**Eficiența algoritmilor de sortare**

În acest document sunt prezentate testele realizate în limbajul C++ pentru 6 algoritmi de sortare foarte cunoscuți. Aceștia sunt RadixSort, MergeSort, ShellSort, CountingSort, BubbleSort și STL Sort (sortarea nativă a limbajului folosit).

În cele ce urmează vor fi prezentate pe rând testele și concluziile trase.

**Primul test** este bazat pe sortarea a **10^6 elemente** care au fost inițial în **ordine** **descrescătoare**. Rezultatele sunt următoarele:

Radix (baza 10): 97 milisecunde

Radix (baza 64): 56 milisecunde

Radix (baza 2^16): 26 milisecunde

Merge: 109 milisecunde

Shell: 82 milisecunde

Counting: 7 milisecunde

STL: 36 milisecunde

Din cauza complexității, BubbleSort-ul a fost testat doar pe 10^5 elemente și a realizat sortarea într-un timp de 25 secunde, astfel, putem concluziona că pentru sortarea a 10^6 elemente timpul ar fi fost de aproximativ 100 de ori mai mare, adică aproximativ 40 de minute.

Concluziile trase din primul test sunt următoarele:

- o bază mai mare scade semnificativ timpul de sortare din cadrul algoritmului RadixSort

- MergeSort și ShellSort au timp asemănători, dat fiind și complexitatiile medii ale acestor algoritmi fiind O(n^2 \* log n)

- CountingSort este cel mai rapid deoarece maximul din vector este 10^6, un număr relativ mic

**Al doilea test** este bazat pe sortarea a **10^6 numere random** care aparțin **intervalului [1,10^3]**. Timpii sunt media a 10 teste.

Radix (baza 10): 56 milisecunde

Radix (baza 64): 30 milisecunde

Radix (baza 2^16): 14 milisecunde

Merge: 162.70 milisecunde

Shell: 4.6 secunde

Counting: 3.7 milisecunde

STL: 103 milisecunde

Bubble: -

Putem trage o nouă concluzie din acest test, mai exact știm că timpul de sortare al ShellSort-ului este puternic influențat de ordinea în care se afla elementele din cauza gap-ului.

**Al treilea test** este bazat pe sortarea a **10^6 numere random** care aparțin **intervalului [1,10^9]**. Timpii sunt media a 10 teste.

Radix(10): 134 milisecunde

Radix(64): 84.5 milisecunde

Radix(2^16): 36.8 milisecunde

Merge: 187.5 milisecunde

Shell: 408 milisecunde

Counting: 3.6 secunde

STL: 140.5 milisecunde

Bubble: -

Concluziile trase din al treilea test sunt următoarele:

- 10^9 elemente devin un număr ridicat și pentru CountingSort, acesta încetinind semnificativ

- deși sunt la fel de multe elemente ca la testul 2, ShellSort-ul este mai rapid, probabil din cauza faptului că majorarea maximului atât de mult duce la o probabilitate mult mai mică de a apărea mai multe elemente cu valori egale

**Al partulea test** este bazat pe sortarea unui vector cu **10^5 elemente** care este **aproape sortat**.

Funcția care generează vectorul aproape sortat este bazată pe alegerea unui număr random (să îl numim m) intre n / 1000 și n / 500 urmând să fie făcute m swap-uri de elemente alese random. Timpii sunt media a 10 teste.

Radix (baza 10): 6.9 milisecunde

Radix (baza 64): 3.3 milisecunde

Radix (baza 2^16): 3 milisecunde

Merge: 10 milisecunde

Shell: 11.8 milisecunde

Counting: 0.7 milisecunde

STL: 5.5 milisecunde

Medie Bubble: 5 secunde

O altă concluzie pe care o putem trage este că timpul de sortare al BubbleSort-ului scade când vectorul este aproape sortat, acesta fiind de 5 ori mai rapid decât în cazul în care cele 10^5 elemente erau sortate descrescător.

**Al cincilea test** este bazat pe sortarea a **10^3 numere random** care aparțin **intervalului [1,10^9]**. Timpii sunt media a 10 teste.

Toți algoritmii au media între 0 și 0.2 milisecunde dar există și anumite excepții:

1. Radix în baza (2^16) are media 0.5 milisecunde
2. Counting 3 secunde
3. Bubble 1.6 milisecunde

Concluziile pe care le putem trage sunt urmatorele:

- Creșterea bazei în care este făcut RadixSort-ul scade timpul de sortare până într-un anumit punct optim, după acel punct timpii de sortare încep să crească.

- CountingSort-ul nu este eficient în cazul în care vrem să sortăm numere care au un maxim ridicat

**Ultimul test** este bazat pe sortarea unui vector **10^7** **elemente** care este deja **sortat crescător**.

Radix(10): 1.1 secunde

Radix(64): 561 milisecunde

Radix(2^16): 288 milisecunde

Merge: 1.2 secunde

Shell: 388 milisecunde

Counting: 73 milisecunde

STL: 593 milisecunde

Bubble: 10 milisecunde

Concluzia pe care o putem trage este că pentru un vector sortat (sau aproape sortat) BubbleSort-ul este un algoritm foarte eficient.

O **concluzie finală** importantă este faptul că fiecare algoritm de sortare are avantajele și dezavantajele lui și cu cât avem mai multe informații despre ce fel de vector urmează să fie sortat, avem o șansă mai mare să alegem algoritmul potrivit pentru acea situație.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Radix(10) | Radix(64) | Radix(2^16) | Merge | Shell | Counting | STL | Bubble |
| Test 1 |  |  |  |  |  |  |  |  |
| Test 2 |  |  |  |  |  |  |  |  |
| Test 3 |  |  |  |  |  |  |  |  |
| Test 4 |  |  |  |  |  |  |  |  |
| Test 5 |  |  |  |  |  |  |  |  |
| Test 6 |  |  |  |  |  |  |  |  |

În tabelul de mai sus, culorile pe fiecare test semnifică următoarele:

Verde - timpul de sortare se află pe locurile 1-3

Galben - timpul de sortare estese află pe locurile 4-5

Roșu - timpul de sortare se află pe locurile 6-8