Tema 1 de laborator

Aioanei Florin Verzotti Matteo Voaides Robert

Compararea unor algoritmi de sortare prin comportamentul lor asupra mai multor suite de teste

Cuprins

1 Introducere

În această lucrare ne propunem să comparăm performanțele mai multor algoritmi de sortare din punct de vedere al complexității teoretice și al performanței practice. Scopul este de a evidenția avantajele și dezavantajele fiecărui algoritm în funcție de dimensiunea și natura datelor de intrare.

2 Algoritmi analizați

- Quick Sort
  - Random Pivot
  - Median Pivot
  - Half Pivot
  - Ternary Quick Sort
- Radix Sort
  - base 10
  - base 16
  - base 2^{16}
- Merge Sort
- Intro Sort
- Tim Sort
- Shell Sort

3 Complexitate teoretică

Tabela 1: Tabel comparativ al complexităților algoritmilor analizați

Algoritm	**Best case**	**Average**	**Worst Case**
Quick Sort (Random Pivot)	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$
Quick Sort (Median Pivot)	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Quick Sort (Half Pivot)	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$
Ternary Quick Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$
Radix Sort (base 10)	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$
Radix Sort (base 16)	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$
Radix Sort (2^{16})	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$
Merge Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Intro Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Tim Sort	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Shell Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{5/4})$	$\mathcal{O}(n^2)$

4 Implementare

Toți algoritmii au fost organizați într-un repository GitHub, structurat clar. Fiecare algoritm de sortare are propriul său header dedicat în folderul include/, iar implementarea este realizată în fișiere .cpp separate.

5 Metodologie experimentală

Pentru testarea performanțelor, am creat un sistem automatizat de rulare și înregistrare a testelor, organizat în jurul fișierului principal `main.cpp`. Setările fiecărui test sunt definite într-un fișier de configurare `test_config.csv`, unde fiecare linie specifică: `nameoftest, numberofvalues, maxvalue`. Aceste configurații sunt parcurse și procesate cu ajutorul funcției `readTestConfigs()`, iar pentru fiecare algoritm de sortare implementat în sistem, se generează un vector aleatoriu corespunzător parametrilor testului.

Toți algoritmii sunt înregistrați într-un map, ce permite selectarea și rularea lor pe baza numelui.

Timpul de execuție pentru fiecare sortare este măsurat cu funcția `measureTime()`, iar rezultatele sunt salvate în fișierul `results.csv`, folosind funcția `logResult()` incluzând: numele testului, algoritmul, dimensiunea vectorului, valoarea maximă, timpul de execuție (în secunde) și dacă vectorul rezultat este sortat corect.

Pentru o testare completă, aplicația poate fi rulată cu comanda `./main all`, caz în care se rulează toți algoritmii în toate testele definite.

Fiecare algoritm este rulat o singură dată per configurație, iar logica poate fi extinsă ușor pentru repetare multiplă și calculul unei medii. Măsurarea timpului s-a realizat utilizând funcționalități din biblioteca standard `<chrono>`.

6 Rezultate experimentale

6.1 Numere putine, valori dispersate

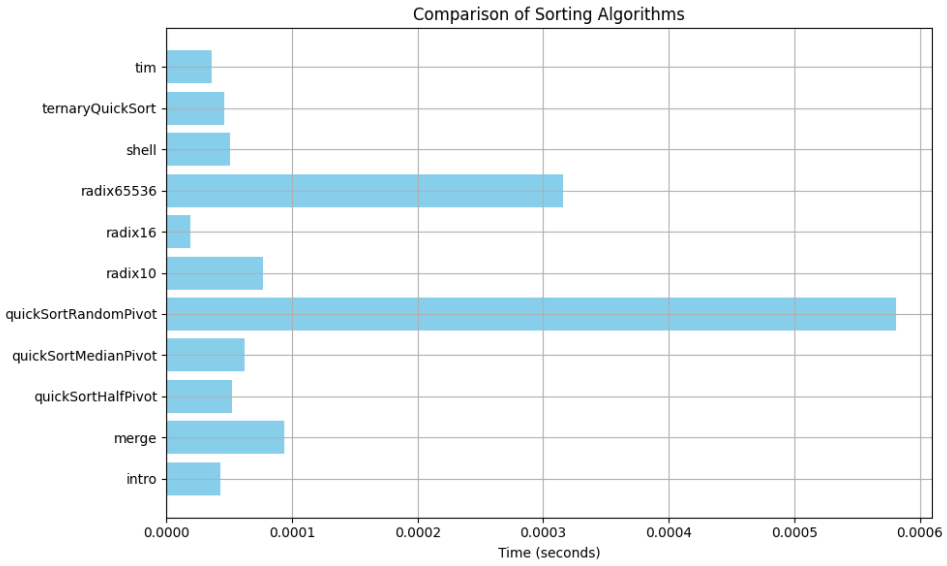

Figura 1: Pentru 1000 de numere si valori intre 0 si 10^9

7 Concluzii

- Algoritmii simpli sunt potriviți doar pentru seturi mici de date.
- Quick Sort este ideal pentru cazuri generale, dar trebuie tratat cu atenție pentru cazurile nefavorabile.
- Merge Sort este fiabil și stabil, ideal pentru date mari și sortare externă.

Anexă: Cod sursă (fragment)

```
void bubbleSort(int arr[], int n) {
    bool swapped;
    for (int i = 0; i < n-1; i++) {
        swapped = false;
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                std::swap(arr[j], arr[j+1]);
                swapped = true;
            }
        }
        if (!swapped)
            break;
    }
}
```