Ejercicio 1:

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

T(n) es O(f(n)) si existen constantes positivas c y n0 tal que:

$T(n) \leq cf(n)$ cuando $n \geq n0$

$6n^3 \leq cn^2$ cuando $n \geq 0$, no se cumple para todo n ya que no existe una constante c que cumpla la desigualdad para todo $n \geq 0$.

$6n^3 \leq 7n^3$ cuando $n \geq 0$, se cumple para todo n. Por lo tanto $6n^3$ es de $O(n^3) \neq \boldsymbol{O}(n^2)$.

Ejercicio 2:

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

5	8	10	7	6	9	3	4	2	1

Tomando como pivot el primer elemento de la lista. O(n log 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?

QuickSort(A): $O(n^2)$

Insertion-Sort(A): O(n)

Merge-Sort(A): O(nlogn)

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:

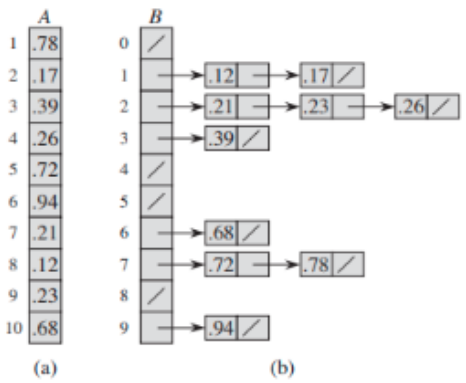
Implementar un algoritmo Contiene-Suma(A,n) que recibe una lista de enteros A y un entero n y devuelve True si existen en A un par de elementos que sumados den n. Analice el costo computacional.

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.

BucketSort

Uno de los requisitos de BucketSort es que cada elemento de la lista ingresada de n elementos debe cumplir: $0 \leq A[i] < 1$, con $0 \leq i < n$.




```
BUCKET-SORT(A)
1  let B[0..n − 1] be a new array
2  n = A.length
3  for i = 0 to n − 1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊n A[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```

Tomando como ejemplo el vector A de entrada, vemos que cumple con la condición de entrada, por lo tanto, procedemos a trabajar con él. Una de las desventajas de BucketSort es la **alta complejidad espacial**, ya que trabajaremos adicionalmente con un array que tiene como elementos LinkedLists.

Este array adicional es, en nuestro caso, el vector B, se llena el vector B de n-1 elementos con punteros a una LinkedList. Luego se procede a recorrer todo el vector A y se inserta el elemento A[i] en la lista de la posición ⌊n*A[i]⌋ de B.

Por ejemplo, el primer elemento de A es 0.78, se multiplica por n=10:

0.78*10 = 7.8 → ⌊7.8⌋ = 7,

Por lo tanto, se inserta 0.78 en la lista de la posición 7 del vector B. Y así se procede con todos los demás elementos de A.

Una vez finalizado ese proceso, se deben ordenar todas las sublistas o **buckets** resultantes en B. Para esto se utiliza el método de ordenamiento InsertionSort. Luego se concatenan las listas resultantes de B en orden desde 0 hasta n-1.

BucketSort tiene un **peor caso de $O(n^2)$** y un **caso promedio y mejor caso de O(n).**

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. **Hacer una análisis por cada algoritmo implementado del caso mejor, el caso peor y una perspectiva del caso promedio.**