**Comprender la partición y el uso del pivote: aprender cómo se selecciona un pivote y cómo se reorganizan los elementos en torno a él.**

Hay muchas opciones diferentes para elegir pivotes.

Elija siempre el primer (o último) elemento como pivote. La siguiente implementación selecciona el último elemento como pivote.
El problema con este enfoque es que termina en el peor de los casos cuando la matriz ya está ordenada.
Elija un elemento aleatorio como pivote.
Este es un enfoque preferido porque no tiene un patrón para determinar cuál es el peor de los casos.
Elija el elemento mediano como pivote. Este es un enfoque ideal en términos de complejidad temporal, ya que podemos encontrar la mediana en el tiempo lineal y la función de partición siempre dividirá la matriz de entrada en dos mitades. Pero en promedio es bajo, ya que el hallazgo medio tiene constantes altas.

**Implementar Quick Sort en C++ de forma recursiva.**

https://github.com/Marambulag/Algorithms/tree/main/Algorithms
**quick.cpp**

**Analizar la complejidad temporal del algoritmo.**

Mejor caso: O(n log n) cuando el pivote divide el arreglo en partes casi iguales.
Caso promedio: O(n log n) bajo la suposición de que se elige un pivote aleatorio.
Peor caso: O($n^2$) cuando el arreglo está ya ordenado o en orden inverso, y se elige el primer o el último elemento como pivote.

**Comparar el rendimiento de Quick Sort en C++ con otros algoritmos como Merge Sort, Insertion Sort y Selection Sort.**

Ventajas de Quick Sort:
Es un algoritmo de divide y vencerás que facilita la resolución de problemas.
Es eficiente en grandes conjuntos de datos.
Tiene una sobrecarga baja, ya que sólo requiere una pequeña cantidad de memoria para funcionar.
Es compatible con caché ya que trabajamos en la misma matriz para ordenar y no copiar datos a ninguna matriz auxiliar.
El algoritmo de propósito general más rápido para datos de gran tamaño cuando no se requiere estabilidad.
Es recursivo de cola y, por lo tanto, se puede realizar toda la optimización de las llamadas de cola.

Desventajas de la Quick sort:
Tiene una complejidad temporal en el peor de los casos de O(N 2 ), que ocurre cuando el pivote se elige mal.
No es una buena opción para conjuntos de datos pequeños.

No es una clasificación estable, lo que significa que si dos elementos tienen la misma clave, su orden relativo no se conservará en la salida ordenada en caso de una clasificación rápida, porque aquí estamos intercambiando elementos según la posición del pivote (sin considerar su orden original). Posiciones).

**Probar la implementación con diferentes estrategias de elección de pivote (primero, último, aleatorio o mediana de tres) y observar cómo afecta el rendimiento en diferentes tipos de conjuntos de datos.**

**Estrategia de Elección de Pivote**:

- **Primer elemento**: Puede ser ineficiente para arreglos ya ordenados o casi ordenados
- **Último elemento**: Similar al primer elemento, también puede ser ineficiente en arreglos ordenados.
- **Aleatorio**: Seleccionar un pivote aleatorio generalmente ofrece un buen rendimiento
- **Mediana de tres**: Esta estrategia tiende a ser la más eficiente en muchos casos

**Referencias**

https://www.geeksforgeeks.org/quick-sort-algorithm/