**Quick Sort**

El Quicksort es un algoritmo de ordenación eficiente y ampliamente utilizado. se basa en la estrategia de "dividir y conquistar". El algoritmo funciona dividiendo la lista de elementos en particiones, ordenando cada partición de manera independiente y luego combinándolas para obtener la lista ordenada final.

A continuación, se presenta una descripción básica del funcionamiento del Quicksort:

**Algoritmo:**

1. Si la longitud de la lista es 0 o 1, entonces ya está ordenada y se devuelve la lista.

2. Elegir un elemento de la lista como pivote.

3. Particionar la lista alrededor del pivote de manera que los elementos menores que el pivote estén a su izquierda y los mayores estén a su derecha.

4. Aplicar Quicksort recursivamente a las sublistas generadas por la partición (elementos menores y mayores).

5. La lista ordenada es la concatenación de la lista de elementos menores, el pivote y la lista de elementos mayores.

**Pseudocódigo:**

Quicksort(lista):

Si longitud(lista) <= 1:

Devolver lista

Pivote = ElegirPivote(lista)

Particionar lista alrededor del pivote

ListaMenores = Quicksort(sublista de elementos menores que el pivote)

ListaMayores = Quicksort(sublista de elementos mayores que el pivote)

Devolver concatenación de ListaMenores, pivote, y ListaMayores

A continuación, se muestra la implementación de QuickSort.

quicksort.h

#include <iostream>

**struct** Nodo {

**int** dato;

Nodo\* siguiente;

Nodo\* anterior;

Nodo(**int** valor) : dato(valor), siguiente(nullptr), anterior(nullptr) {}

};

**class** **ListaEnlazada** {

**private:**

Nodo\* cabeza;

**public:**

ListaEnlazada();

~ListaEnlazada();

**void** **insertar**(**int** valor);

**void** **mostrar**();

**void** **quicksort**();

**private:**

Nodo\* **obtenerUltimoNodo**();

Nodo\* **Particionar**(Nodo\* inicio, Nodo\* fin);

Nodo\* **QuicksortRecursivo**(Nodo\* inicio, Nodo\* fin);

};

quicksort.cpp

#include "quicksort.h"

ListaEnlazada::ListaEnlazada() : cabeza(nullptr) {}

ListaEnlazada::~ListaEnlazada() {

**while** (cabeza != nullptr) {

Nodo\* temp = cabeza;

cabeza = cabeza->siguiente;

**delete** temp;

}

}

**void** ListaEnlazada::insertar(**int** valor) {

Nodo\* nuevoNodo = **new** Nodo(valor);

**if** (cabeza == nullptr) {

cabeza = nuevoNodo;

} **else** {

Nodo\* temp = cabeza;

**while** (temp->siguiente != nullptr) {

temp = temp->siguiente;

}

temp->siguiente = nuevoNodo;

nuevoNodo->anterior = temp;

}

}

**void** ListaEnlazada::mostrar() {

Nodo\* temp = cabeza;

**while** (temp != nullptr) {

std::cout << temp->dato << " ";

temp = temp->siguiente;

}

std::cout << "**\n**";

}

**void** ListaEnlazada::quicksort() {

cabeza = QuicksortRecursivo(cabeza, obtenerUltimoNodo());

}

Nodo\* ListaEnlazada::obtenerUltimoNodo() {

Nodo\* temp = cabeza;

**while** (temp != nullptr && temp->siguiente != nullptr) {

temp = temp->siguiente;

}

**return** temp;

}

Nodo\* ListaEnlazada::Particionar(Nodo\* inicio, Nodo\* fin) {

**int** pivote = fin->dato;

Nodo\* i = inicio->anterior;

**for** (Nodo\* j = inicio; j != fin; j = j->siguiente) {

**if** (j->dato <= pivote) {

i = (i == nullptr) ? inicio : i->siguiente;

std::swap(i->dato, j->dato);

}

}

i = (i == nullptr) ? inicio : i->siguiente;

std::swap(i->dato, fin->dato);

**return** i;

}

Nodo\* ListaEnlazada::QuicksortRecursivo(Nodo\* inicio, Nodo\* fin) {

**if** (inicio != nullptr && inicio != fin && inicio != fin->siguiente) {

Nodo\* pivote = Particionar(inicio, fin);

QuicksortRecursivo(inicio, pivote->anterior);

QuicksortRecursivo(pivote->siguiente, fin);

}

**return** inicio;

}

Main.cpp

#include "quicksort.cpp"

**int** **main**() {

ListaEnlazada lista;

**int** opcion;

**do** {

std::cout << "**\n**\*\*\* Menu\*\*\***\n**";

std::cout << "1. Insertar elemento**\n**";

std::cout << "2. Mostrar lista**\n**";

std::cout << "3. Aplicar Quicksort**\n**";

std::cout << "0. Salir**\n**";

std::cout << "Ingrese su opcion: ";

std::cin >> opcion;

**switch** (opcion) {

**case** **1**: {

**int** elemento;

std::cout << "Ingrese un elemento: ";

std::cin >> elemento;

lista.insertar(elemento);

**break**;

}

**case** **2**:

std::cout << "Lista actual: ";

lista.mostrar();

**break**;

**case** **3**:

std::cout << "Aplicando Quicksort...**\n**";

lista.quicksort();

std::cout << "Lista ordenada: ";

lista.mostrar();

**break**;

**case** **0**:

std::cout << "Saliendo del programa.**\n**";

**break**;

**default:**

std::cout << "Opción no válida. Inténtelo de nuevo.**\n**";

**break**;

}

} **while** (opcion != **0**);

**return** **0**;

}

**Conclusiones**

El algoritmo Quicksort destaca por su eficiencia y simplicidad en la implementación. Su estrategia "divide y conquista" permite ordenar listas de manera rápida y es especialmente efectivo en conjuntos de datos grandes. La elección inteligente del pivote y la partición eficiente contribuyen a su rendimiento. Sin embargo, es importante considerar que Quicksort puede tener un rendimiento deficiente en casos específicos, como listas ya ordenadas o casi ordenadas, lo que puede afectar su eficacia en ciertos escenarios. A pesar de esto, Quicksort es una elección popular en la práctica y se ha convertido en un algoritmo de referencia para la ordenación.

**Referencia**

Edelkamp, S., Weiß, A., & Wild, S. (2020). QuickXsort: A fast sorting scheme in theory and practice. *Algorithmica*, *82*(3), 509-588.

Hossain, M. S., Mondal, S., Ali, R. S., & Hasan, M. (2020, March). Optimizing complexity of quick sort. In *International conference on computing science, communication and security* (pp. 329-339). Singapore: Springer Singapore.