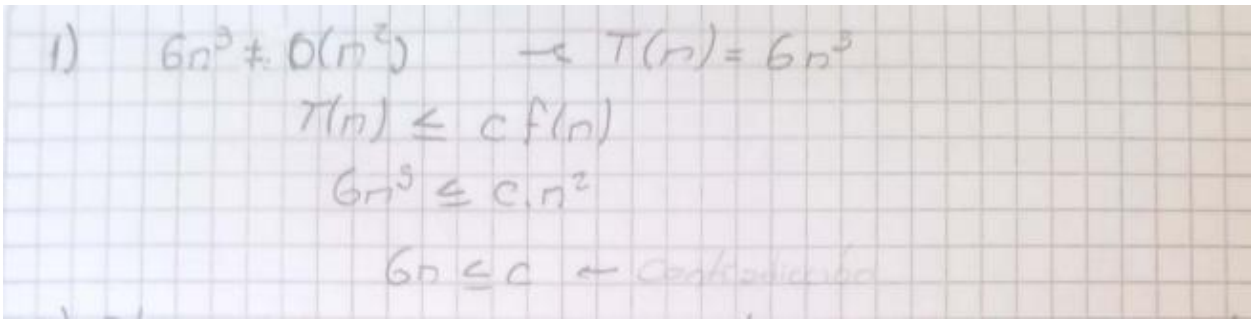
Ejercicio 1:

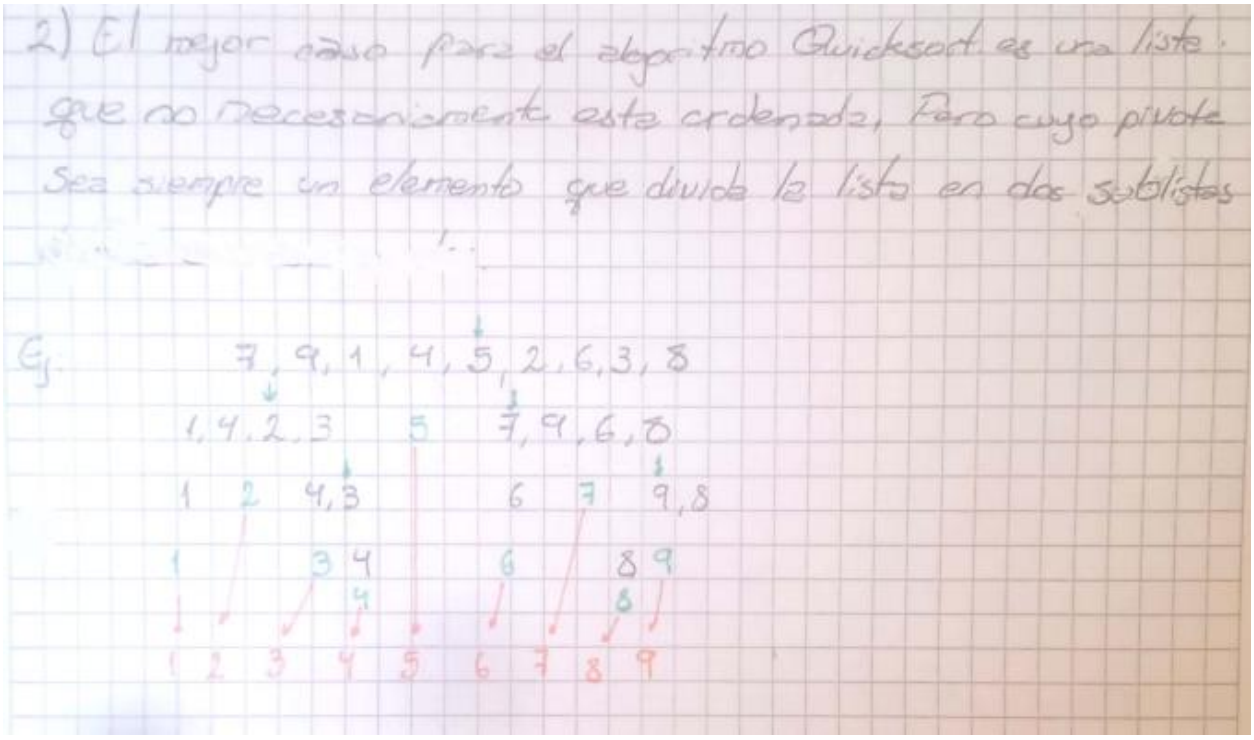
Demuestre que $6n^3 \neq \boldsymbol{O}(n^2)$.



contradiccion

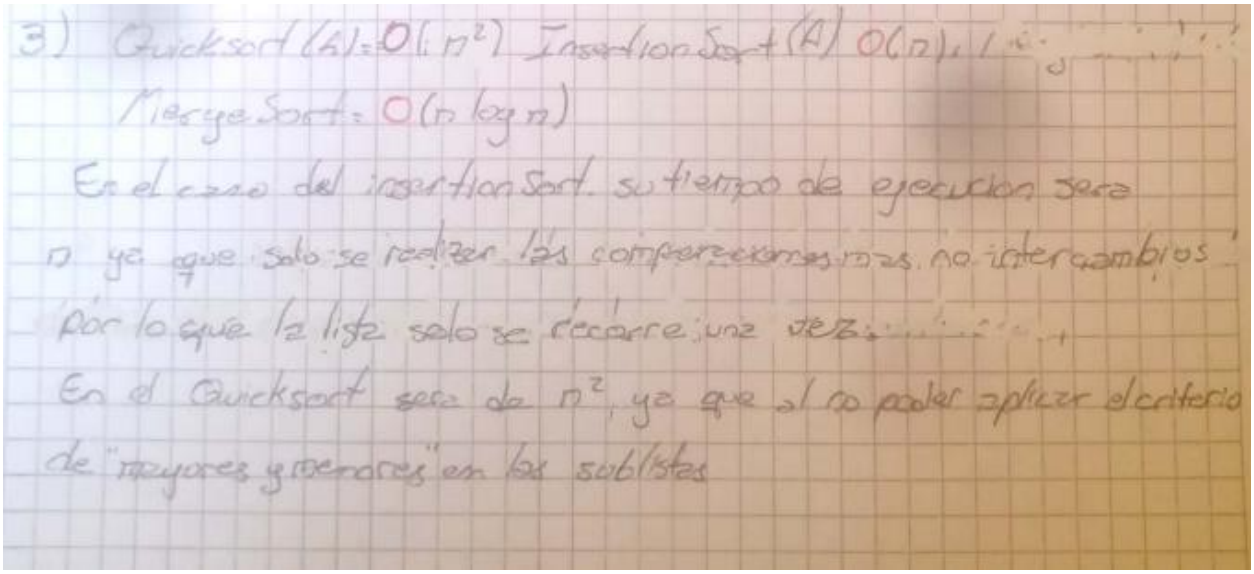
Ejercicio 2:

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



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?



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

```python
# Ejercicio 4
def ordenar_list(list):
    medio = len(list)//2
    aux = list[medio-1]
    izq = list[:medio-1]
    der = list[medio:]
    #print(izq)
    #print(der)
    #print(aux)
    rta = []

    while izq != [] and der != []:

        if izq != []:
            rta.append(izq.pop())
        if der != []:
            rta.append(der.pop())

    rta.insert(medio -1 , aux)
    #print(rta)
    return rta

```

Ejercicio 5:

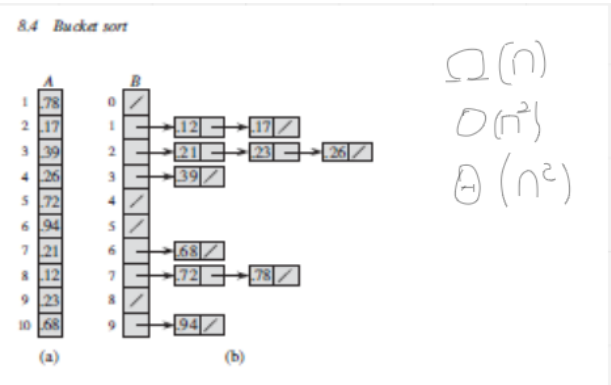
Implementar un algoritmo **Contiene-Suma(A,n)** que recibe una lista de enteros A y un entero n y devuelve True si existen en A un par de elementos que sumados den n. Analice el costo computacional.

```python
# Ejercicio 5

#Al contener la funcion sort, la complejidad se eleva a O(n log n) en el peor de los casos
# Tita(n log n)
# Omega(n)

def contiene_suma(A, n):
    A.sort() # O(n log n)
    left = 0
    right = len(A) - 1
    while left < right:
        suma = A[left] + A[right]
        if suma == n:
            return True
        elif suma < n:
            left += 1
        else:
            right -= 1
    return False
```

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.

Bucket Sort:



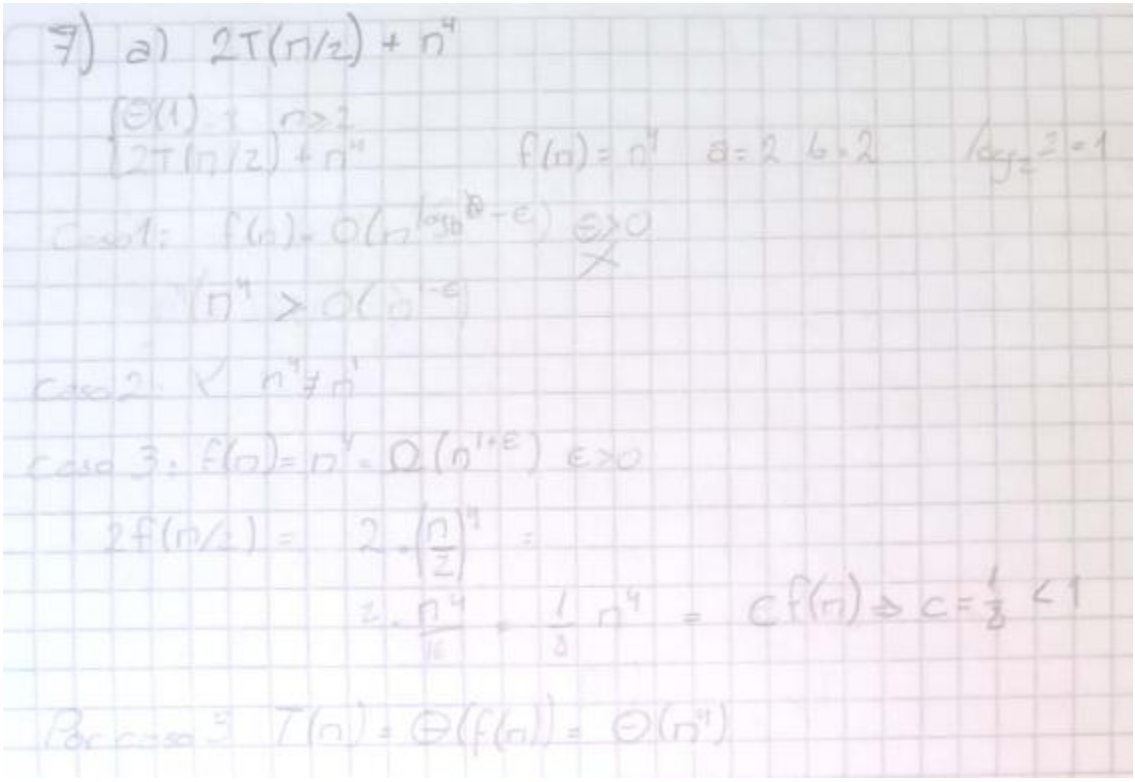
El algoritmo bucket sort basa su funcionalidad en separar los elementos de la lista en distintas cubetas cuyo criterio de seleccion se lo da el usuario, por ejemplo en subgrupos del 1-10, 11-20,

etc. Como se ve en la imagen, y aplicar en cada subgrupo otro algoritmo de ordenamiento elegido como insertion sort, merge sort, etc. Si se utiliza el algoritmo insertion sort para ordenar las sublistas, el orden de complejidad queda como en la imagen siendo O(n^2) el peor caso, pero este puede variar ya que se puede utilizar un algoritmo distinto en cada cubeta

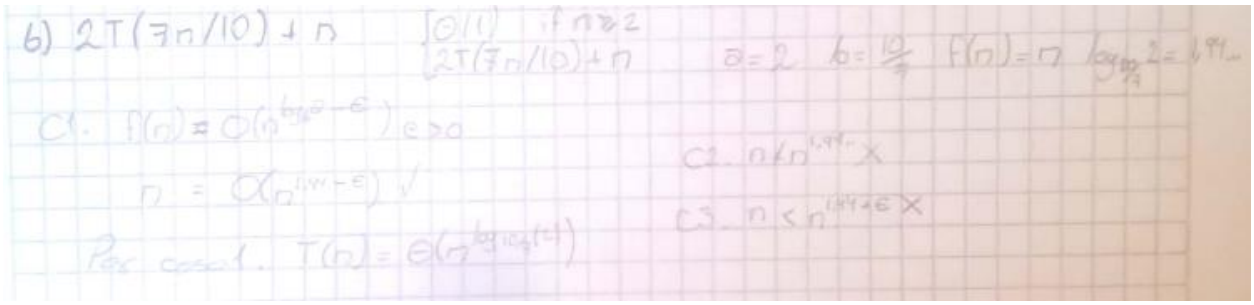
Ejercicio 7:

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

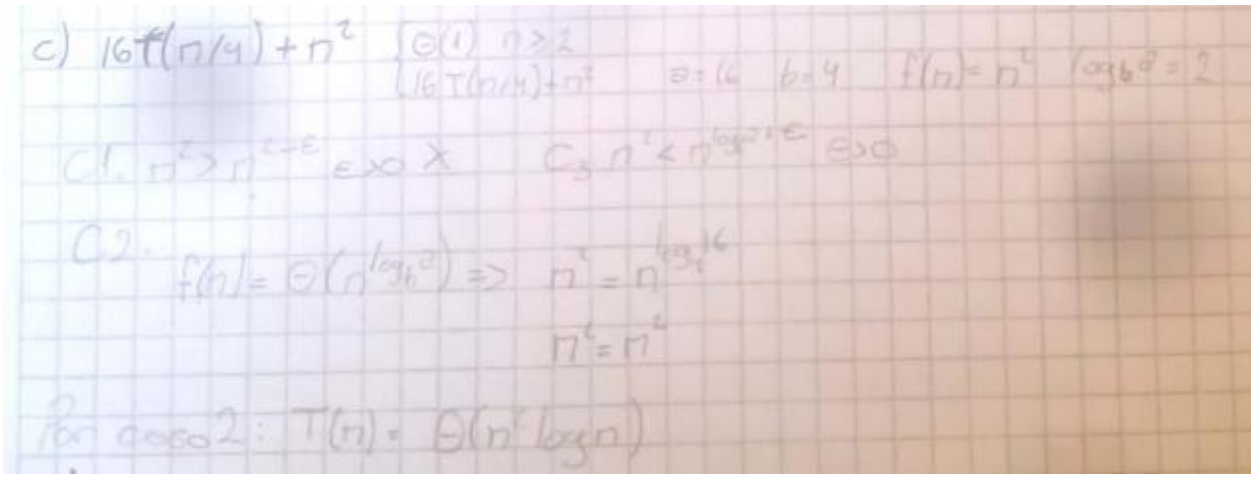
a. T(n) = 2T(n/2) + n^4



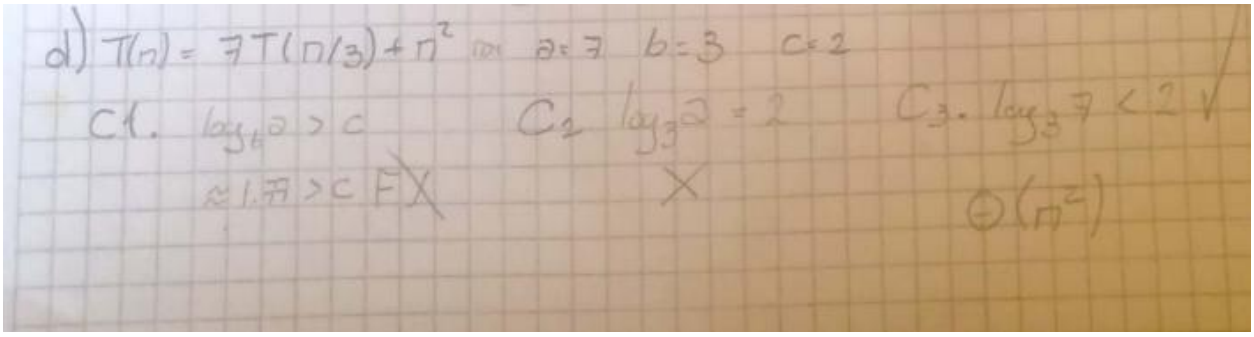
b. T(n) = 2T(7n/10) + n



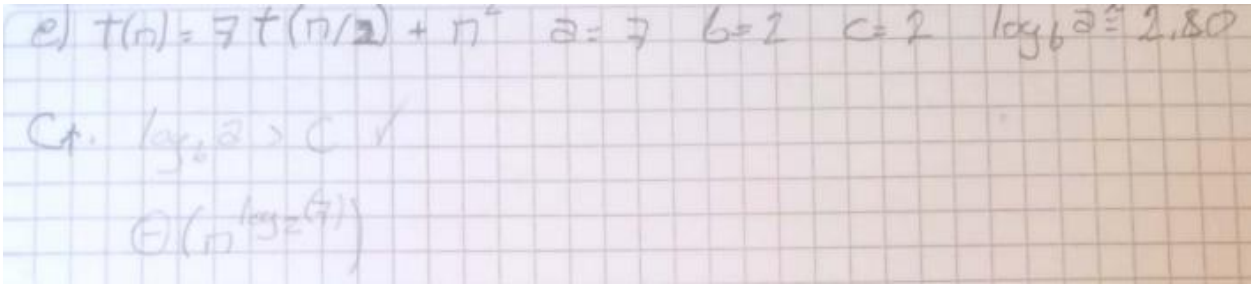
c. $T(n) = 16T(n/4) + n^2$



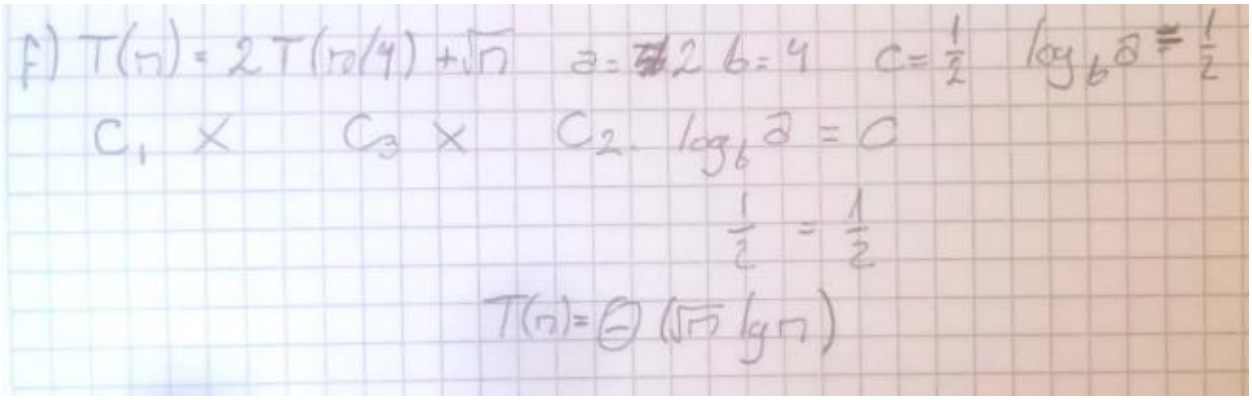
d. $T(n) = 7T(n/3) + n^2$



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.**