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) ?

El mejor caso para el Quicksort (mínimo 10 elementos) es tener una lista desordenada y tomar el elemento del medio, esto último depende de la lista mas que nada otros tomar cualquier pivote sin hacer diferencia.

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?

Quicksort(A): T(n) = n^2

En el caso del Quicksort lo que pasaría es que no importa que pivote se tome ya que el algoritmo no dividira la lista en dos sino que pondra a todos los elemtos en una lista izquierda o derecha (esto depende de como uno plantee el algoritmo con valores repetidos, ya sea <= o >=).

Insertion-Sort(A): T(n) = n

En el Insertion-Sort la lista solo se recorrera una vez debido a que nunca encontrara a un elemento que sea menor al de su izquierda.

Merge-Sort(A): T(n) = n(log(n))

Aquí no importa que elementos contenga la lista ya que el Merge-Sort dividira la lista a la mitad recursivamente y al final compara elementos iguales y los pondrá en la lista resultante en “orden” (esto depende también de como se implemente el algoritmo).

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.

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.

```python
#Ejercicio 5
def Contiene_Suma(A,n):
    if A is None:
        return
    else:
        for i in range(len(A)):
            for j in range(i, len(A)):
                if A[i] + A[j] == n:
                    return True
    return

list = [2,6,8,3,1,0]
number = 7
output = Contiene_Suma(list,number)
if output is True:
    print("Existe el par")
else:
    print("No existe un par")
```

El costo computacional es de T(n) = n^2

Analizar casos de complejidad:

Peor Caso: O(n^2)

Este sería el peor caso ya que accederiamos a los dos bucles for para poder calcular la suma de los pares y que sea igual al entero pasado por parámetro. El primer for es para tomar un pivote de la lista y el segundo for es para poder sumar este “pivote” con los restantes elementos y vefificar si la suma es igual a n.

Caso Promedio: Θ(n)

El caso promedio del código pasaría solo cuando se accede una vez al primer for y más de una vez el segundo for, si la suma del valor del primer for por un valor de los restantes es igual a n. Ejemplo:

Lista =[1,2,3,0] y n=4

Accedera una vez al primer for tomando el 1 y luego recorrerá dos veces el segundo for para tomar el 3 y ahí se cumpliría que la suma es igual a n=4.

Mejor Caso: $\Omega(1)$

El mejor caso del código anterior sería que la suma del par de enteros sean los elementos de las dos primeras posiciones ya que si esto sucedo solo se estaría accediendo una vez a ambos bucles.

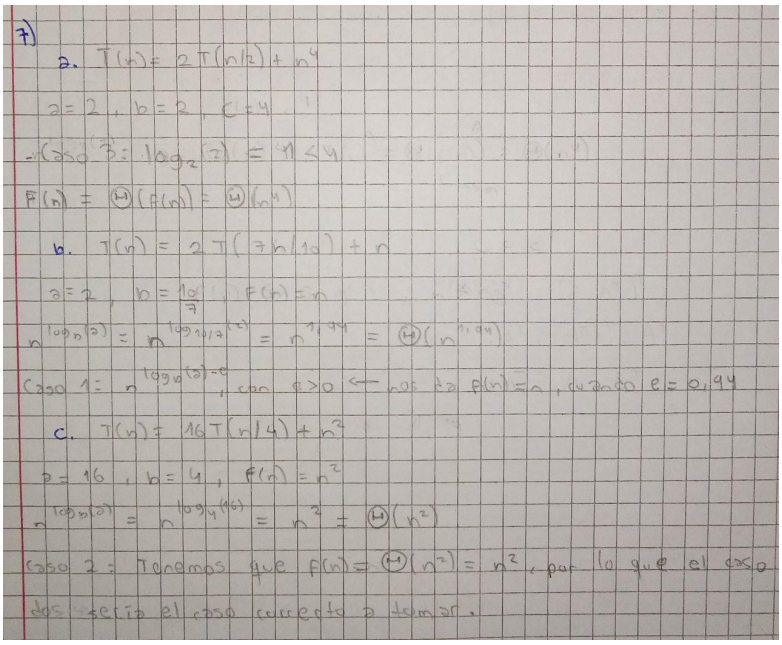
Ejercicio 6:

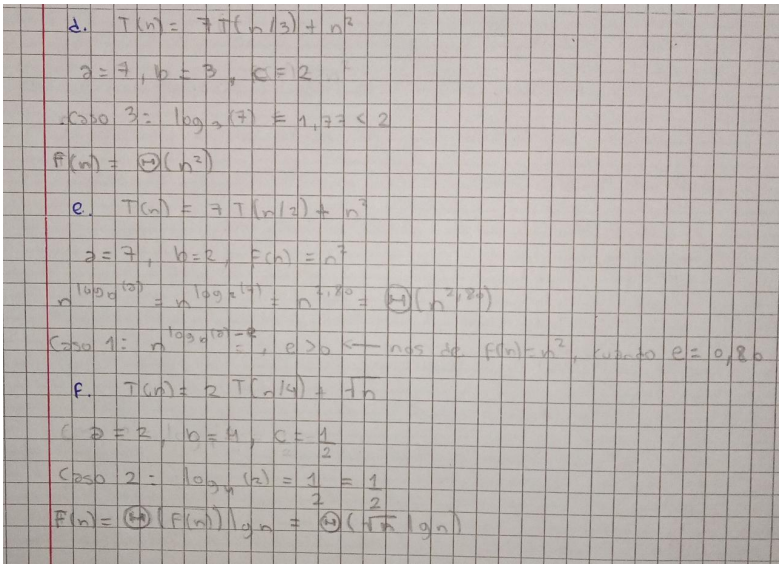
Investigar otro algoritmo de ordenamiento como BucketSort, HeapSort o RadixSort, brindando un ejemplo que explique su funcionamiento en un caso promedio. Mencionar su orden y explicar sus casos promedio, mejor y peor.

Ejercicio 7:

A partir de las siguientes ecuaciones de recurrencia, encontrar la complejidad expresada en Θ(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. $T(n) = 2T(n/4) + \sqrt{n}$





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.**