**Quick Sort folosind varianta secvențială**

Quick Sort este un algoritm de sortare foarte eficient, care funcționează pe principiul „divide et impera”. Ideea de bază este să alegem un element din listă numit **pivot**, să organizăm elementele astfel încât cele mai mici să fie pe stânga, cele mai mari pe dreapta, și apoi să aplicăm recursiv aceeași metodă pe sublistele rezultate.

### **Pașii algoritmului Quick Sort:**

1. **Alegerea pivotului** – De obicei, se ia ultimul element din listă (dar poate fi și unul aleatoriu).
2. **Partiționarea listei** – Se rearanjează elementele astfel încât toate valorile mai mici decât pivotul să fie în stânga acestuia, iar cele mai mari în dreapta. Pivotul ajunge la poziția finală.
3. **Recursivitate** – Se aplică aceiași pași pe cele două subliste (cea din stânga pivotului și cea din dreapta).

### **Specificațiile mașinii pe care am rulat implementarea codului Quick Sort:**

1. **Procesor:** AMD Ryzen 7 4800H
2. **RAM:** 8GB DDR4
3. **Sistem de operare:** Windows 11
4. **Compilator:** Microsoft Visual Studio 2022

| **Șirul introdus** | **Număr elemente** | **Timp de execuție** |
| --- | --- | --- |
| 5 2 3 6 12 | 5 | 2 ms |
| 7 9 10 2 5 4 12 45 23 11 1 3 14 32 | 14 | 3 ms |
| 23 14 12 10 16 17 22 8 2 4 5 7 9 10 1 67 98 42 57 29 | 20 | 4 ms |
| 1 4 5 9 3 2 10 8 7 15 98 56 32 34 6 34 18 23 101 87 98 43 56 78 90 88 | 26 | 7 ms |

**Codul sursă**

#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Funcție pentru a găsi poziția pivotului și a rearanja elementele

int partition(vector<int>& arr, int st, int dr) {

int pivot = arr[dr]; // Alegem pivotul ca ultimul element

int i = st - 1; // Poziția pentru elementele mai mici decât pivotul

for (int j = st; j < dr; j++) {

if (arr[j] < pivot) {

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[dr]); // Plasăm pivotul în poziția corectă

return i + 1;

}

// Funcția QuickSort

void quickSort(vector<int>& arr, int st, int dr) {

if (st < dr) {

int pivotIndex = partition(arr, st, dr); // Găsim pivotul

quickSort(arr, st, pivotIndex - 1); // Sortăm partea stângă

quickSort(arr, pivotIndex + 1, dr); // Sortăm partea dreaptă

}

}

int main() {

int n;

cout << "Introduceti numarul de elemente: ";

cin >> n;

vector<int> v(n);

cout << "Introduceti " << n << " numere: ";

for (int i = 0; i < n; i++) {

cin >> v[i];

}

// Începem măsurarea timpului

auto start = high\_resolution\_clock::now();

quickSort(v, 0, v.size() - 1);

// Terminăm măsurarea timpului

auto stop = high\_resolution\_clock::now();

// Calculăm durata

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Vectorul sortat: ";

for (int x : v) {

cout << x << " ";

}

cout << "\nTimp de executie: " << duration.count() << " ms" << endl;

return 0;

}

***Funcția partition***

* Alege pivotul (ultimul element din interval).
* Mută elementele mai mici decât pivotul în stânga acestuia.
* Returnează poziția finală a pivotului.

***Funcția quickSort***

* Sortează recursiv sub-vectorii din stânga și dreapta pivotului.

***Funcția main***

* Citește un vector de la tastatură.
* Măsoară timpul de execuție al algoritmului folosind chrono::high\_resolution\_clock.
* Afișează vectorul sortat și timpul de execuție.