Entender el proceso de selección: en cada pasada, identificar el elemento más pequeño de la porción no ordenada de la lista y colocarlo en su posición correcta.

Divide el arreglo en dos partes: la parte ordenada (a la izquierda) y la parte no ordenada (a la derecha).
Comienza desde el segundo elemento (considerando que el primer elemento está ordenado).
Toma el elemento actual (llamado key) de la parte no ordenada y lo compara con los elementos de la parte ordenada.
Mueve los elementos que son mayores que key un lugar a la derecha para hacer espacio.
Inserta key en su posición correcta en la parte ordenada.
Repite hasta que todos los elementos estén ordenados.

Implementar el algoritmo de Selection Sort en cpp.
https://github.com/Marambulag/Algorithms/tree/main/Algorithms

Evaluar la eficiencia del algoritmo, discutiendo la complejidad temporal y espacial.

Time Complexity of Insertion Sort
Worst case: O(n 2)

Space Complexity O(1)

Comparar el rendimiento de Selection Sort con otros algoritmos de ordenamiento, como Bubble Sort, observando diferencias en tiempos de ejecución y número de comparaciones.

Advantages of Insertion Sort:
Simple and easy to implement.
Stable sorting algorithm.
Efficient for small lists and nearly sorted lists.
Space-efficient.
Adoptive. the number of inversions is directly proportional to number of swaps. For example, no swapping happens for a sorted array and it takes O(n) time only.
Disadvantages of Insertion Sort:
Inefficient for large lists.
Not as efficient as other sorting algorithms (e.g., merge sort, quick sort) for most cases.

Probar la implementación con diferentes conjuntos de datos para identificar cómo varía su comportamiento dependiendo del tamaño y el orden inicial de los elementos.

Debido a su complejidad entre más grande peor su rendimiento

Referencia:
https://www.geeksforgeeks.org/selection-sort-algorithm-2/