Algoritmi de sortare

Florete Fabian-Andrei -134

Algoritmii aleși

- IntroSort
- MergeSort
- RadixSort(baza 256)
- HeapSort
- ShellSort

Metode de comparație:

În primul rând, fiecare algoritm trebuie să rezolve **20** de teste. Aceste teste au dimensiune variabilă a datelor, începând cu **10** numere și pană la **10^7** numere. Pe parcursul acestor teste apar și valori intermediare din intervale precum: (10^2- 10^3, 10^6, 10^7, etc.)

Fiecare test are N numere din intervalul [0, 2147483647)], 2147483647 fiind numarul natural cel mai mare care încape in int.

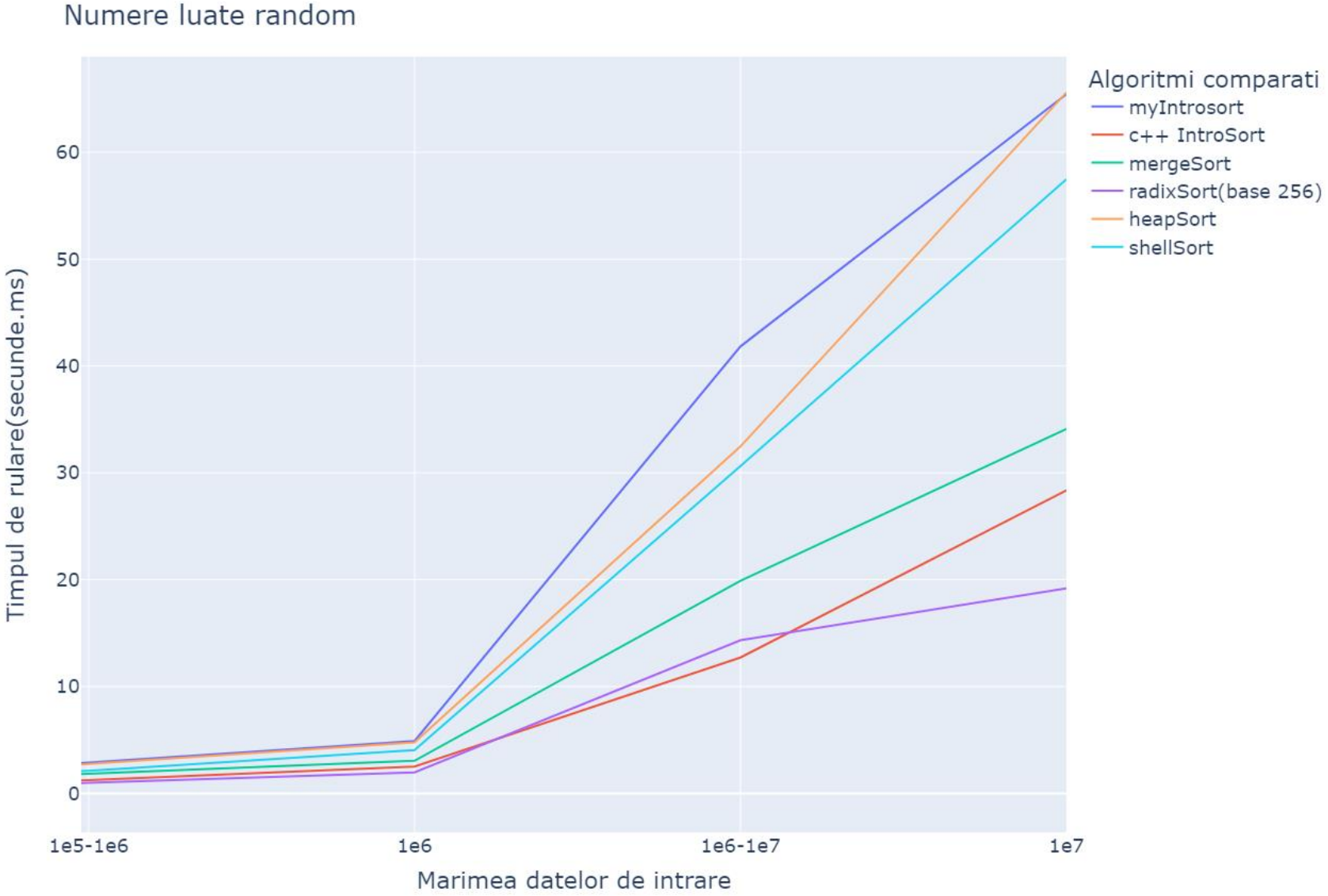
Termenul de comparație va fi timpul necesar sortării celor 20 de seturi de numere.

- Există 3 tipuri de teste:
  1. Numere random: cele N numere sunt selectate complet la întâmplare din intervalul posibil
  2. Numere apropiate: pentru fiecare număr din test, mai apar între 1 și 1000 numere apropiate de el (apropiat => Următorul_număr – Număr < 100)
  3. Numere care se repetă: pentru fiecare număr din test, mai apar între 0 și N numere identice după el

Comparația prin numere random: așteptări

- Timpul la mergesort să crească constant în raport cu dimensiunea numerelor
- Timpul la heap sort din ce in ce mai mare(datorita implementării cu copiere)

REZULTATE

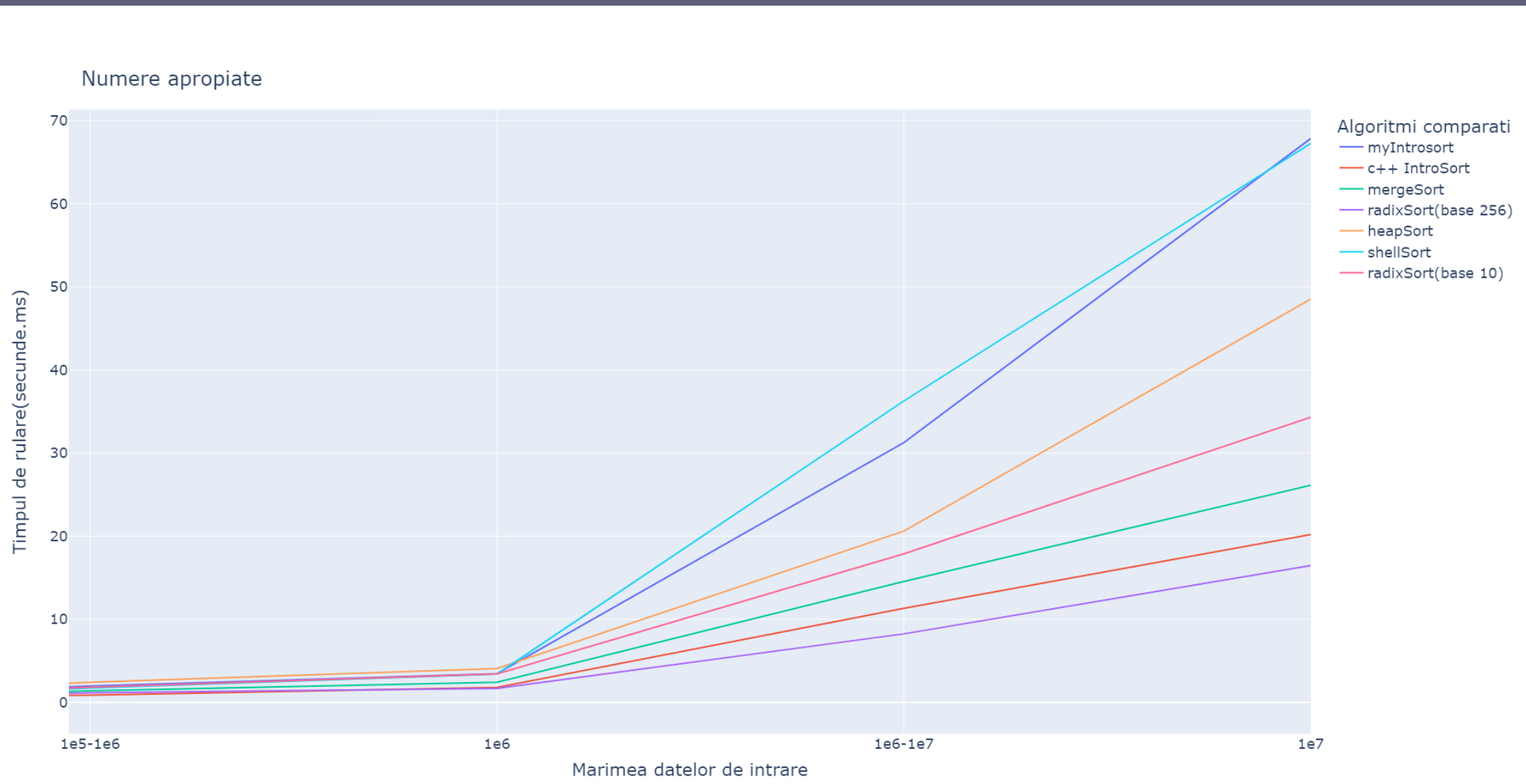

Numere luate random
Timpul de rulare(secunde.ms)
0
10
20
30
40
50
60
1e5-1e6
1e6
1e6-1e7
1e7
Marimea datelor de intrare
Algoritmi comparati
myIntrosort
c++ IntroSort
mergeSort
radixSort(base 256)
heapSort
shellSort

Observații și concluzii

- Pana la dimensiunea 1e6 nu exista diferențe majore între algoritmi
- Radix sort mai rapid decât intro sortul din c++ dupa dimensiunea 1e6-1e7
- Deși implementarea myIntrosort folosește același heapSort ca și heapSort-ul prezentat, timpul este mai mare, datorita copierii datelor, alegerii pivotului și parametrizare diferită în decizia utilizării heapSort vs insertionSort

Comparația prin numere apropiate: așteptări

- Radix sort in baza 10 mai încet, deoarece bucketurile in care se impart numerele nu vor fi atât de semnificative

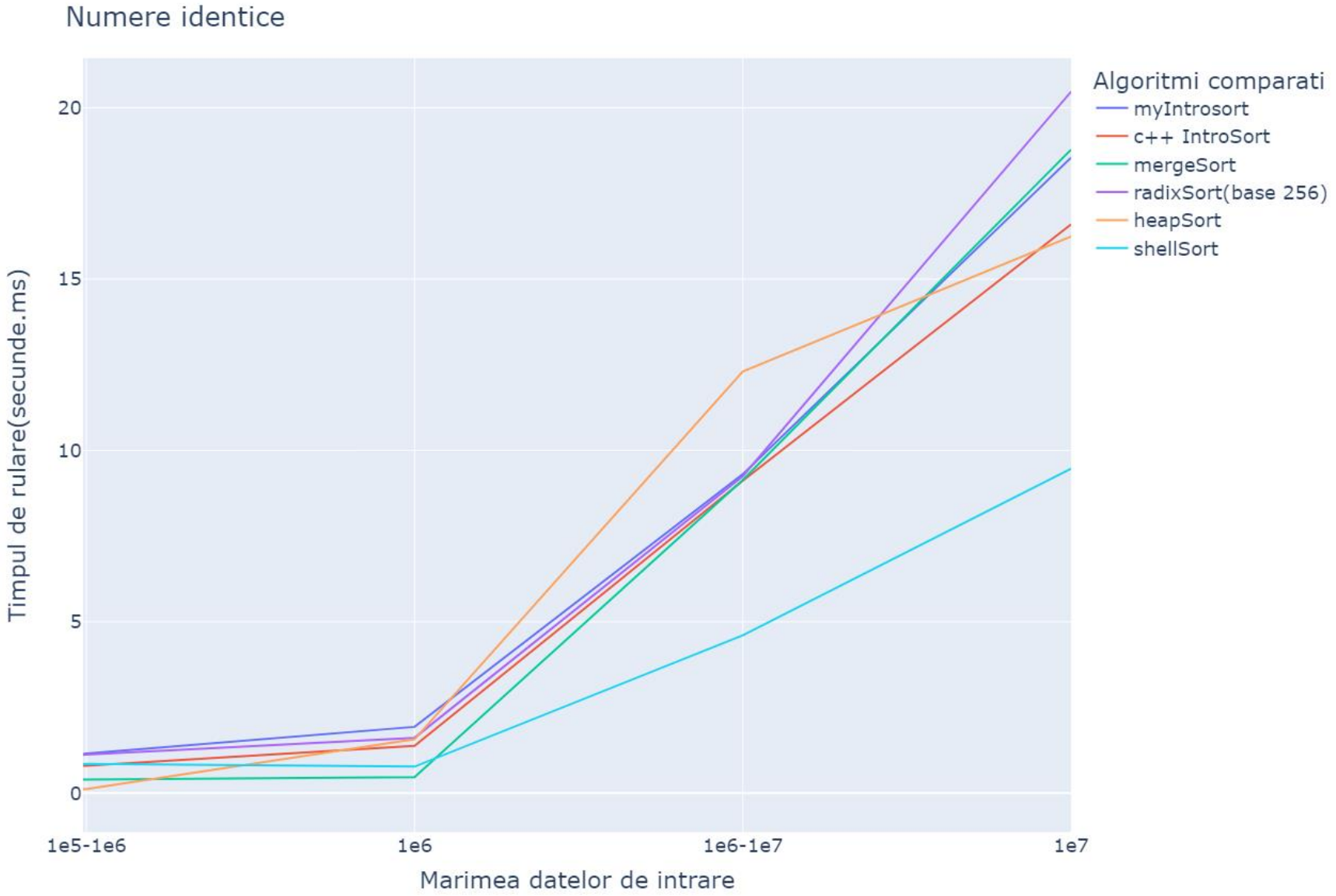

Numere apropiate
Timpul de rulare(secunde.ms)
70
60
50
40
30
20
10
0
1e5-1e6
1e6
1e6-1e7
1e7
Marimea datelor de intrare
Algoritmi comparati
myIntrosort
c++ IntroSort
mergeSort
radixSort(base 256)
heapSort
shellSort
radixSort(base 10)

Observații și concluzii

- RadixSort(base 10) mai incet decat c++ introSort, dar c++ introSort mai incet decat RadixSort(base 256)
- Shell sort mai incet chiar și decât myIntrosort
- Diferență mai mare dintre heapSort si myIntroSort => s-a folosit mai puțin heapSort ul în sortarea prin myIntroSort

Comparația prin numere identice: așteptări

- Radix sort mult mai incet(nu mai are criterii de pus in buket)
- ShellSort mai rapid, nu este nevoit sa faca multe interschimbări
- HeapSort încet,


Numere identice
Timpul de rulare(secunde.ms)
0
5
10
15
20
1e5-1e6
1e6
1e6-1e7
1e7
Marimea datelor de intrare
Algoritmi comparati
myIntrosort
c++ IntroSort
mergeSort
radixSort(base 256)
heapSort
shellSort

Observații și concluzii

- Overall, toți timpii sunt reduși
- Radix sort este cel mai lent
- Shell sort rapid, puține swap-uri
- Heap sort deși încet pe puține numere, rapid pe multe numere(O data format un heap, extragerea din el este mul mai rapidă, deoarece multe numere sunt egale)