

Practica 8

- 1) Determinar para cada función $t(n)$ en la siguiente tabla, cual es el mayor tamaño n de una instancia de un problema que puede ser resuelto en cada uno de los tiempos indicados en las columnas de la tabla, suponiendo que el algoritmo para resolverlo utiliza $t(n)$ microsegundos.

$t(n)$	1 seg.	1 min.	1 hora	1 día	1 mes	1 año	1 siglo
$\log_2(n)$	2^{10^6}	$2^6 * 10^7$	$2^{36 * 10^8}$	$2^{864 * 10^8}$	$2^{25920 * 10^8}$	$2^{311040 * 10^8}$	$2^{31104000 * 10^8}$
\sqrt{n}	10^{12}	$(6 * 10^7)^2$	$(36 * 10^8)^2$	$(864 * 10^8)^2$	$(25920 * 10^8)^2$	$(311040 * 10^8)^2$	$(31104000 * 10^8)^2$
n	10^6	$6 * 10^7$	$36 * 10^8$	$864 * 10^8$	$25920 * 10^8$	$311040 * 10^8$	$31104000 * 10^8$
$n \times \log_2(n)$	6274 6	280141 7	1333800 00	27551000 00	718710000 00	7870896061 98	67699498 46 3641
n^2	$\sqrt{10^6}$	$\sqrt{6 * 10^7}$	$\sqrt{36 * 10^8}$	$\sqrt{864 * 10^8}$	$\sqrt{25920 * 10^8}$	$\sqrt{311040 * 10^8}$	$\sqrt{3110400 * 10^8}$
2^n	$\frac{\ln(10^6)}{\ln(2)}$	$\frac{\ln(6 * 10^7)}{\ln(2)}$	$\frac{\ln(36 * 10^8)}{\ln(2)}$	$\frac{\ln(864 * 10^8)}{\ln(2)}$	$\frac{\ln(25920 * 10^8)}{\ln(2)}$	$\frac{\ln(311040 * 10^8)}{\ln(2)}$	$\frac{\ln(31104000 * 10^8)}{\ln(2)}$
$n!$	9	11	12	13	15	16	17

- 2) Si el tiempo de ejecución en el mejor caso de un algoritmo, $t_m(n)$, es tal que $t_m(n) \in \Omega(f(n))$ y el tiempo de ejecución en el peor caso de un algoritmo, $t_p(n)$, es tal que $t_p(n) \in O(f(n))$, ¿Se puede afirmar que el tiempo de ejecución del algoritmo es $\Theta(f(n))$?

Si, se puede afirmar que el tiempo de ejecución del algoritmo es $\Theta(f(n))$, ya que esta acotado inferiormente en el mejor caso por $f(n)$ ($t_m(n) \in \Omega(f(n))$) y esta acotado superiormente en el peor caso por $f(n)$ ($t_p(n) \in O(f(n))$).

En el mejor caso el tiempo de ejecución va a ser más grande o igual que c_1 (constante positiva) $\cdot f(n)$ y en el peor caso va a ser menor o igual que c_2 (constante positiva) $\cdot f(n)$. Con ello, se cumple la definición de $\Theta(f(n))$ $\{t: N \rightarrow R^+ / \exists c_1, c_2 \in R^+, n_0 \in N \text{ tq } c_1 f(n) \leq t(n) \leq c_2 f(n), n \geq n_0\}$

- 3) Un algoritmo tarda 1 segundo en procesar 1000 items en una máquina determinada. ¿Cuánto tiempo tomara procesar 10000 items si se sabe que el tiempo de ejecución del algoritmo es n^2 ? ¿y si se sabe que es $n \times \log_2 n$? ¿Qué se estaría asumiendo en todos los casos?

$$1000^2 \rightarrow 1 \text{ segundo}$$

$$10000^2 \rightarrow 100 \text{ segundos (regla de 3 simples duh)}$$

$$1000 \cdot \log_2 1000 \rightarrow 1 \text{ segundo}$$

$$10000 \cdot \log_2 10000 \rightarrow 40/3 = 13,3333$$

En todos los casos se asume que es el peor.

- 4) Un algoritmo toma n^2 días y otro n^3 segundos para resolver una instancia de tamaño n de un problema. Mostrar que el segundo algoritmo superara en tiempo al primero solamente en instancias que requieran más de 20 millones de años para ser resueltas.

$$T1(n) = n^2 \text{ días}$$

$$T2(n) = n^3 \text{ segundos}$$

1. Convertimos a misma unidad de tiempo. Paso días a segundos

$$T1(n) = n^2 * 60 * 60 * 24 \text{ segundos} = n^2 * 86400 \text{ segundos}$$

2. $T1(n) / T2(n) = 86400 / n$

La relación entre los tiempos $T1$ y $T2$ depende inversamente de n . Esto significa que a medida que n aumenta, $T1$ se vuelve relativamente más pequeño en comparación con $T2$.

3. Ahora vamos a ver en qué valor de n ambos algoritmos tardan lo mismo:

$$- n^2 * 86400 = n^3 \rightarrow$$

$$n = 86400$$

$$- n^3 > n^2 * 86400 \rightarrow$$

$$n > 86400$$

Es decir, para cualquier valor de n mayor a 86400 el segundo algoritmo va a tardar más que el primero.

4. Ambos algoritmos para 86400 van a tardar un total de 86400^3 segundos que son aproximadamente 20438383,1 años. De esta manera, queda demostrado que para instancias que requieran aproximadamente mas de 20 millones de años para ser resueltas el segundo algoritmo superara en tiempo al primero.

- 5) ¿Cuáles y cuantas serían las operaciones elementales necesarias para multiplicar dos enteros n y m por medio del algoritmo enseñado en la escuela primaria? ¿Esta cantidad depende de la entrada? Justifique.

Las operaciones elementales en este algoritmo son la multiplicación, la suma y las multiplicaciones por 10. Depende de n y m

```
sum = 0
base_i = 1
```

```

base_j = 1
operand = [3, 3, 3]
multiplier = [1, 2]

for i in operand[::-1]:
    base_j = 1
    for j in multiplier[::-1]:
        sum += j * base_j * i * base_i
        base_j *= 10
    base_i *= 10

print(sum)

```

(no voy a intentar mas con esto es un pire dios los bendiga de hecho robe casi todo el punto)

- 6) Dar el tiempo de ejecución en función de n de los siguientes algoritmos y una $f(n)$ tal que el tiempo de ejecución pertenezca a $\Theta(f(n))$. Determine si cada algoritmo o partes de este tiene casos de análisis (peor, mejor, etc.).

a)

```

p ← 0
for i ← 1 to n do
    for j ← 1 to n2 do
        for k ← 1 to n3 do
            p ← p + 1

```

b)

```

p ← 0
for i ← 1 to n do
    for j ← 1 to i do
        for k ← 1 to n do
            p ← p + 1

```

- a) La asignación de 0 a p es una constante 1 y la suma de p son dos constantes (suma y asignación). El for de más afuera hace n iteraciones, el que le sigue n^2 y el siguiente n^3 . Como no hay mejor y peor n , no existe mejor ni peor caso, por lo que $t(n) = 1 + 2n^6$ que $\in \Theta(n^6)$.

$$\begin{aligned}
 1 + \sum_{i=1}^n \sum_{j=1}^{n^2} \sum_{k=1}^{n^3} 2 &= \\
 1 + \sum_{i=1}^n \sum_{j=1}^{n^2} n^3 * 2 &= \\
 1 + \sum_{i=1}^n n^2 * n^3 * 2 &= \\
 1 + n * n^2 * n^3 * 2 &=
 \end{aligned}$$

$$1 + 2n^6$$

- b) La asignación de 0 a p es una constante 1 y la suma de p son dos constantes (suma y asignación). El for de mas afuera hace n iteraciones. El siguiente hace i iteraciones y el for de más adentro hace n iteraciones, por lo que $t(n) = n^3 + n^2 + 1$ que $\in \Theta(n^3)$. En este caso tampoco hay mejor o peor valor de n ni de i.

$$1 + \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^n 2 =$$

$$1 + \sum_{i=1}^n \sum_{j=1}^i n * 2$$

$$1 + \sum_{i=1}^n i * n * 2$$

$$1 + 2n \sum_{i=1}^n i$$

$$1 + 2n * \left(\frac{n(n+1)}{2} \right)$$

$$1 + n^3 + n^2$$