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	$6 * 10^7$	$36 * 10^8$	$864 * 10^8$	$25920 * 10^8$	$311040 * 10^8$	$31104000 * 10^8$
$n \times \log_2(n)$	62746	2801417	133380000	2755100000	71871000000	787089606198	6769949846 3641
n^2	$\sqrt{10^6}$	$\sqrt{6 * 10^7}$	$\sqrt{36 * 10^8}$	$\sqrt{864 * 10^8}$	$\sqrt{25920 * 10^8}$	$\sqrt{311040 * 10^8}$	$\sqrt{31104000 * 10^8}$
2^n	$\frac{\ln(10^6)}{\ln(2)}$	$\frac{\ln(6 * 10^7)}{\ln(2)}$	$\frac{\ln(36 * 10^8)}{\ln(2)}$	$\frac{\ln(864 * 10^{88})}{\ln(2)}$	$\frac{\ln(25920 * 10^8)}{\ln(2)}$	$\frac{\ln(311040 * 10^8)}{\ln(2)}$	$\frac{\ln(31104000 * 10^8)}{\ln(2)}$
n!	9	11	12	13	15	16	17

2) Si el tiempo de ejecución en el mejor caso de un algoritmo, tm(n), es tal que tm(n) ∈ Ω(f(n)) y el tiempo de ejecución en el peor caso de un algoritmo, tp(n), es tal que tp(n) ∈ O(f(n)), ¿Se puede afirmar que el tiempo de ejecución del algoritmo es Θ(f(n))?

   Si, se puede afirmar que el tiempo de ejecución del algoritmo es Θ(f(n)), ya que esta acotado inferiormente en el mejor caso por f(n) (tm(n) ∈ Ω(f(n))) y esta acotado superiormente en el peor caso por f(n) (tp(n) ∈ O(f(n))).

   En el mejor caso el tiempo de ejecución va a ser más grande o igual que c1(constante positiva) * f(n) y en el peor caso va a ser menor o igual que c2 (constante positiva) * f(n). Con ello, se cumple la definición de Θ(f(n)) {t:N → R+ / ∃ c1, c2 ∈ R+, n0 ∈ N tq c1 f(n) ≤ t(n) ≤ c2 f(n), n ≥ n0}

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 ➔ 1 segundo

   10000^2 ➔ 100 segundos (regla de 3 simples duh)

   $1000^* \log_2 1000$ ➔ 1 segundo

   $10000^* \log_2 10000$ ➔ 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 días

   T2(n) = n^3 segundos

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

      T1(n) = n^2 * 60 * 60 * 24 segundos = n^2 * 86400 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 →

        n = 86400

      - n^3 > n^2 * 86400 →

        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)
```

(dios los bendiga 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 Θ(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 n^2 do
    for k ← 1 to n^3 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 ∈ Θ(n^6).

$$1 + \sum_{i=1}^{n} \sum_{j=1}^{n^2} \sum_{k=1}^{n^3} 2 =$$

$$1 + \sum_{i=1}^{n} \sum_{j=1}^{n^2} n^3 * 2$$

$$1 + \sum_{i=1}^{n} n^2 * n^3 * 2$$

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

$$1 + 2n^6$$

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.

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

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

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

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

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

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

7) Definir y analizar el tiempo de ejecución de la multiplicación de una matriz triangular inferior por una matriz completa (en la que todos sus elementos pueden ser diferentes de 0). ¿Qué formas tiene de hallar t(n)? ¿De cuántas formas podría encontrar la pertenencia a O() o a Θ() del algoritmo?

   El tiempo de ejecución será igual que la multiplicación entre 2 matrices "normales" con t(n) $\in O(n^3)$. La única diferencia es que en este caso se pueden ahorrar algunas iteraciones (los elementos por encima de la diagonal principal son 0) si se tiene en cuenta la estructura.

   Se trata de un algoritmo donde no existe peor ni mejor caso (no hay mejor o pero valor de columnas ni de filas), por lo que el valor de Θ() será igual que el de O(), ya que el algoritmo tiene la misma complejidad en el mejor y peor de los casos.

8) ¿Encuentra algún inconveniente para analizar las iteraciones while y repeat como recurrencias?

   Si, debido a que el numero de iteraciones puede variar según la condición, el análisis con funciones y/o recurrencias es más complejo. Esta complejidad recae en encontrar que función es la que se reduce. El problema es el análisis de las iteraciones no uniformes y encontrar la función que representa la cantidad de iteraciones. Hay que entrar con el detalle fino de lo que hay que hacer dentro de una iteración no uniforme e incluso hay algunas iteraciones para las cuales no es posible ni si quiera definir la función ni identificar una recurrencia. Esta función, cabe aclarar, que se puede encontrar como una recurrencia o no.

9) Considerar las matrices A, B, C ∈ $IR^{(n \times n)}$, y la notación tal que Xi,j , con 1 ≤ i, j ≤ 2 y X cualquiera de las matrices A, B o C, identifica una de las cuatro submatrices de orden n/2.

a) Dar el orden del tiempo de ejecución del algoritmo D&C que se describe con las ecuaciones

$$
\begin{aligned}
C_{1,1} &= A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} \\
C_{1,2} &= A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\
C_{2,1} &= A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} \\
C_{2,2} &= A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2}
\end{aligned}
$$

¿Sería necesario definir algo más?

Sería necesario definir las llamadas recursivas de los casos en donde la matriz no es 2x2 y aclarar que se realiza cuando si son 2x2.

Utilizando la receta calculamos T(n)

T(0) = 1
T(n) = 8T(n/2) + f(n^2).

a = 8 b=2 f(n)=n^k k=2

a > b^k → a > 2^2 → a > 4

Θ($n^{\log b(a)}$) → Θ(n^3)

b) Buscar el algoritmo de Strassen, dar su definición en función de las submatrices Xi,j anteriores y dar su orden de tiempo de ejecución.

Video a partir del minuto 5:50 honestamente lo explica mejor de lo que yo jamás podría.

c) Comparar los dos algoritmos de multiplicación de matrices. ¿Los dos son algoritmos D&C? ¿Alguno de los dos es “mejor” que el otro en cuanto a tiempo de ejecución?

Ambos utilizan D&C, pero el de Strassen es “mejor” ya que su tiempo de ejecución es de Θ($n^{2.81}$)