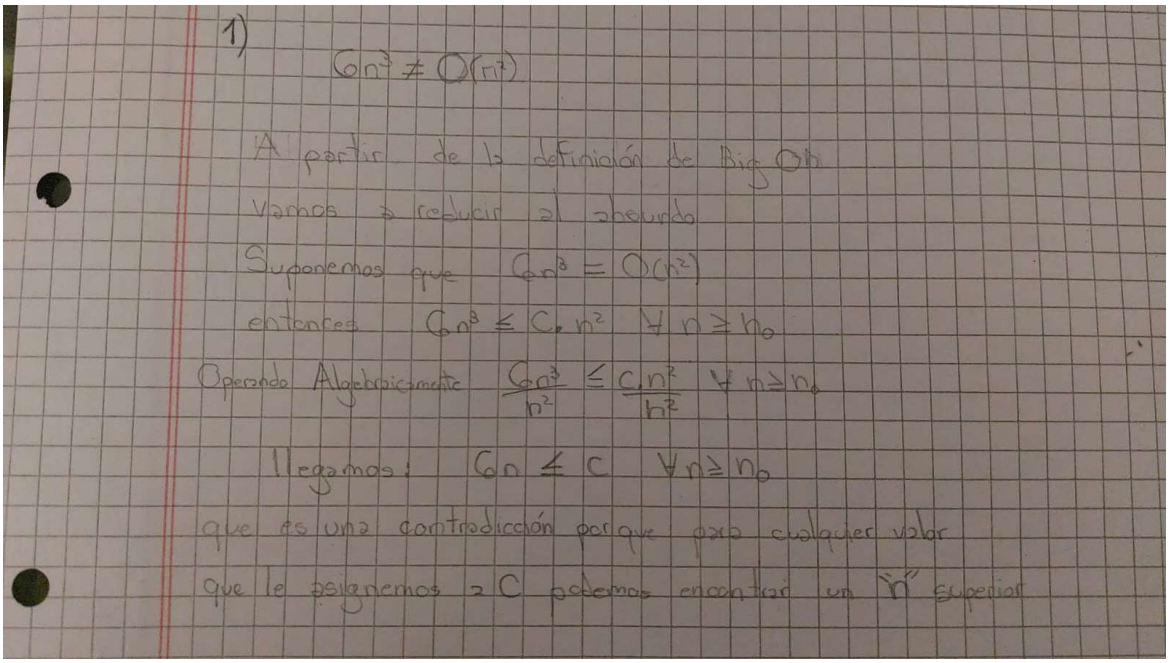
1)



1) $Cn^3 \neq O(n^2)$

A partir de la definición de Big Oh vamos a reducir al absurdo

Suponemos que $Cn^3 = O(n^2)$

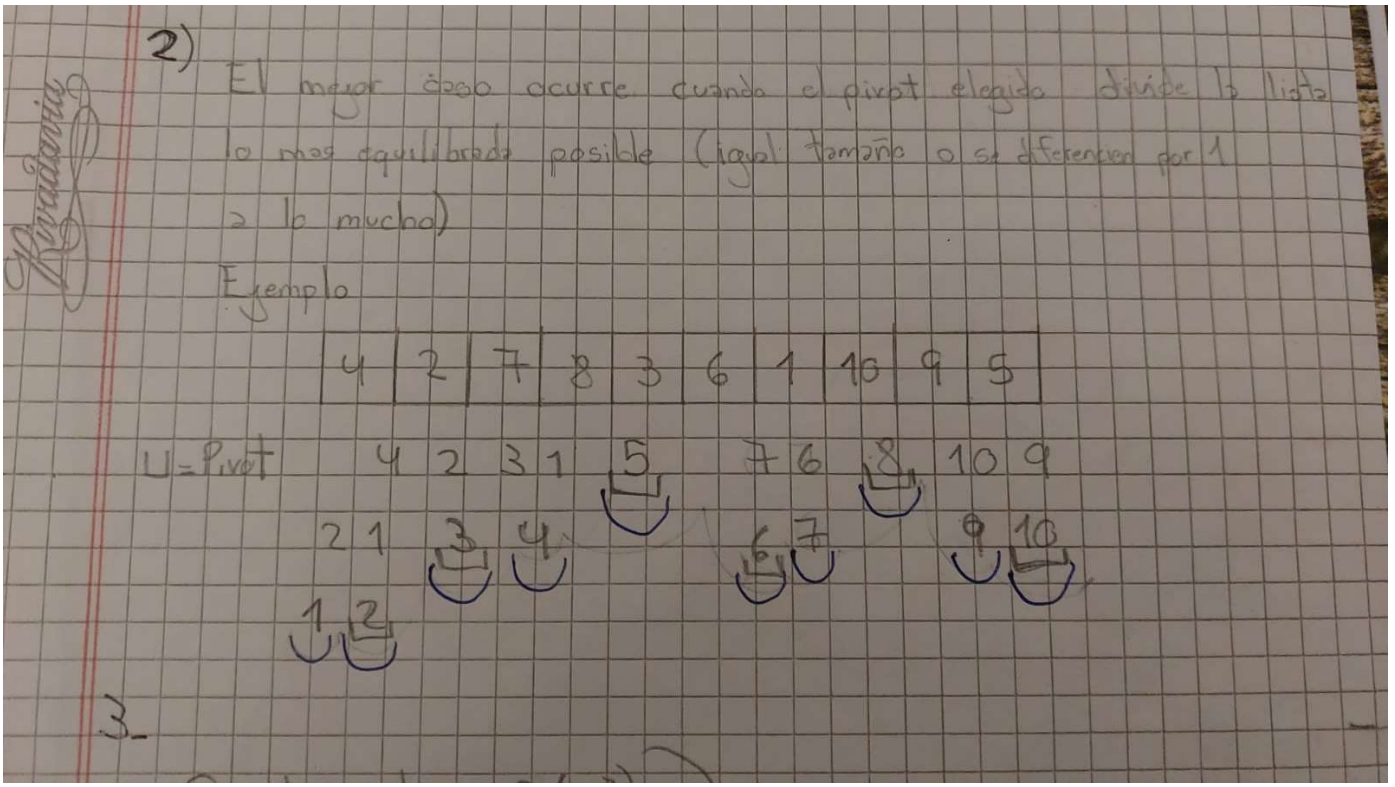
entonces $Cn^3 \leq C \cdot n^2 \quad \forall n \geq n_0$

Operando Algebraicamente $\frac{Cn^3}{n^2} \leq \frac{C \cdot n^2}{n^2} \quad \forall n \geq n_0$

llegamos: $Cn \leq C \quad \forall n \geq n_0$

que es una contradicción porque para cualquier valor que le asignemos a C podemos encontrar un n superior

2)



2)

El mejor caso ocurre cuando el pivot elegido divide la lista lo mas equilibrado posible (igual tamaño o se diferencien por 1 a lo mucho)

Ejemplo

4	2	7	8	3	6	1	10	9	5

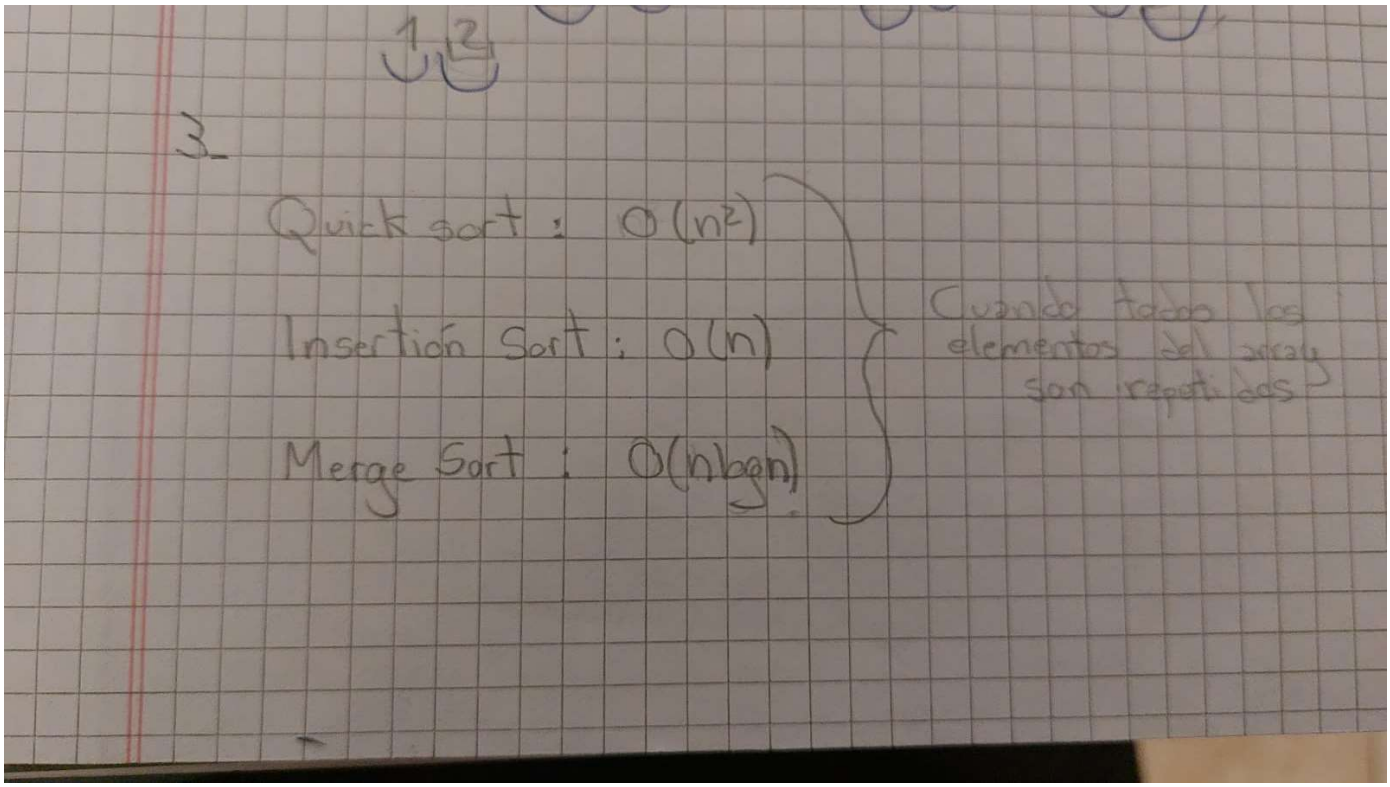
U=Pivot 4 2 3 1 5 7 6 8 10 9

2 1 3 4 6 7 9 10

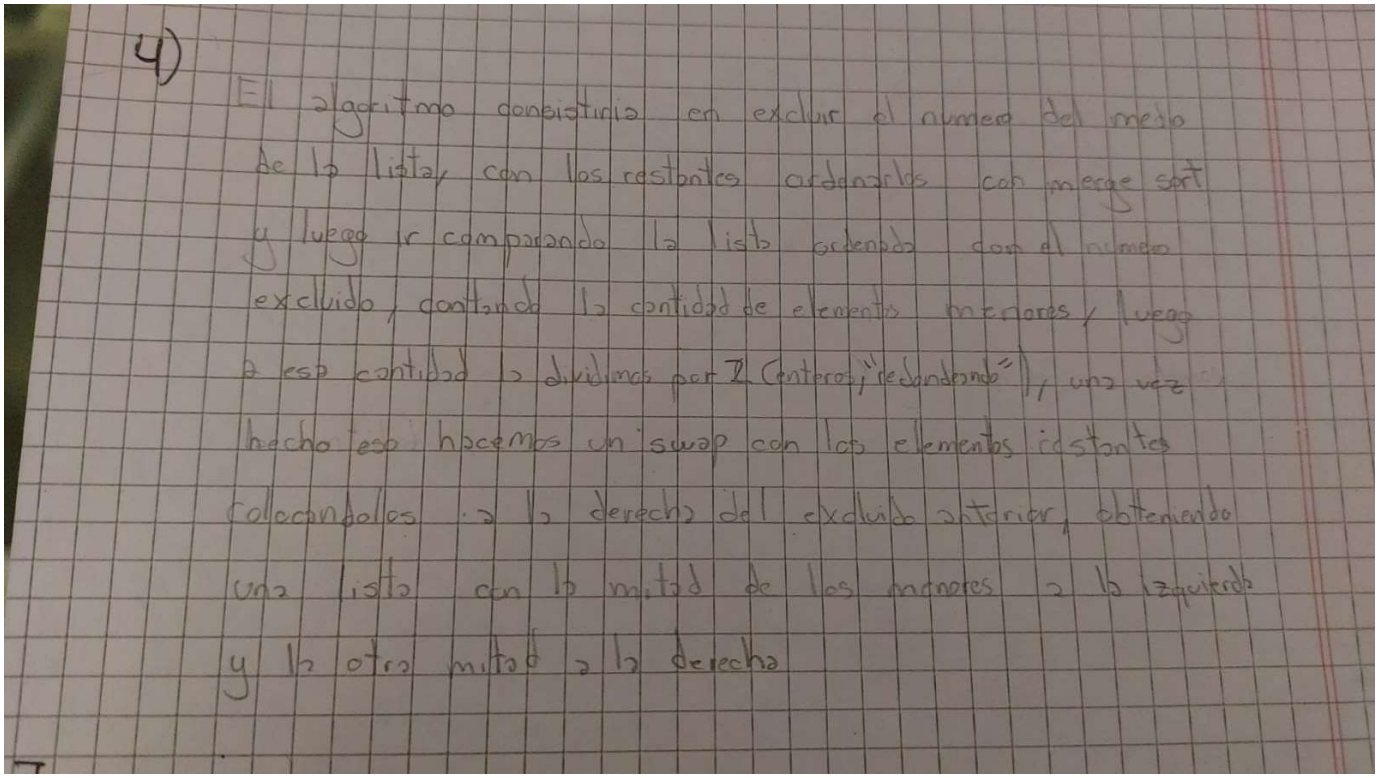
1 2

3-

3)



4)



5)Complejidad O(n^2)

```python
"""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."""
def contienesuma(A,n):
    for i in range(len(A)):
        for j in range(i+1,len(A)):
            if (A[i] + A[j]) == n:
                return True
    return False
```

6)

Bucket Sort

BucketSort es un método de ordenamiento que consiste en dividir un conjunto de elementos a ordenar en varios "baldes" o cubetas, donde cada balde contendrá un rango de elementos. Luego, se ordenan los elementos en cada balde y se combinan los baldes para obtener el conjunto de elementos ordenados.

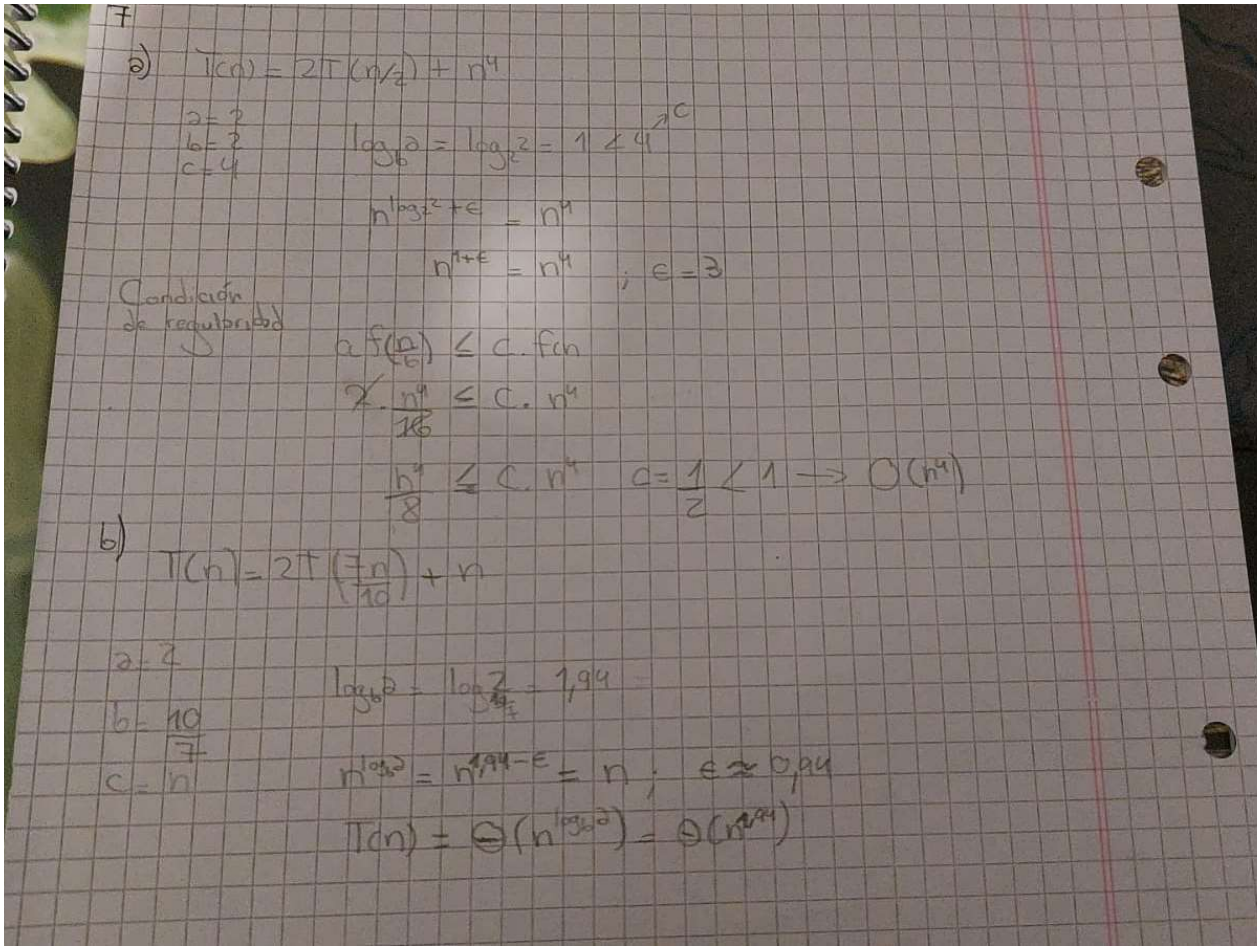
La complejidad temporal de BucketSort depende del número de baldes que se utilicen para dividir los elementos. En el caso promedio, la complejidad es O(n+k), donde n es la cantidad de elementos y k es el número de baldes. Si el número de baldes es proporcional a la cantidad de elementos, la complejidad se acerca a O(n).

El mejor caso de BucketSort ocurre cuando todos los elementos están distribuidos uniformemente en los baldes, lo que significa que cada balde tiene aproximadamente el mismo número de elementos. En este caso, la complejidad temporal de BucketSort es lineal, es decir, O(n).

Por otro lado, el peor caso de BucketSort ocurre cuando todos los elementos se encuentran en un solo balde, lo que significa que se deben ordenar todos los elementos de manera individual. En este caso, la complejidad temporal de BucketSort es O(n^2)

Ejemplo: Funcionamiento caso promedio
Tenemos la lista [0.42, 0.32, 0.12, 0.52, 0.43, 0.34, 0.95, 0.15, 0.75, 0.43].
Determinamos el rango:[0,1]
Utilizaremos baldes en funcion del rango en este caso serian: 10 para poder cubrir 0.1 unidades
Se distribuyen los elementos del arreglo en los baldes segun su valor
Balde 0 = []
Balde 1 = [0.12, 0.15]
Balde 2 = []
Balde 3 = [0.32, 0.34]
Balde 4 = [0.42, 0.43, 0.43]
Balde 5 = [0.52]
Balde 6 = []
Balde 7 = [0.75]
Balde 8 = []
Balde 9 = [0.95]
Ordenamos cada balde individualmente.(se suele usar una versión modificada de Insertion Sort), luego se conectan los baldes y tendríamos el arreglo ordenado
[0.12, 0.15, 0.32, 0.34, 0.42, 0.43, 0.43, 0.52, 0.75, 0.95].

7)



7

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

$a = ?$
$b = ?$
$c = 4$

$\log_b a = \log_2 2 = 1 < 4 \rightarrow c$

$n^{\log_2 2 + \epsilon} = n^4$

$n^{1+\epsilon} = n^4 \quad ; \epsilon = 3$

Condición de regularidad

$a f(\frac{n}{b}) \leq c \cdot f(n)$

$2 \cdot \frac{n^4}{16} \leq c \cdot n^4$

$\frac{n^4}{8} \leq c \cdot n^4 \quad c = \frac{1}{2} < 1 \rightarrow O(n^4)$

b) $T(n) = 2T(\frac{7n}{10}) + n$

$a = 2$
$b = \frac{10}{7}$
$c = 1$

$\log_b a = \log_{\frac{10}{7}} 2 = 1,94$

$n^{\log_b a} = n^{1,94 - \epsilon} = n \quad ; \epsilon \approx 0,94$

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{1,94})$

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

$a = 16$

$b = 4$

$f(n) = n^2$

$\log_b a = \log_4 16 = 2$

$n^{\log_4 2} = f(n)$

$n^2 = n^2$

$T(n) = \Theta(n^{\log_4(2)} \cdot \log n) = \Theta(n^2 \cdot \log n)$

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

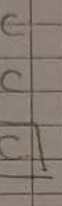
$a = 7$

$b = 3$

$c = n^2$

$\log_b a = \log_3 7$

como es menor que la función "c"



la complejidad es $O(n^2)$

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

$a = 7$

$b = 2$

$c = 2$

$\log_b a = \log_2 7$

estamos en el caso donde $\log_b a$ es mayor que c

entonces su complejidad es $O(n^{\log_b(a)}) = O(n^{2.8})$

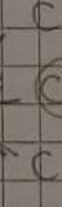
f) $T(n) = 2T(n/4) + \sqrt{n}$

$a = 2$

$b = 4$

$c = \frac{1}{2}$

$\log_b a = \log_4 2 = \frac{1}{2}$



estamos en el caso donde $\log_b a$ es igual a c entonces su complejidad es: $O(n^c \log n) = O(\sqrt{n} \log n)$