**Eficiența algoritmilor ce sortare**

Documentul acesta prezintă concluziile trase în urma testării unor algoritmi de sortare în limbajul C++. Algoritmii studiați sunt Bubblesort, Mergesort, Radixsort, Countsort, Shellsort. Am realizat și o comparație cu Sort-ul nativ al limbajului.

Tabelul următor ilustrează cazurile în care fiecare dintre algoritmii de sortare selectați sunt eficienți. Obs:Algoritmul de QuickSort nu este testat cu aceeași precizie ca celelalte metode de sortare și este prezent pentru o comparație cu Sort-ul nativ al limbajului de programare C++. Obs(2): Comparația cu QS nu este cea mai bună deoarece selecția pivotului este făcuta într-un mod ineficient, prin alegerea mijlocului.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Timpii sunt in secunde | Algoritmul folosit | | | | | | |
| Testul | Bubblesort | Mergesort | Radix | Shellsort | Countsort | STL Sort | QS |
| 1 | 0.25 | ~0.001 | <0.001 | 0.0028 | <0.001 | ~0.001 | ~0.001 |
| 2 | 29.2 | 0.013 | 0.0013 | 0.021 | <0.001 | 0.012 | 0.017 |
| 3 | <0.001 | 0.007 | 0.003 | 0.004 | <0.001 | 0.005 | 0.006 |
| 4 | >300 | 0.14 | 0.035 | 0.24 | 0.014 | 0.18 | 0.15 |
| 5 | >300 | 1.26 | 0.21 | 1.32 | 0.04 | 1.21 |  |
| 6 | >300 | 19.3 | 3.85 | 49.9 | 1.21 | 17.03 |  |
| 7 | >300 | 18.9 | 4.3 | 27.65 | 2.71 | 38.29 | 39.35 |
| 8 | 0.3 | 13.2 | 4.8 | 8.8 | 0.8 | 9.2 | 7.9 |

Legenda testelor:

1. 10^4 numere din intervalul [1, 10^5]
2. 10^5 numere din intervalul [1, 10^5]
3. 10^5 numere din intervalul [1, 10^5], aproape sortat.
4. 10^6 numere
5. 10^7 numere din intervalul [1,100]
6. 10^8 numere din intervalul [99990000-10^8] (numere mari)
7. 10^8 numere din intervalul [1, 10^8]
8. 10^8 numere din intervalul [1, 10^8], sortat.

În cazul testelor pentru algoritmul Radix am folosit baze diferite, rezultatul din tabel fiind media testelor cea mai mică, respectiv baza corespunzătoare cazului respectiv. Testele au fost facute pentru:

* 10
* 2^10
* 2^11
* 2^12
* 2^13
* 2^15
* 2^16

Concluziile testelor făcute sunt următoarele:

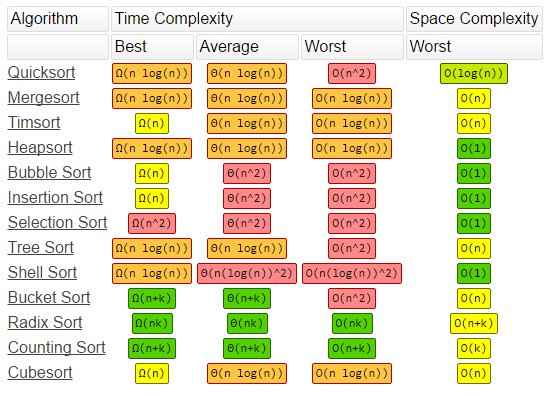
În cazurile des întâlnite, mai precis seturi medii spre mari de date și nesortate, algoritmul cel mai eficient pentru sortarea numerelor naturale este Countsort, de complexitate O(n+max), unde n este numărul de elemente, iar max e elementul cel mai mare din setul de date. Dezavantajul acestui algoritm este utilizarea unei memorii adiționale de O(max). Cazul pe care vrem să îl evităm când folosim acest algoritm este cel în care avem un set mic de date, în care numerele sunt mari.

O performanță ce se apropie de cea a Countsort-ului este cea a Radixsort-ului. Aceeași problemă se regăsește și aici, cea a spațiului adițional: O(n+max).

Algoritmii care nu necesită spațiu adițional în acest caz sunt Bubblesort și Shellsort. În schimb, performanța lor scade drastic, fiind eficienți doar în cazuri limită, precum vectori aproape sortați.

Cel mai echilibrat algoritm este Mergesort-ul, deoarece are o performanță medie pentru aproape toate testele, iar spațiul adițional este doar de O(n).

În final, se observă că algoritmul nativ de sortare al limbajului C++ are o performanță medie, utilizând și spatiu adițional de O(log(n)).

Testele făcute ilustrează aproximativ complexitățile fiecărui algoritm:

Sursă: https://lamfo-unb.github.io/2019/04/21/Sorting-algorithms/