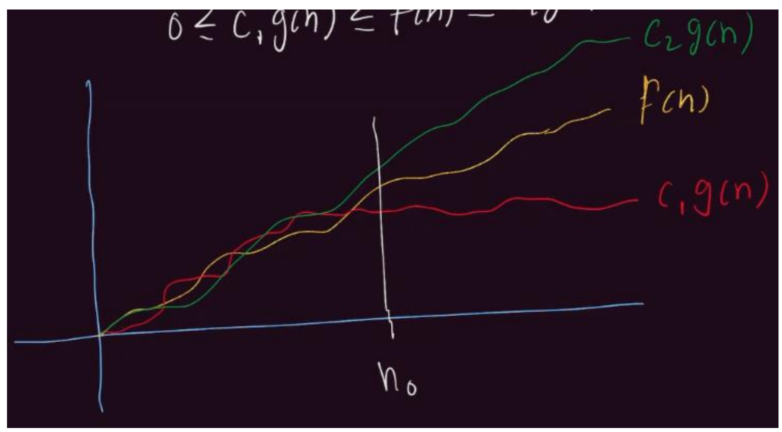
obs La notación que se usará para describir la complejidad algorítmica estará definida en términos de funciones cuyo dominio es $\mathbb{N}\cup\{0\}$, aunque comunmente se mostrará extendido a $\mathbb{R}$.

$$f(n) \in \Theta(g(n))$$
$$f(n) = \Theta(g(n))$$

Notación Θ.

Def: Dada una función $g(n)$. $\Theta(g(n))$ denota el conjunto de funciones definidas como:

$$\Theta(g(n)) = \{f(n) : \exists_n\ c_1, c_2 > 0 \text{ y } n_0 > 0 \text{ tal que } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0\}$$



Decimos que $g(n)$ es un ajuste asintótico para $f(n)$

obs Si $f(n) \in \Theta(g(n))$, entonces $f(n) \geq 0$ $\forall n \in \mathbb{N}$, tales funciones son llamadas asintóticamente no negativas o asintóticamente positivas.

Ejemplo:

(1) $f(n) = \frac{1}{2}n^2 - \frac{1}{2}n \in \Theta(n^2)$

Sol: En efecto, se tiene:

$$\frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \quad \forall\ n \geq 1 \quad \text{— (1)}$$

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq c_1 n^2$$

* Encontrar un valor $c > 0$ tal que

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{c}n^2 > 0$$ .....*

Si: $\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{c}n^2$ && $\frac{1}{2}n^2 - \frac{1}{c}n^2 > 0$

Si: $-\frac{1}{2}n \geq -\frac{1}{c}n^2$ && $\frac{c-2}{2c}n^2 > 0$

Si: $\frac{1}{2}n \leq \frac{1}{c}n^2$ && $(c-2)n^2 > 0$

Sabemos que n^2 >0 entonces

Si: $cn \leq 2n^2$ && $c - 2 > 0$

Si: $c \leq 2n$ && $c > 2$

para $n=1$ $c \leq 2$ && $c > 2$ $\times$

Para $n=2$ $c \leq 2(2) = 4$ && $c > 2$ $\boxed{C = 4}$

2 < C <= 2 no se cumple

2 < C <=4 si se cumple

Sustituyendo C=4 en (*)

Se tiene que

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{4}n^2 \quad \forall\ n \geq 2$$

$$= \frac{1}{4}n^2 \quad \forall\ n \geq 2 \quad \text{— (2)}$$

de (1) y (2) se tiene:

$$\underset{c_1}{\frac{1}{4}}n^2 \leq \frac{1}{2}n^2 - \frac{1}{2}n \leq \underset{c_2}{\frac{1}{2}}n^2 \quad \forall\ n \geq \underset{n_0}{2}$$

i.e tomando $c_1 = 1/4$, $c_2 = 1/2$ y $n_0 = 2$,

Se tiene:

$$c_1 n^2 \leq \frac{1}{2}n^2 - \frac{1}{2}n \leq c_2 n^2 \quad \forall\ n \geq n_0$$

$$\therefore \quad \frac{1}{2}n^2 - \frac{1}{2}n \in \Theta(n^2) \quad //$$

② $6n^3 \notin \Theta(n^2)$

Sol: Supongamos que $6n^3 \in \Theta(n^2)$

$\Rightarrow \exists_n\ c_1, c_2 > 0$ y $n_0 > 0$ m $0 \le c_1 n^2 \le 6n^3 \le c_2 n^2 \quad \forall n \ge n_0$

$\Rightarrow c_1 \le 6n \le c_2$

$\Rightarrow \cancel{\lim_{n\to\infty} c_1} \le \cancel{\lim_{n\to\infty} 6n} \le \cancel{\lim_{n\to\infty} c_2}$ #c

$\therefore 6n^3 \notin \Theta(n^2)$ //

Colocamos n^2 para factorizarlo puesto que

Necesitamos acotarlo por arriba

Clase 03 Sep

viernes, 3 de septiembre de 2021 14:42

Ejemplo: $f(n) = \frac{1}{2}n^2 - 3n \in \Theta(n^2)$ g(n)

[$\exists_n\ c_1, c_2 > 0$ y $n_0 > 0$ m $0 \le c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2 \ \forall n \ge n_0$]

Sol: Se tiene;

$$\frac{1}{2}n^2 - 3n \le \frac{1}{2}n^2 + 3n \quad \forall n \ge 1$$
$$\le \frac{1}{2}n^2 + 3n^2 \quad \forall n \ge 1$$
$$= \frac{7}{2}n^2 \quad \forall n \ge 1$$

c_1 n_0

por otro lado

$$\frac{1}{2}n^2 - 3n \ge \frac{1}{4}n^2$$

Si: $\frac{1}{2}n^2 - \frac{1}{4}n^2 - 3n \ge 0$

Si: $\frac{1}{4}n^2 - 3n \ge 0$

Si: $n\left(\frac{1}{4}n - 3\right) \ge 0$

Si: $\frac{1}{4}n - 3 \ge 0$

Si: $n \ge 12$ //, luego $\frac{1}{2}n^2 - 3n \ge \frac{1}{4}n^2 \quad \forall n \ge 12$

c_1 n_0

Se tiene:

$$\frac{1}{4}n^2 \le \frac{1}{2}n^2 - 3n \le \frac{7}{2}n^2 \quad \forall n \ge 12$$

c_1 c_2 n_0

$\therefore \frac{1}{2}n^2 - 3n \in \Theta(n^2)$

obs En general. Si $P(n)=a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n^1 + a_0$ es un polinomio de grado d $(d>0)$, entonces $P(n) \in \Theta(n^d)$.

obs Si $P(n) = K$ con K constante, entonces $P(n) \in \Theta(n^0) = \Theta(1)$.

Big o

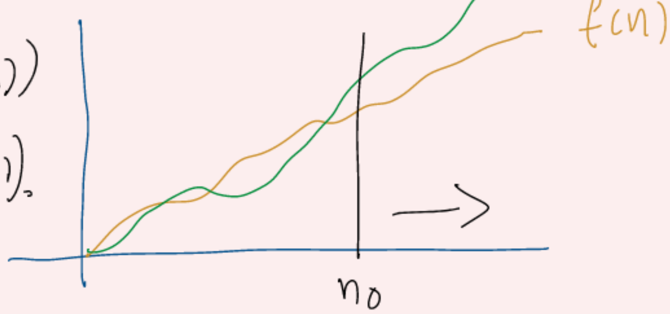
Notación O.

Def: Dada una función $g(n)$. $O(g(n))$ denota el conjunto de funciones definida como:

$$O(g(n)) = \{ f(n) : \exists n \; c>0 \;\&\; n_0>0 \text{ constantes } \ni \; 0 \le f(n) \le c\,g(n) \;\; \forall n \ge n_0 \}.$$

$f(n) \in O(g(n))$

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



Ejemplo: $f(n) = 100n + 5 \in \Theta(n^2)$

Sol: Se tiene:

$$f(n) = 100n + 5 \le 100n + 5n \quad \forall n \ge 1$$
$$= 105n \quad \forall n \ge 1$$
$$\le \underbrace{105}_{c} n^2 \quad \forall n \ge \underbrace{1}_{n_0}$$

$\therefore \quad 100n + 5 \le c n^2 \quad \forall n \ge n_0$

con $c = 105$ & $n_0 = 1$

$\therefore \quad 100n + 5 \in \Theta(n^2)$ //

$\Rightarrow f(n) \in \Theta(n)$

$f(n) \in \Theta(n)$

$f(n) \in \Theta(n^2)$

$f(n) \in \Theta(n \log n)$

$f(n) \in \Theta(n^3)$

Siempre nos quedamos con la de menor grado

El alumno que suba a la plataforma teams lo sube a plataforma

obs Se tiene que $\Theta(g(n)) \subseteq \Theta(g(n))$

para cada $g(n)$.

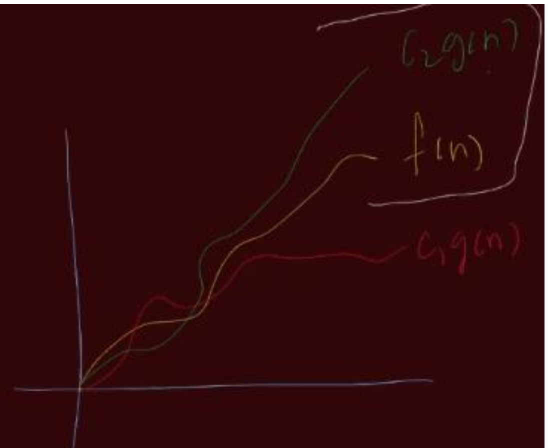
En efecto, ya que, sea $f(n) \in \Theta(g(n))$

$\Rightarrow \exists\, c_1, c_2 > 0$ y $n_0 > 0$ ∋

$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \quad \forall n \ge n_0$

$\Rightarrow \quad 0 \le f(n) \le c_2 g(n) \quad \forall n \ge n_0$

$\therefore \quad f(n) \in \Theta(g(n))$ //



obs Si $p(n)=a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n^1 + a_0$,

entonces $p(n) = \Theta(n^d)$.

obs Si $p(n) = k$ con k una constante,

entonces $p(n) = \Theta(1)$.

Notación Ω.

Def: Dada una función $g(n)$. $\Omega(g(n))$ denota el conjunto definido como:

$$\Omega(g(n)) = \{ f(n) : \exists\, c, n_0 > 0 \ni 0 \leq c\, g(n) \leq f(n) \quad \forall n \geq n_0 \}$$

$f(n) \in \Omega(g(n))$

$f(n) = \Omega(g(n))$.

obs $g(n)$ es una cota inferior asintótica para $f(n)$.

Teorema: Dada dos funciones $f(n)$ y $g(n)$,
$f(n) \in \Theta(g(n))$ sii $f(n) \in O(g(n))$ y $f(n) \in \Omega(g(n))$.

Sol:
$f(n) \in \Theta(g(n)) \Rightarrow \exists_n\ c_0, c_1 > 0$ y $n_0 \geq 0$
/ $0 \leq c_0 g(n) \leq f(n) \leq c_1 g(n) \quad \forall n \geq n_0$

sii $f(n) \in O(g(n))$ y $f(n) \in \Omega(g(n))$

Teorema *

Si $h_1(n) \in O(g_1(n))$ y $h_2(n) \in O(g_2(n))$, entonces
$h_1(n) + h_2(n) \in O(\max\{g_1(n), g_2(n)\})$.

Dem: Sea $h_1(n) \in O(g_1(n))$ y $h_2(n) \in O(g_2(n))$

$\Rightarrow \exists_n\ c_1 > 0$ y $n_1 > 0$ / $0 \leq h_1(n) \leq c_1 g_1(n) \ \forall n \geq n_1$

y $\exists_n\ c_2 > 0$ y $n_2 \geq 0$ / $0 \leq h_2(n) \leq c_2 g_2(n) \ \forall n \geq n_2$

$\forall n \geq n_0 = \max\{n_1, n_2\}$.

$\Rightarrow 0 \leq h_1(n) + h_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \quad \forall n \geq n_0$ con $c = \max\{c_1, c_2\}$

$\leq c\, g_1(n) + c\, g_2(n) \quad \forall n \geq n_0$

$= c\,[g_1(n) + g_2(n)] \quad \forall n \geq n_0$

$\leq c\,[\max\{g_1(n), g_2(n)\} + \max\{g_1(n), g_2(n)\}]$

$= 2c \max\{g_1(n), g_2(n)\} \quad \forall n \geq n_0$

c_0

$\therefore h_1(n) + h_2(n) \in O(\max\{g_1(n), g_2(n)\})$. ///

Notación o y ω.

Def Sean $f(n)$ y $g(n)$ dos funciones.
$f(n)$ es $o(g(n))$ si $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Def Sean $f(n)$ y $g(n)$ dos funciones.
$f(n)$ es $\omega(g(n))$ si $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

obs $5n^2+2n+1 \in \Theta(n^2)$, por otro lado

$$\lim_{n \to \infty} \frac{5n^2+2n+1}{n^2} = \lim_{n \to \infty} \left(5 + \frac{2}{n} + \frac{1}{n^2}\right) = 5 = c \neq 0$$

luego $5n^2+2n+1 \notin o(n^2)$.

Esto es, en general, si $f(n) \in \Theta(g(n))$ no significa que $f(n) \in o(g(n))$.

obs $\frac{n^2}{2} \in \Omega(n)$, por otro lado

$$\lim_{n \to \infty} \frac{\frac{n^2}{2}}{n} = \infty \Rightarrow \frac{n^2}{2} \in \omega(n)$$

pero

$$\lim_{n \to \infty} \frac{\frac{n^2}{2}}{n^2} = \frac{1}{2} = c$$

$$\lim_{n \to \infty} \frac{\frac{n^2}{2}}{n^2} = \frac{1}{2} = c$$

$\Rightarrow$ $\frac{n^2}{2} \notin \omega(n^2)$ pero $\frac{n^2}{2} \in \Omega(n^2)$

Es decir, si $f(n) \in \Omega(g(n))$, no significa que $f(n) \in \omega(g(n))$.

17.- Inducción matemática

$$T(n) = \begin{cases} 2 & \text{si } n = 2 \\ 2T(n/2) + n & \text{si } n = 2^k, \text{ para } k > 1 \end{cases}$$

$T(n) = n \log n$

Sol: Sea $n = 2^k$ $(k = \log n)$

tenemos

$$T(2^k) = \begin{cases} 2 & \text{si } k = 1 \\ 2T(2^{k-1}) + 2^k & \text{si } k > 1 \end{cases}$$

$$\left[T(2^k) = 2^k \log 2^k \right]$$

caso base:

$k = 1$

$T(2^1) = T(2) = 2 = 2 \log 2$ ✓

caso inductivo:

Supongamos que $\underline{T(2^{k-1}) = 2^{k-1} \log 2^{k-1}}$ H.I

se tiene

$$\begin{aligned} T(2^k) &= 2\underline{T(2^{k-1})} + 2^k \\ &= 2\, 2^{k-1} \log 2^{k-1} + 2^k \quad \text{por H.I} \\ &= 2^k \log\left(\frac{2^k}{2}\right) + 2^k \\ &= 2^k \left[\log 2^k - \log 2\right] + 2^k \\ &= 2^k \log 2^k - 2^k \cancel{\log 2} + 2^k \\ &= 2^k \log 2^k \end{aligned}$$

$\therefore T(n) = n \log n \therefore T(n) \in \Theta(n \log n)$