

### Ejercicio 1:

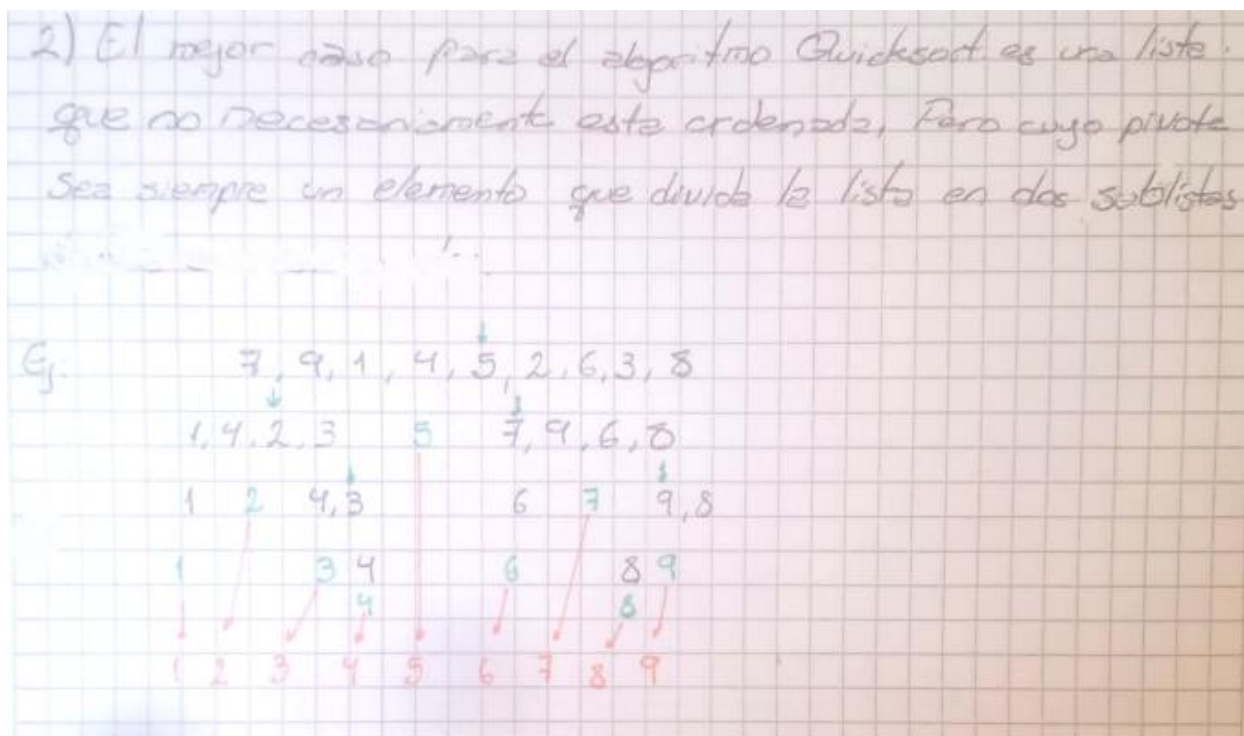
Demuestre que  $6n^3 \neq O(n^2)$ .

1)  $6n^3 \neq O(n^2) \leftarrow T(n) = 6n^3$   
 $T(n) \leq c f(n)$   
 $6n^3 \leq c \cdot n^2$   
 $6n \leq c \leftarrow \text{Contradicción}$

contradiccion

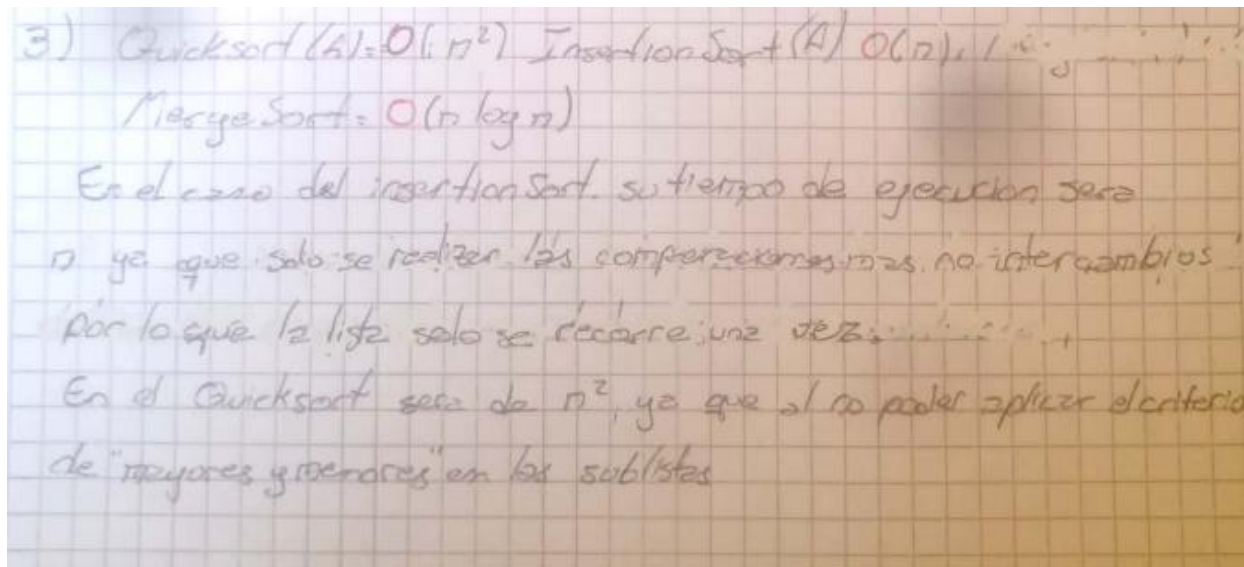
### Ejercicio 2:

¿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
---	---	---	---	---	---	---	---	----	---

```
1  # Ejercicio 4
2  def ordenar_list(list):
3      medio = len(list)//2
4      aux = list[medio-1]
5      izq = list[:medio-1]
6      der = list[medio:]
7      #print(izq)
8      #print(der)
9      #print(aux)
10     rta = []
11
12     while izq != [] and der != []:
13
14         if izq != []:
15             rta.append(izq.pop())
16         if der != []:
17             rta.append(der.pop())
18
19     rta.insert(medio - 1 , aux)
20     #print(rta)
21     return rta
22
```

### 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.

```
52 # Ejercicio 5
53
54 #Al contener la funcion sort, la complejidad se eleva a O(n log n) en el peor de los casos
55 # Tita(n log n)
56 # Omega(n)
57
58 def contiene_suma(A, n):
59     A.sort() # O(n log n)
60     left = 0
61     right = len(A) - 1
62     while left < right:
63         suma = A[left] + A[right]
64         if suma == n:
65             return True
66         elif suma < n:
67             left += 1
68         else:
69             right -= 1
70     return False
```

### 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  $n \leq 2$ . 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.  $T(n) = 7T(n/3) + n^2$

Handwritten solution for recurrence relation d. The recurrence is  $T(n) = 7T(n/3) + n^2$  with  $a=7$ ,  $b=3$ , and  $c=2$ . Three cases are checked:  $C_1: \log_3 7 > 2$  is marked as false (FX) with a note  $\approx 1.77 > 2$ ;  $C_2: \log_3 7 = 2$  is marked as false (X);  $C_3: \log_3 7 < 2$  is marked as true (✓) and leads to the complexity  $\Theta(n^2)$ .

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

Handwritten solution for recurrence relation e. The recurrence is  $T(n) = 7T(n/2) + n^2$  with  $a=7$ ,  $b=2$ , and  $c=2$ .  $\log_2 7 \approx 2.80$ . Case  $C_1: \log_2 7 > 2$  is marked as true (✓) and leads to the complexity  $\Theta(n^{\log_2 7})$ .

f.  $T(n) = 2T(n/4) + \sqrt{n}$

Handwritten solution for recurrence relation f. The recurrence is  $T(n) = 2T(n/4) + \sqrt{n}$  with  $a=2$ ,  $b=4$ , and  $c=1/2$ .  $\log_4 2 = 1/2$ . Cases  $C_1$  and  $C_3$  are marked as false (X). Case  $C_2: \log_4 2 = 1/2$  is marked as true (✓) and leads to the complexity  $T(n) = \Theta(\sqrt{n} \lg n)$ .

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.