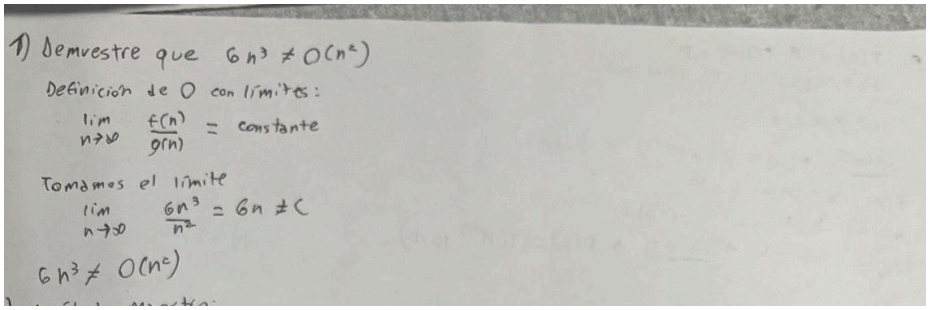
Ejercicio 1:

Demuestre que $6n^3 \neq \boldsymbol{O}(n^2)$.



Ejercicio 2:

¿Cómo sería un array de números (mínimo 10 elementos) para el mejor caso de la estrategia de ordenación Quicksort(n) ?

Un caso de array que podemos usar en Quicksort y que sea O(n log(n)), sería un caso en el cual el array se encuentre “balanceado”, es decir, cuando dividimos el problema en 2, comparando con el pivote, las dos partes tengan la misma cantidad de elementos.

[7,1,2,6,5,9,3,0,8,4]

Ejercicio 3:

Cuál es el tiempo de ejecución de la estrategia **Quicksort(A), Insertion-Sort(A)** y **Merge-Sort(A)** cuando todos los elementos del array A tienen el mismo valor?

InsertionSort: En este caso, vamos a tener que recorrer todo el arreglo, pero no realizar cambios, la complejidad será de O(n)

QuickSort: Al ser todos los elementos iguales al pivote, va a ser O(n^2)

MergeSort: Siempre ejecuta en O(n log(n))

Ejercicio 4:

Implementar un algoritmo que ordene una lista de elementos donde siempre el elemento del medio de la lista contiene antes que él en la lista la mitad de los elementos menores que él. Explique la estrategia de ordenación utilizada.

Mi algoritmo consta de:

Primero, ordenar la lista

Luego, intercambiar el tercer elemento con el de la mitad de la lista

Y finalmente, intercambiar el primero con el último

Ejemplo de lista de salida

7	3	2	8	5	4	1	6	10	9

Ejercicio 5:

Implementar un algoritmo **Contiene-Suma(A,n)** que recibe una lista de enteros A y un entero n y devuelve True si existen en A un par de elementos que sumados den n. Analice el costo computacional.

Ejercicio 6:

Investigar otro algoritmo de ordenamiento como BucketSort, HeapSort o RadixSort, brindando un ejemplo que explique su funcionamiento en un caso promedio. Mencionar su orden y explicar sus casos promedio, mejor y peor.

RadixSort:

Radix Sort es un algoritmo de ordenamiento no comparativo que ordena los elementos procesando sus dígitos uno a uno. Funciona bien para números enteros o strings. Su complejidad es

$O(d*(n+b))$, donde d es el número de dígitos, n es el número de elementos, y b es la base del sistema numérico utilizado.

Es eficiente para grandes cantidades de datos con claves de tamaño fijo, pero puede no ser óptimo para claves de tamaño variable o rangos de valores muy grandes.

Ejercicio 7:

A partir de las siguientes ecuaciones de recurrencia, encontrar la complejidad expresada en $\Theta(n)$ y ordenarlas de forma ascendente respecto a la velocidad de crecimiento. Asumiendo que T(n) es constante para n <= 2. Resolver 3 de ellas con el método maestro completo: $T(n) = a\ T(n/b) + f(n)$ y otros 3 con el método maestro simplificado: $T(n) = a\ T(n/b) + n^c$

a. $T(n) = 2T(n/2) + n^4$
b. $T(n) = 2T(7n/10) + n$
c. $T(n) = 16T(n/4) + n^2$
d. $T(n) = 7T(n/3) + n^2$
e. $T(n) = 7T(n/2) + n^2$
f. $T(n) = 2T(n/4) + \sqrt{n}$

7) Método maestro:

a) $T(n) = 2T(n/2) + n^4$

$a=2 \quad b=2 \quad c=4 \quad f(n) = n^4$

$n^{\log_2 2} = n$

Caso 3 con $\epsilon = 3$

Verifico si: $a \cdot f(n/b) \leq c \cdot f(n)$

$a\, f(n/b) = 2\left(\frac{n}{2}\right)^4 = \frac{n^4}{8} \leq \frac{1}{8} n^4 \qquad c = \frac{1}{8}$

Luego: $T(n) = \Theta(f(n)) = \Theta(n^4)$

b) $T(n) = 2T(7n/10) + n$

$a=2 \quad b=10/7 \quad c=1 \quad f(n)=n$

$n^{\log_{10/7} 2} \approx n^{1,44}$

Caso 1 $\quad \epsilon \times 0,44 \rightarrow T(n) \simeq \Theta(n^2)$

c) $T(n) = 16T(n/4) + n^2$

$a=16 \quad b=4 \quad c=2 \quad f(n) = n^2$

$n^{\log_4 16} = n^2$

Caso 2 $\rightarrow T(n) = \Theta(n^2 \log(n))$

d) $T(n) = 7T(n/3) + n^2$

$a=7 \quad b=3 \quad c=2 \quad f(n) = n^2$

$\log_3 7 \simeq 1,77 < 2$

Caso 3 $\quad T(n) = \Theta(n^2)$

e) $T(n) = 7T(n/2) + n^2$

$a=7 \quad b=2 \quad c=2 \quad f(n) = n^2$

$\log_2 7 \approx 2,8 > 2$

Caso 1 $\quad T(n) = \Theta(n^{\log_2 7}) \simeq \Theta(n^3)$

f) $T(n) = 2T(n/4) + n^{1/2}$

$a=2 \quad b=4 \quad c=1/2 \quad f(n) = n^{1/2}$

$\log_4 2 = 1/2 = c \rightarrow$ Caso 2 $\rightarrow T(n) = T(n^{1/2} \lg n)$

A tener en cuenta:

1. **Usen lápiz y papel primero**
2. ~~**No se puede utilizar otra Biblioteca mas alla de algo1.py y linkedlist.py**~~
3. **Hacer una análisis por cada algoritmo implementado del caso mejor, el caso peor y una perspectiva del caso promedio.**