STRUCTURI DE DATE

SORTĂRI

PREZENTARE

Nidelea Gabriela-Andreea
Grupa 132

TIPURI DE SORTĂRI

01

BubbleSort

02

MergeSort

03

QuickSort

04

CountSort

05

RadixSort

BubbleSort

Cea mai simplă metodă de sortare, realizându-se prin permutarea repetată a doua câte două elemente aflate în ordinea greșită.

avg:
53. 746 s

Timpi de rulare

Teste:

nrteste=7

n = 234	**maxi** = 78956
n = 2222	**maxi** = 78664
n = 34378	**maxi** = 90000
n =12300	**maxi** = 5677
n = 1000	**maxi** = 10000000
n = 167888	**maxi** = 1097654
n = 34	**maxi** = 999

rularea nr.1

T1= 0 s
T2=0.01 s
T3=3.091 s
T4=0.348 s
T5=0.002 s
T5=50.207 s
T6=0 s

timp total: 53,658 s

rularea nr. 2

T1= 0 s
T2=0.008 s
T3=3.028 s
T4=0.347 s
T5=0.002 s
T6=50.449 s
T7=0 s

timp total: 53, 834 s

MergeSort

MergeSort-ul împarte array-ul în jumătăți, până ajunge la un singur element în fiecare sub-array, urmând ca apoi sa le îmbine și să le sorteze, două cate două, la fiecare pas,

avg:
0.0475 s

Timpi de rulare

Teste:

nrteste=7

n = 234	**maxi** = 78956
n = 2222	**maxi** = 78664
n = 34378	**maxi** = 90000
n =12300	**maxi** = 5677
n = 1000	**maxi** = 10000000
n = 167888	**maxi** = 1097654
n = 34	**maxi** = 999

Rularea nr.1

T1= 0 s
T2=0 s
T3=0.006 s
T4=0.002 s
T5=0 s
T5=0.037 s
T6=0 s

timp total: 0.045 s

Rularea nr. 2

T1= 0 s
T2=0 s
T3=0.007 s
T4=0.002 s
T5=0 s
T6=0.041 s
T7=0 s

timp total: 0.05 s

QuickSort

QuickSort-ul este asemănător MergeSort-ului, folosindu-se de un algoritm asemănător de partiționare, însă de această dată, fiind introdus și un element de tip pivot (am ales metoda cu pivot = ultimul element). Se împarte array-ul dat în sub-array-uri, conținând elementele mai mici decât pivotul și, respectiv, elementele mai mari decât pivotul. Se reinițializează pivotul la următorul apel al funcției cu ultimul element al noului sub-array identificat.

avg:
0.164 s

Timpi de rulare

Teste:

nrteste=7

n = 234	**maxi** = 78956
n = 2222	**maxi** = 78664
n = 34378	**maxi** = 90000
n =12300	**maxi** = 5677
n = 1000	**maxi** = 10000000
n = 167888	**maxi** = 1097654
n = 34	**maxi** = 999

Rularea nr.1

T1= 0 s
T2=0 s
T3=0.006 s
T4=0.002 s
T5=0.001 s
T5=0.03 s
T6=0 s

timp total: 0.039 s

Rularea nr. 2

T1= 0 s
T2=0 s
T3=0.006 s
T4=0.002 s
T5=0 s
T6=0.28 s
T7=0.001 s

timp total: 0,289 s

CountSort

CountSort-ul folosește o metodă care se bazează pe identificarea maximului din array și creează un alt tablou de dimensiunea maximului plus unu. Tratează, acest nou tablou (inițializat cu zero) ca pe un tablou de frecvența, incrementând, acolo unde este cazul, numarul de apariții al elementului cu index-ul respectiv. Apoi, CountSort sortează vectorul inițial, decremetând cu 1 fiecare casuță a tabloului frecvența și introducând noua valoare în array-ul inițial.

avg:
0.023 s

Timpi de rulare

Teste:

nrteste=7

n = 234	**maxi** = 78956
n = 2222	**maxi** = 78664
n = 34378	**maxi** = 90000
n =12300	**maxi** = 5677
n = 1000	**maxi** = 10000000
n = 167888	**maxi** = 1097654
n = 34	**maxi** = 999

Rularea nr.1

T1=0.001 s
T2=0.001 s
T3=0.004 s
T4=0.001 s
T5=0 s
T5=0.011 s
T6=0 s

timp total: 0.018 s

Rularea nr. 2

T1=0 s
T2=0 s
T3=0.02 s
T4=0.001 s
T5=0 s
T6=0.007 s
T7=0 s

timp total: 0.028 s

RadixSort

RadixSort-ul este o metodă de sortare care se folosește următorul algoritm: identifică maximul din array, iar apoi implementează o metodă derivată din CountSort ("countingsort") pentru fiecare cifră din maxim. Sortează elementele după cifra unităților, apoi după cea a miilor, etc. Când maximul rămâne fără cifre de împrumutat, array-ul inițial va fi sortat,

avg:
0.0505 s

Timpi de rulare

Teste:

nrteste=7

n = 234	**maxi** = 78956
n = 2222	**maxi** = 78664
n = 34378	**maxi** = 90000
n =12300	**maxi** = 5677
n = 1000	**maxi** = 10000000
n = 167888	**maxi** = 1097654
n = 34	**maxi** = 999

Rularea nr.1

T1= 0 s
T2=0.001 s
T3=0.007 s
T4=0.002 s
T5=0 s
T5=0 s
T6=0.04 s
T7=0 s

timp total: 0.05 s

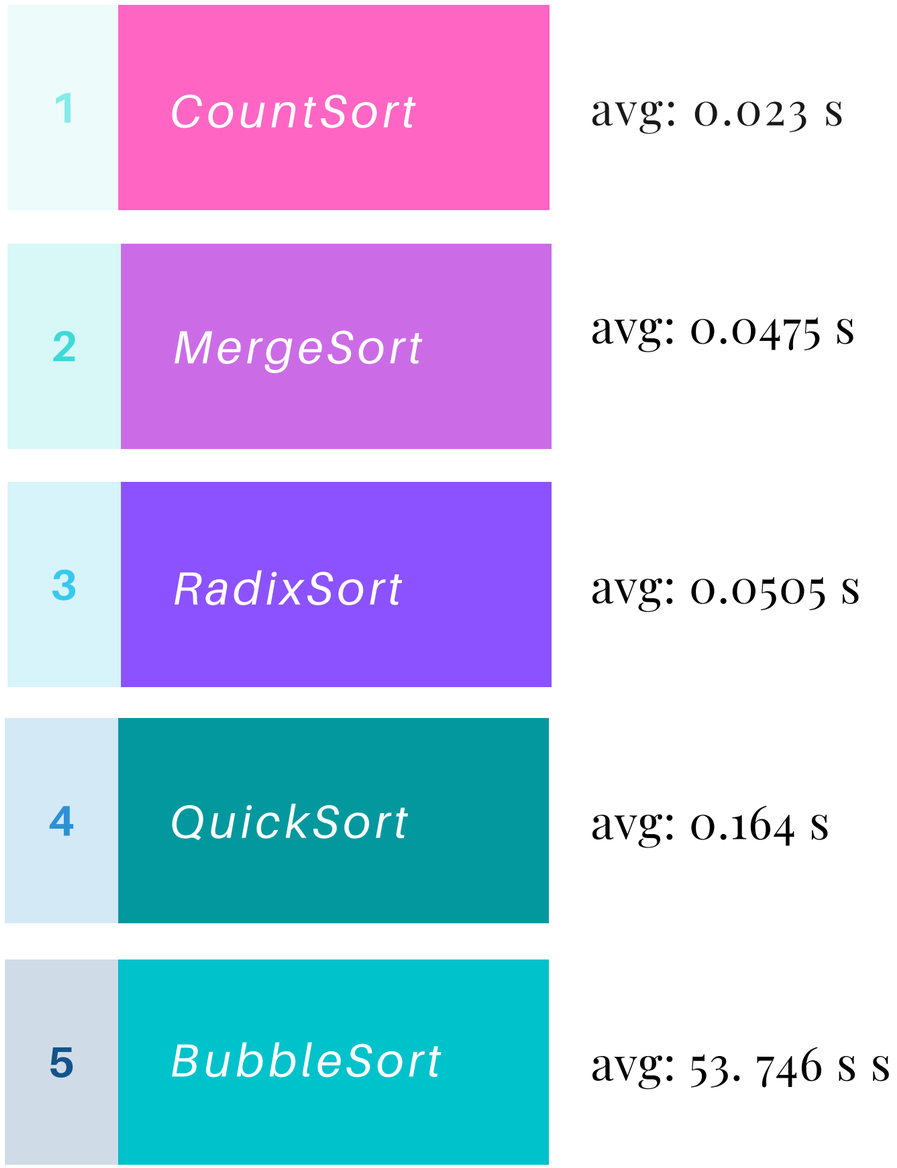
Rularea nr. 2

T1= 0.001 s
T2=0 s
T3=0.009 s
T4=0.002 s
T5=0 s
T6=0. 039 s
T7=0 s

timp total: 0.051 s

Clasament

(după timpul mediu de rulare)



Observatii



1. BubbleSort este o metoda foarte simplă de înțeles și de incrementat, însă destul de ineficientă, mai ales pentru numere mari, de tipul 10^8.

2. CountSort, deși folosește tablouri de frecvență, funcționează într-un interval de timp eficient, chiar și pentru numere mari.

3. RadixSort, QuickSort și MergeSort sunt metode în care regasim conceptul de recursivitate. Putem observa că timpul lor de execuție este asemănător (mai ales între MergeSort și RadixSort).

Sortarea nativă în Code::Blocks

avg:
0.055 s

Timpi de rulare

Teste:

nrteste=7

n = 234	**maxi** = 78956
n = 2222	**maxi** = 78664
n = 34378	**maxi** = 90000
n =12300	**maxi** = 5677
n = 1000	**maxi** = 10000000
n = 167888	**maxi** = 1097654
n = 34	**maxi** = 999

Rularea nr.1

T1=0 s
T2=0.001 s
T3=0.012 s
T4=0.002 s
T5=0 s
T6=0.052 s
T7=0 s

timp total: 0.067 s

Rularea nr. 2

T1= 0 s
T2=0 s
T3=0.006 s
T4=0.002 s
T5=0 s
T6=0. 035 s
T7=0 s

timp total: 0.043 s

Comparație între timpii rulare ai metodelor și cel nativ al sortării Code :: blocks

BubbleSort vs sort nativ

sort-ul nativ al Code::Blocks-ului este mai eficient d.p.d.v. al timpului

MergeSort vs sort nativ

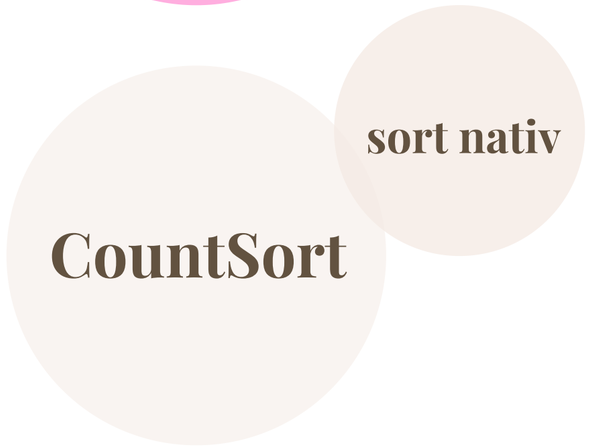
MergeSort-ul este mai eficient d.p.d.v. al timpului

RadixSort vs sort nativ

RadixSort este mai rapid d.p.d.v. al timpului

QuickSort vs sort nativ

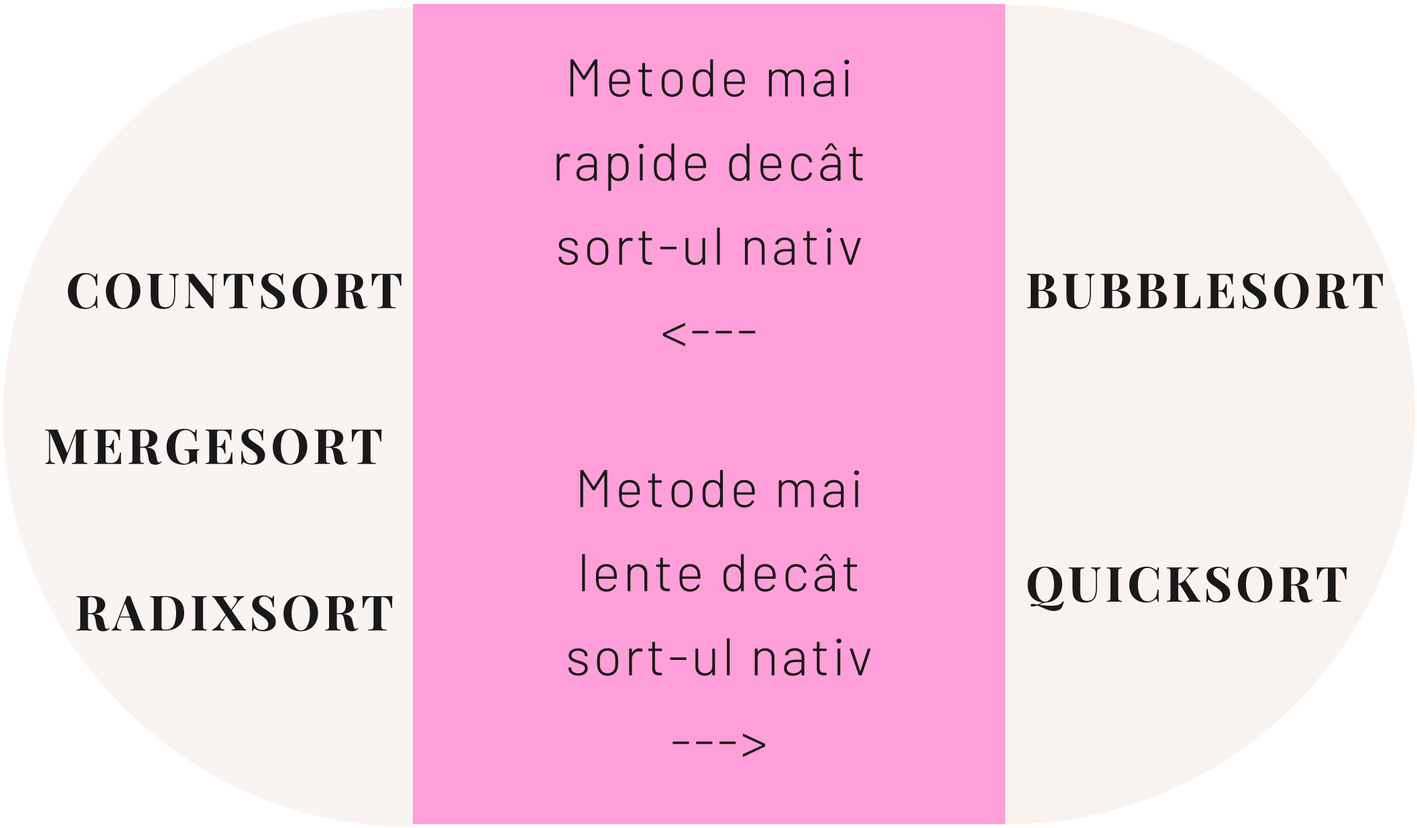
sort-ul nativ al Code::Blocks-ului este mai eficient d.p.d.v. al timpului



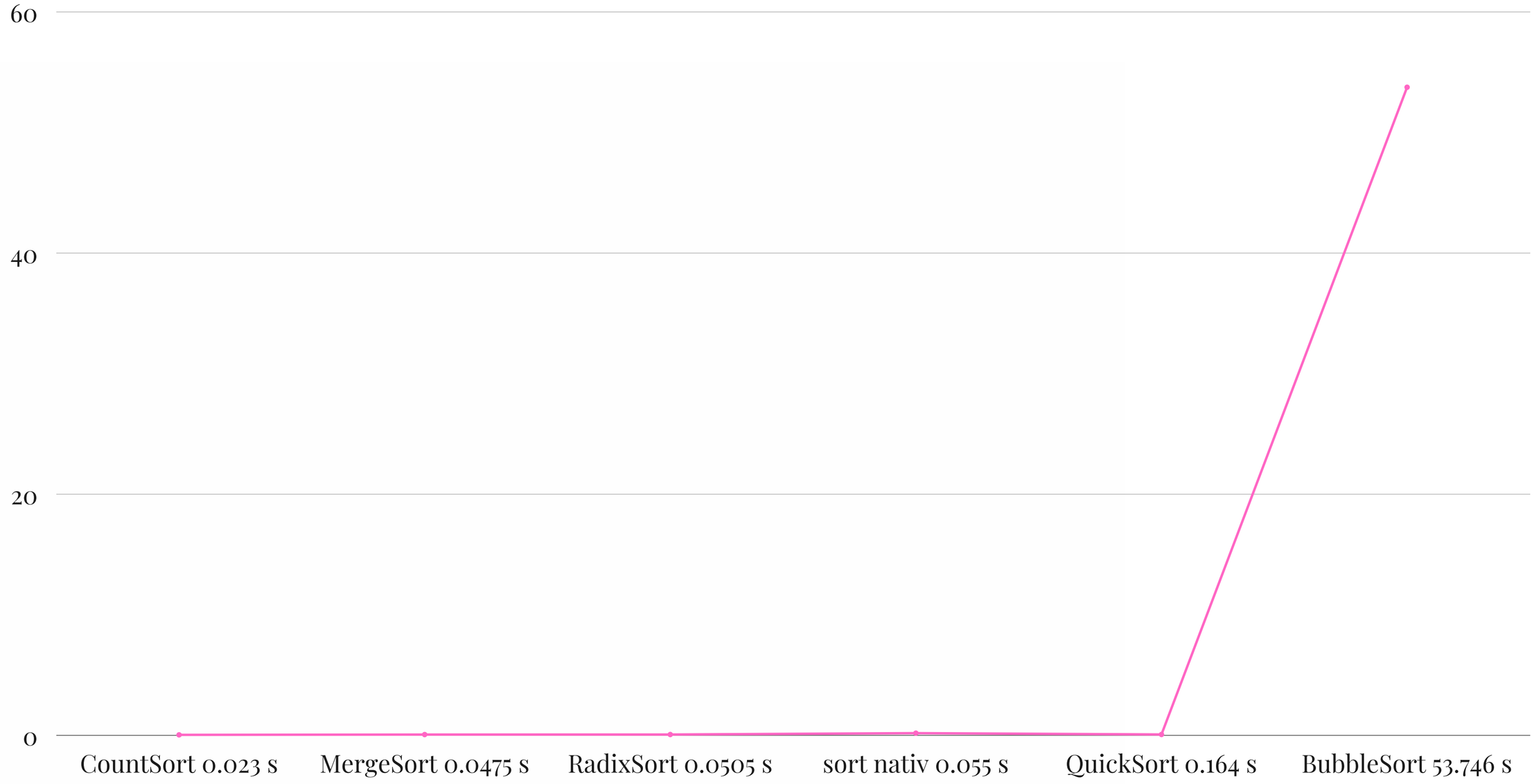
CountSort vs sort nativ

CountSort este mai rapid d.p.d.v. al timpului

DIAGRAMA METODE VS SORT NATIV



STATISTICA TIMPILOR DE RULARE



Scurtă analiză în %



Diferența de timp dintre CountSort și MergeSort este de 106.52%, adică metoda CountSort este de aproximativ două ori mai rapidă decât metoda MergeSort.

Diferența de timp dintre CountSort și BubbleSort este de 233678.26%, adică metoda BubbleSort este de aproximativ 2337 de ori mai lentă decât CountSort.

Final