Comprender el concepto de divide y vencerás: dividir la lista en mitades, aplicar recursividad para ordenar cada mitad y luego combinar las sublistas ordenadas en una sola lista.
Implementar Merge Sort en C++, utilizando la recursividad y manejo eficiente de la memoria para las sublistas.

https://github.com/Marambulag/Algorithms/tree/main/Algorithms
Dividir: divide la lista o matriz de forma recursiva en dos mitades hasta que ya no se pueda dividir.
Conquistar: cada submatriz se ordena individualmente mediante el algoritmo de ordenación por combinación.
Fusionar: los subarreglos ordenados se vuelven a fusionar en el orden ordenado. El proceso continúa hasta que todos los elementos de ambos subarreglos se hayan fusionado.

Analizar la complejidad temporal y espacial del algoritmo, notando que Merge Sort tiene una complejidad temporal de O(n log n).

C**omplejidad temporal**:
Best Case: O(n log n)
Average Case: O(n log n)
Worst Case: O(n log n)

complejidad espacial de Merge Sort es O(n)

Comparar el rendimiento de Merge Sort en C++ con otros algoritmos como Insertion Sort, Bubble Sort y Selection Sort, enfocándose en su ventaja en grandes conjuntos de datos.

Ventajas de merge sort:
Estabilidad: Merge sort es un algoritmo de clasificación estable, lo que significa que mantiene el orden relativo de elementos iguales en la matriz de entrada.
Rendimiento garantizado en el peor de los casos: la ordenación por combinación tiene una complejidad temporal en el peor de los casos de O(N logN), lo que significa que funciona bien incluso en grandes conjuntos de datos.
Fácil de implementar: el enfoque de divide y vencerás es sencillo.
Naturalmente paralelo: fusionamos subarreglos de forma independiente, lo que lo hace adecuado para el procesamiento paralelo.
Desventajas de la merge sort:
Complejidad del espacio: la clasificación por combinación requiere memoria adicional para almacenar los subarreglos combinados durante el proceso de clasificación.
No in situ: la clasificación por combinación no es un algoritmo de clasificación in situ, lo que significa que requiere memoria adicional para almacenar los datos ordenados. Esto puede ser una desventaja en aplicaciones donde el uso de memoria es una preocupación.
Más lento que QuickSort en general. QuickSort es más compatible con el caché porque funciona in situ.

Probar la implementación con diferentes tamaños de listas, observando cómo el algoritmo maneja grandes volúmenes de datos y su capacidad de ordenamiento eficiente.

En general muy estable por su complejidad en worst case

Referencia

https://www.geeksforgeeks.org/merge-sort/