Ejercicio 1

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

Supongamos que la cantidad de operaciones elementales de un algoritmo está dada por $T(n) = 6n^3$
Entonces, lo que plantea la desigualdad anterior es que el orden de complejidad en el peor de los casos del algoritmo no puede ser $O(n^2)$, lo cuál es cierto, pues no existe función potencial con potencia 2 tal que multiplicada por una constante cualquiera C acote a la función T cuando n tienda a infinito. Por lo tanto podemos asegurar que el enunciado es verdadero

Ejercicio 2

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

El mejor de los casos de quicksort es cuando el pivote seleccionado siempre termina al centro de la lista, de tal forma que la complejidad es O(nlog(n))
Además, el peor de los casos es cuando los elementos ya se encuentran ordenados o inversamente ordenados, pues en este caso no se divide en n/2 el problema, sinó que se quita solo un elemento de la lista haciendo que el orden de complejidad en el peor caso sea O(n^2)
Por lo tanto el mejor de los casos sería en una lista del siguiente tipo
A = [1, 6, 5, 2, 10, 7, 8, 9, 4, 3]
de esta forma, si el pivote queda en el centro de la lista obtenemos O(nlog(n))

De hecho, esta respuesta depende de cómo se escoja el pivote, si se hace aleatoriamente entonces tenemos que no importa la entrada, a mayor cantidad de elementos de la lista tenemos O(nlog(n))

Ejercicio 3

Cuál es el tiempo de ejecución de la estrategia **Quicksort(A), Insertion-Sort(A)** y **Mergesort(A)** cuando todos los elementos del array A tienen el mismo valor?

En este caso, Insertion sort resulta ser el más rápido, con una complejidad O(n)
MergeSort se mantiene constante O(nlog(n))
QuickSort O(nlog(n))

Ejercicio 4

```python
"""
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.
"""

# Lo primero que voy a hacer es encontrar el valor medio de la lista de entrada,
# de esta manera me aseguro que van a existir elementos mas grandes y mas chicos
# para llevar a cabo tal tarea, voy a utilizar un algoritmo de ordenamiento y luego
# retirar el emenento del medio de la lista
a = [4, 5, 7, 1, 2, 6, 10, 8, 9, 3]

sorted_list = sorted(a)   # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
middle_index = len(sorted_list) // 2 - 1  # 10 // 2 - 1 = 4
middle_value = sorted_list[middle_index]

greater = sorted_list[middle_index+1:]
smaller = sorted_list[:middle_index]
smaller_count = len(smaller)
greater_count = len(greater)

# El índice del valor medio en la nueva lista debe ser el mismo
half_smaller = smaller_count // 2
half_greater = greater_count // 2

result = smaller[:half_smaller] + greater[:half_greater] + [middle_value] + smaller[half_smaller:] + greater[half_greater:]
print(result)
```