Merge Sort

Counting Sort

PROIECT SORTARI

Quick Sort

O comparatie intre cele mai populare soratri

Radix Sort

Shell Sort

STRUCTURI DE DATE

Udrea Robert-George

GRUPA 133

DESPRE

Proiectul a fost scris **C++** si compilat cu **g++** (cu flag-ul **-O3**).

Vectorii sortati au fost generati random.

In implementarea programului am folosit vectorii din **STL**.

Procesor: 1,8 GHz Dual-Core Intel Core i5

*Prezentarea a fost realizata in Keynote(MacOS), iar variantele PDF/ PPTX nu afiseaza corect unele diagrame.

MERGE SORT

MERGE SORT

	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
10	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s
10^2	0.00002s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s
10^3	0.00013s	0.00013s	0.00013s	0.00010s	0.00010s	0.00010s	0.00008s	0.00006s
10^4	0.00142s	0.00070s	0.00070s	0.00070s	0.00117s	0.00087s	0.00070s	0.00087s
10^5	0.00951s	0.00813s	0.00810s	0.00870s	0.00853s	0.00817s	0.00814s	0.00808s
10^6	0.09675s	0.09585s	0.09554s	0.09963s	0.10755s	0.09647s	0.10046s	0.09713s
10^7	1.06830s	1.09131s	1.09142s	1.09721s	1.09309s	1.09839s	1.19039s	1.09580s
10^8	12.07922s	12.32391s	13.01862s	12.49393s	12.35145s	12.36039s	12.34904s	12.36395s

INTERPRETAREA REZULTATELOR

- Se pare ca **Merge Sort** se tine bine sub 0.5s cu cel mult 1.000.000 de numere. De aici insa...se observa o crestere destul de abrupta: **0.1s -> 1s -> 12s**. Totusi, rezultatele sunt unele satisfacatoare.

- ***Am compilat si cu **clang** fara nicio optimizare si timpii obtinuti au fost **catastrofali**: la **100.000.000** de numere ajungeam si la **>1min**.

- **Concluzie:** pentru un numar modest de elemente, Merge Sort face o treaba mai mult decat buna.

RADIX SORT

RADIX SORT

	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
10	0.00001s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s
10^2	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00002s	0.00002s	0.00002s
10^3	0.00011s	0.00009s	0.00014s	0.00014s	0.00016s	0.00016s	0.00018s	0.00022s
10^4	0.00080s	0.00102s	0.00108s	0.00106s	0.00068s	0.00081s	0.00094s	0.00107s
10^5	0.00316s	0.00429s	0.00533s	0.00741s	0.00832s	0.01029s	0.01010s	0.01174s
10^6	0.02899s	0.04226s	0.05564s	0.06905s	0.08308s	0.09831s	0.11158s	0.11164s
10^7	0.29607s	0.41881s	0.55715s	0.69242s	0.83442s	0.97378s	1.12132s	1.11268s
10^8	3.41071s	4.43874s	6.60474s	7.83811s	9.54420s	10.25339s	11.41096s	14.49516s

INTERPRETAREA REZULTATELOR

- Ca si Merge Sort, **Radix** se tine bine sub **0.5s** la cel mult 1.000.000 numere. Totusi se oberva ca, pentru numere cu cel mult **6 cifre, Radix Sort** este mai rapid decat Merge.
- La 10, respectiv **100 de milioane de numere**, avem doua extreme: pe numere mici **Radix Sort este mai rapid** decat Merge, dar pe numere mari, **Merge Sort preia stafeta**, Radix ajungand si la **14s.**
- **Concluzie:** Radix Sort are timpi similari cu Merge Sort, dar este clar mai rapid pentru numere mai mici.

SHELL SORT

SHELL SORT

	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
10	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s
10^2	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s
10^3	0.00008s	0.00020s	0.00013s	0.00013s	0.00013s	0.00014s	0.00014s	0.00014s
10^4	0.00098s	0.00142s	0.00131s	0.00106s	0.00099s	0.00100s	0.00112s	0.00100s
10^5	0.00560s	0.00912s	0.01130s	0.01327s	0.01397s	0.01435s	0.01424s	0.01348s
10^6	0.06543s	0.09838s	0.13793s	0.15463s	0.16887s	0.17742s	0.17898s	0.17995s
10^7	0.74931s	1.16145s	1.61706s	1.97468s	2.19587s	2.67755s	2.81908s	2.41041s
10^8	9.05766s	14.05155s	18.40534s	25.33983s	28.71879s	32.27616s	33.29508s	34.15900s

INTERPRETAREA REZULTATELOR

- Ca si Merge si Radix, **Shell Sort** nu depaseste 0.5s pe cel mult 1.000.000 numere.
- Pentru **10 milioane** de numere timpii sunt cel putin **dublii** fata de Radix si Merge..., lucru care denota **ineficienta algoritmului**, indiferent de ‘plaja’ de numere aleasa.
- La **100.000.000, timpul explodeaza.** Ajungem sa depasim 30s... . Pana acum Merge Sort este cel mai constant la 100 de milioane de numere, **Radix si Shell** variind de la **3s la 14s, respectiv de la 9s la 34s.**
- **Concluzie:** La o privire de ansamblu, desi se descurca bine cu mai putine numere, Shell Sort are rezultate si de doua ori mai slabe decat cele doua rivale prezentate anterior.

COUNTING SORT

COUNTING SORT

	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
10	0.00001s	0.00000s	0.00001s	0.00008s	0.00067s	0.00460s	0.02767s	0.27409s
10^2	0.00001s	0.00000s	0.00001s	0.00006s	0.00050s	0.00372s	0.02904s	0.27862s
10^3	0.00001s	0.00005s	0.00006s	0.00012s	0.00158s	0.00427s	0.02755s	0.27529s
10^4	0.00008s	0.00007s	0.00007s	0.00021s	0.00061s	0.00295s	0.02876s	0.27628s
10^5	0.00045s	0.00033s	0.00032s	0.00040s	0.00130s	0.00440s	0.03118s	0.28196s
10^6	0.00284s	0.00291s	0.00286s	0.00308s	0.00564s	0.01488s	0.06390s	0.31931s
10^7	0.02871s	0.02915s	0.02990s	0.03022s	0.06541s	0.09799s	0.32638s	0.70627s
10^8	0.30035s	0.28656s	0.28967s	0.31815s	0.50191s	0.94370s	2.70098s	4.25484s

INTERPRETAREA REZULTATELOR

- **Wow... Counting Sort** a cam pus la colt celelalte sortari. La fel ca si competitorii sai, pana in 1.000.000 de elemente se mentine sub 0.5s.
- La **10 milioane**, 7/8 teste raman sub 0.5s, iar al 8-lea nu depaseste 1s.
- La **100.000.000,** majoritatea testelor **nu depasesc 1s**, iar cel mai mult ajunge la **~4s**. Totusi, Counting Sort are un '**Calcaiul lui Ahile**', acela fiind valoarea maxima din vector: cand avem de sortat numere care ating **10.000.000.000(**foarte mari**), algoritmul cedeaza.**
- **Concluzie:** Counting Sort nu este influentat de numarul de elemente cat este influentat de valoarea maxima a elementelor. Pe numere pe care stim sa le citim fara dificultate, Counting Sort preia stafeta ca fiind cel mai rapid algoritm de pana acum.

QUICK SORT

MENTIUNE

**In cod am implementat doua variante de a alege pivotul,
random si cu mediana din 3, dar timpii obtinuti au fost similari.**

QUICK SORT

	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
10	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s
10^2	0.00005s	0.00004s	0.00004s	0.00004s	0.00004s	0.00004s	0.00004s	0.00004s
10^3	0.00057s	0.00043s	0.00036s	0.00035s	0.00038s	0.00038s	0.00040s	0.00038s
10^4	0.01091s	0.00427s	0.00333s	0.00261s	0.00395s	0.00287s	0.00262s	0.00260s
10^5	0.43139s	0.06823s	0.02935s	0.02554s	0.02127s	0.02134s	0.02052s	0.02126s
10^6	41.10977s	4.62937s	0.69008s	0.30583s	0.26124s	0.21974s	0.21854s	0.22004s
10^7	😕	442.02092s	46.81154s	7.04181s	3.19912s	2.72271s	2.32261s	2.30998s
10^8	😕	😕	😕	473.00140s	72.05877s	33.37423s	30.72823s	25.55059s

INTERPRETAREA REZULTATELOR

- Pentru a se mentine sub pragul de **0.5s**, **Quick Sort** face un **pas in spate** si se limiteaza la 100.000 de numere.
- La **1.000.000** de numere timpii sunt putin mai mari decat competitia, dar aici isi face simtita prezenta **slabiciunea lui QuickSort**: intervalul din care luam numere. Vedem **~40s** pentru un vector cu 0<= nr <=10...
- La **10 milioane** de numere timpii ajung sa fie **similari cu Shell Sort**, ramanand mai slabi decat celelalte. Dar din nou... pentru **numere mici**...nici dupa cateva zeci de minute nu a sortat.
- Pentru **100 de milioane,** timpii ajung sa se apropie de Shell Sort...si cam atat. Pentru numere de pana in **1.000**, dureaza **zeci de minute.**
- **Concluzie:** Pentru o **plaja mare de numere**, QuickSort este ceva mai rapid decat ShellSort.Insa pentru **numere mici**... fiind valori apropiate, se efectueaza **foarte multe comparatii**. Pentru QuickSort, cu cat sunt **valori mai diverse** si mai **bine rasfirate**, cu atat este mai bine.

NATIVE SORT

C++

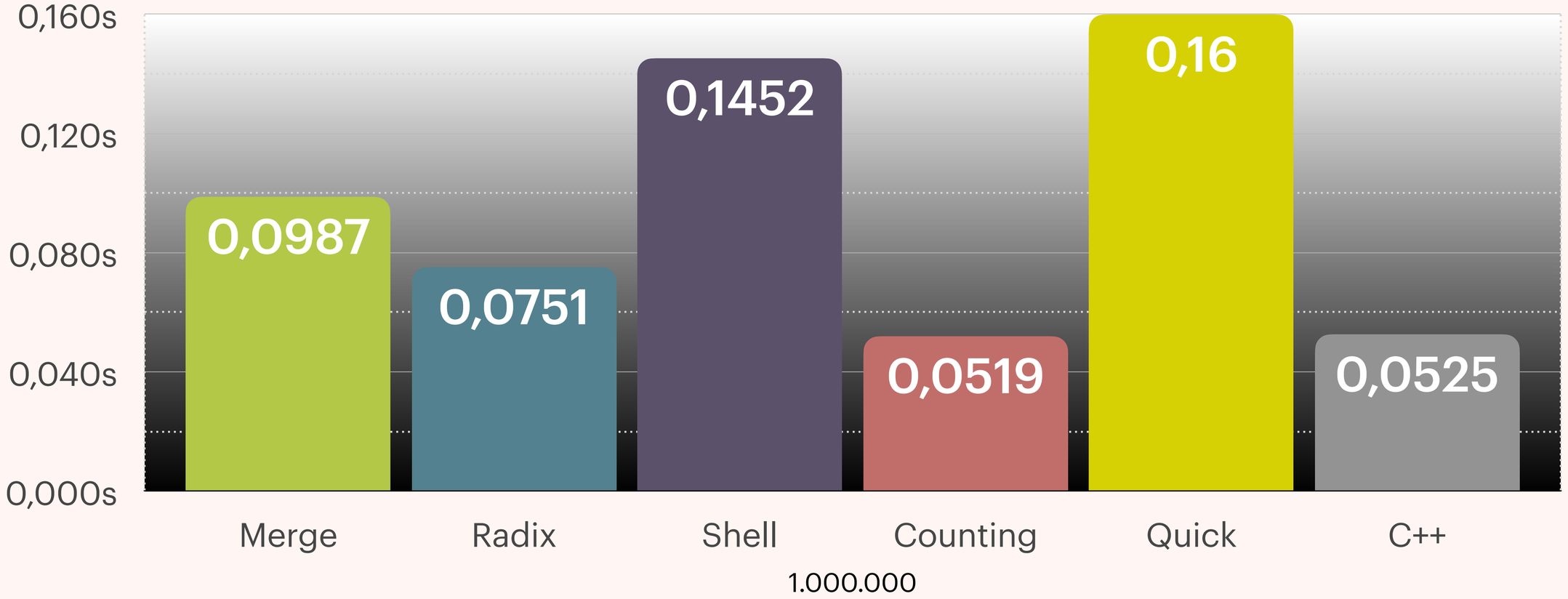
NATIVE SORT C++

	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
10	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s	0.00000s
10^2	0.00005s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s	0.00001s
10^3	0.00027s	0.00006s	0.00007s	0.00010s	0.00007s	0.00007s	0.00007s	0.00006s
10^4	0.00039s	0.00052s	0.00082s	0.00089s	0.00086s	0.00071s	0.00069s	0.00070s
10^5	0.00248s	0.00352s	0.00390s	0.00562s	0.00918s	0.00646s	0.00658s	0.00598s
10^6	0.01466s	0.02648s	0.04035s	0.05166s	0.07001s	0.07244s	0.07152s	0.07256s
10^7	0.13735s	0.27222s	0.39919s	0.52189s	0.64865s	0.80987s	0.85388s	0.84708s
10^8	1.40819s	2.64941s	4.03399s	5.14074s	6.43495s	7.67391s	9.40661s	9.72622s

INTERPRETAREA REZULTATELOR

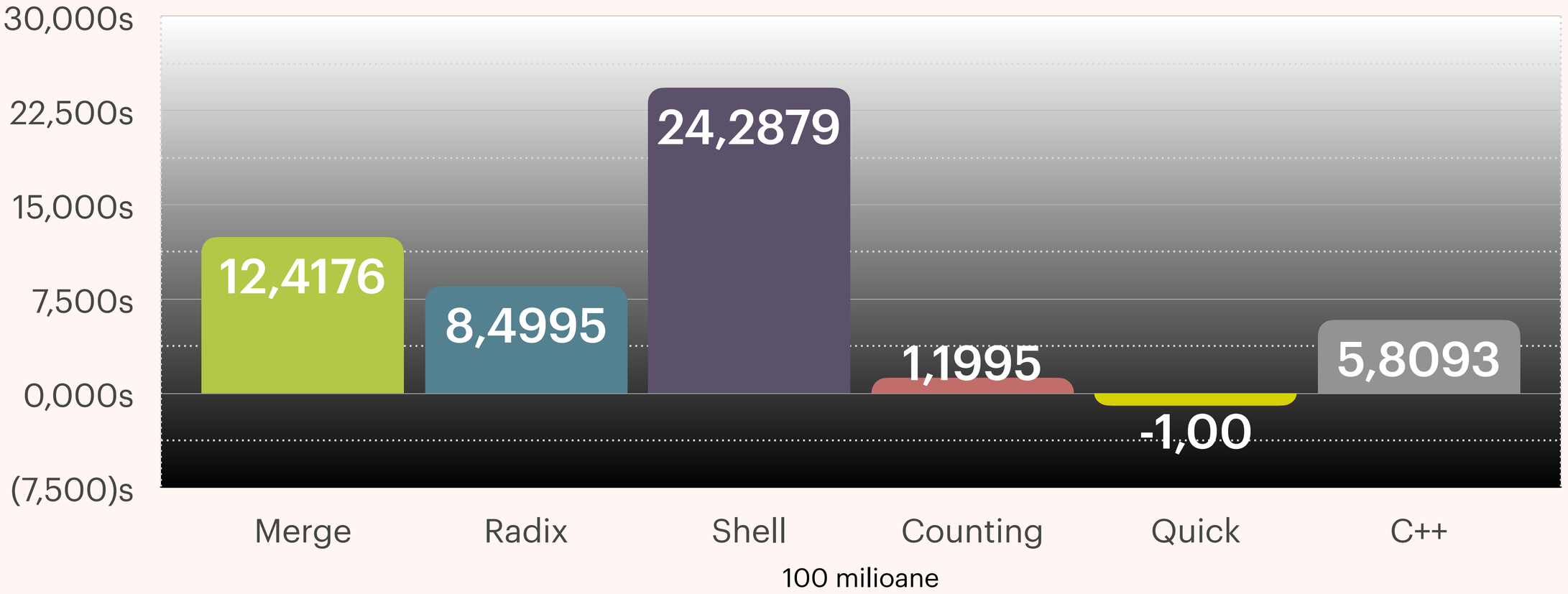
- Pana in **1.000.000**, si sortarea din **C++** se mentine sub **0.5s**.
- Pentru **10 milioane** de numere, pentru numere modeste **nu depaseste 0.5s**, iar atunci cand depaseste, la numere mari, tot ramane **sub 1s**, ceea ce ne bucura.
- La **100.000.000** de numere, timpii sunt mai buni decat la majoritatea sortarilor si **nu depasesc 10s.**
- **Concluzie:** Nu pare sa existe vreo slabiciune sensibila, ca la Counting/Quick. Sortarea are **timpi optimi**, fiind prima sortare dupa Counting Sort la numarul de sortari in **mai putin de o secunda.**

CLASAMENT <=1.000.000



*la 1.000.000 numere QuickSort are 5.96s. Am trecut 0.16 ca sa se vada ca e cel mai lent si sa nu dau diagrama peste cap.

CLASAMENT 10/100 MILIOANE



*QuickSort are -1 pt ca fost **descalificat** deoarece nu a putut sorta toti vectorii. Ar fi avut timpi de ordinul **zecilor de minute**.

‘PROBA DE REZISTENTA’

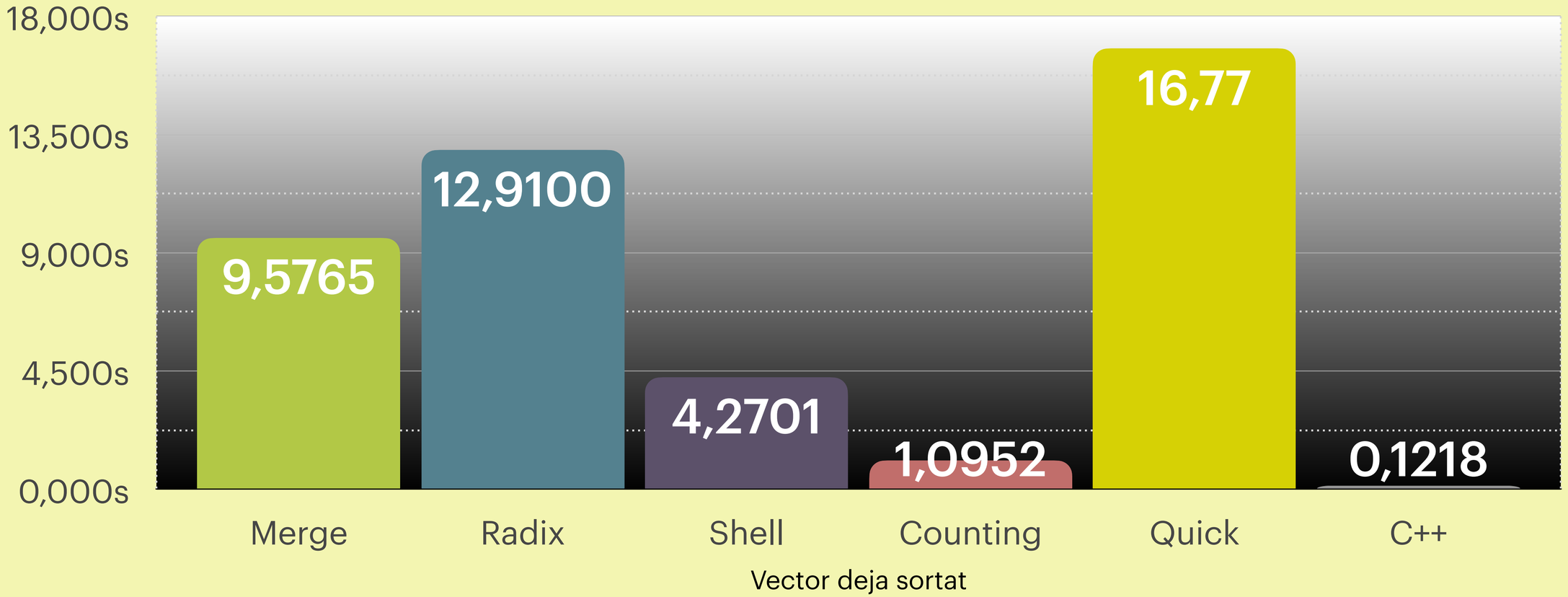
CE ESTE PROBA DE REZISTENTA?

Proba de rezistenta consta in sortarea unui vector deja sortat.

Am vazut ca pana in 10 milioane de numere, timpii sunt similari si in medie nu depasesc 1s. Prin urmare, nu o sa mai sortam 64 de vectori, ci o sa ducem testul de extrem.

Sortam un singur vector cu 100.000.000 de elemnte din intervalul [0, 100.000.000]. Haideti sa vedem cine casiga.

REZULTATE


18,000s
13,500s
9,000s
4,500s
0,000s
9,5765
12,9100
4,2701
1,0952
16,77
0,1218
Merge
Radix
Shell
Counting
Quick
C++
Vector deja sortat

THE END