Balan Paula
Grupa 132

Algoritmi de sortare

1. Prezentarea algoritmilor

1. Bubble Sort

Algoritmul de sortare Bubble Sort este un algoritm simplu de sortare de dimensiuni mici, însă nu este un algoritm practic deoarece este lent. Bubble Sort muta elementele până se află în ordine, elentele maxime fiind sortate primele.

Acest algoritm funcționează pe toate tipurile de intrare (int, float, string etc), are o complexitate medie de O(n^2) și este stabil.

2. Counting Sort

Algoritmul de sortare Counting Sort este un algoritm rapid de numărare. Deși este un algoritm rapid, acesta depinde de dimensiunea elementelor, deoarece poate ajunge să acopere un spațiu de memorie prea mare. Este utilizat în special pentru sortarea unor elemente de dimensiuni reduse.

Counting Sort numără aparițiile celor n elemente din șirul L. Se creează un vector de frecvență unde se memorează numărul de apariții a index-ului elementului, după care se rescrie șirul ordonat.

Acest algoritm nu funcționează pe toate tipurile de intrare, are o complexitate medie de O($n + m$) (n-numarul elementelor; m-elementul maxim) și este stabil.

3. Radix Sort

Radix Sort este un algoritm de sortare non-comparativ. Algoritmul sortează fiecare element de la cifra unităților spre stânga (zeci, sute, etc), folosind algoritmul counting sort pentru a sorta

elementele în funcție de cifră . Există multipli algoritmi de Radix (pentru Baza 10, Baza 2, Baza 4 etc).

Acest algoritm pote funcționa pe toate tipurile de intrare, însă trebuie proiectați algoritmi diferiți pentru fiecare. Este greu de calculat complexitatea și nu este stabil.

4. Merge Sort

Algoritmul de sortare

Merge Sort se bazeaza pe tehnica de programare “Divide et impera”. Împarte șirul în două jumătăți și combină sortat cele două subșiruri, care la rândul lor apelează algoritmul Merge Sort.

Acest algoritm funcționează pe toate tipurile de intrare (int, float, string etc), are o complexitate medie de O($n \log n$) și este stabil.

5. Quick Sort

Quick Sort este un algoritm de sortare care, pentru un șir de n elemente, are un timp de execuție de O(n^2), în cazul cel mai defavorabil în condițiile în care elementul pivot se află la începutul/sfârșitul sirului.

Aloritmul de sortare rapidă se bazeaza pe tehnica de programare “Divide et impera”. Șirul L[l…r] este împartit în două subșiruri nevide L[l..p], L[p+1…r] astfel încât fiecare element al subșirului L[l...p] sa fie mai mic sau egal cu orice element al subșirului L[p+1…r]. p-ul reprezintă indicele elementului “pivot”, în jurul căruia se face partiționarea. Cele două subșiruri L[l...p] și L[p+1...r] sunt sortate prin apeluri recursive ale algoritmului de sortare rapida.

Acest algoritm funcționează pe toate tipurile de intrare (int, float, string etc), are o complexitate medie de O($n \log n$), dar nu este stabil.

2. Prezentarea programului

Am început programul prin implemantarea algoritmilor de sortare. Pentru algortitmul Radix Sort am implementat două subfuncții una pentru numere (Radix_digit) și una pentru cuvinte (Radix_word), iar în funcția principală (RadixSort) am verificat în plus tipul elementelor din listă.

După finalizarea implementării algoritmilor de sortare am construit funcții care să creeze automat liste de diferite dimensiuni și diferite tipuri: int (Random_int), float (Random_float) și str (Random_str). Pentru funcțiile care returnează elemente cifre pe lângă dimensiunea listei se declară și lungimea numerelor.

Funcția Timer calculează și afișează pentru fiecare algoritm de sortare timpul de executare. În antetul funcției sunt trecuți trei parametri l – lista care trebuie sortată, p – un număr ce reprezintă tipul elementelor (0 – int, 1 – float, 2 – str) Variabila p activează/dezactivează funcțiile CountSort și RadixSort. CountSort va fi rulat atunci când p = 0, RadixSort nu va fi rulat atunci când sunt elemente de tip float (p = 1).

Funcția Test generează automat un număr de teste. Se vor realiza teste pe liste de lungimi mici $[10^4 , 10^6]$ și teste pe liste de lungimi mari $[10^6, 10^8]$ astfel: Setăm variabila sz = 1_000 și rulăm random 2 sau 3 tipuri de teste pe numere naturale și reale, după setăm sz = 10_000_000 și repetăm testele pe numere. În cele din urmă se vor face 2 sau 3 tipuri de teste pe liste cu cuvinte.

Funcția Test1 genereaza automat 2 sau 3 teste pentru liste aproape ordonate, respectiv liste ordonate descrescător.

Cele 2 funcții Test fac apel la funcția Timer.

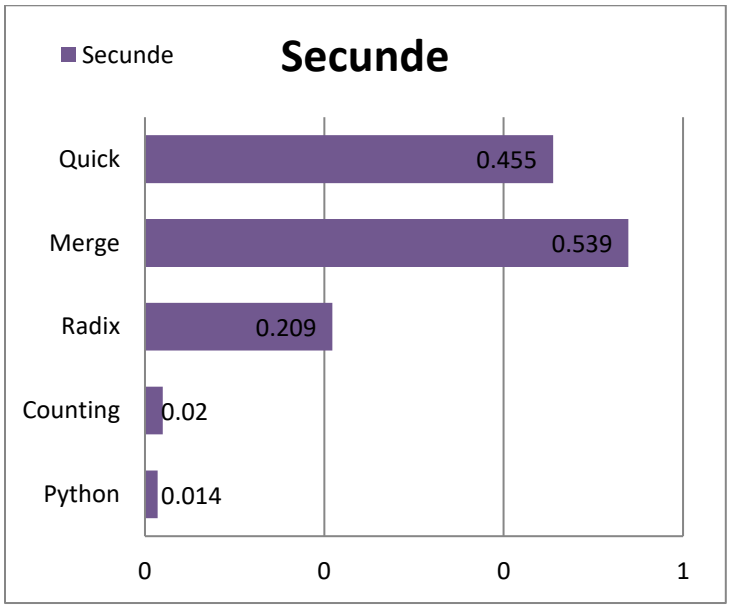
3. Analizarea algoritmilor

- Compatibilitatea sortarilor cu tipuri de date

Sort\Type	int	float	str
Bubble	✓	✓	✓
Counting	✓	✗	✗
Radix	✓	✗	✓
Merge	✓	✓	✓
Quick	✓	✓	✓

- Viteza de rulare a sortarilor pe liste de lungime n^6

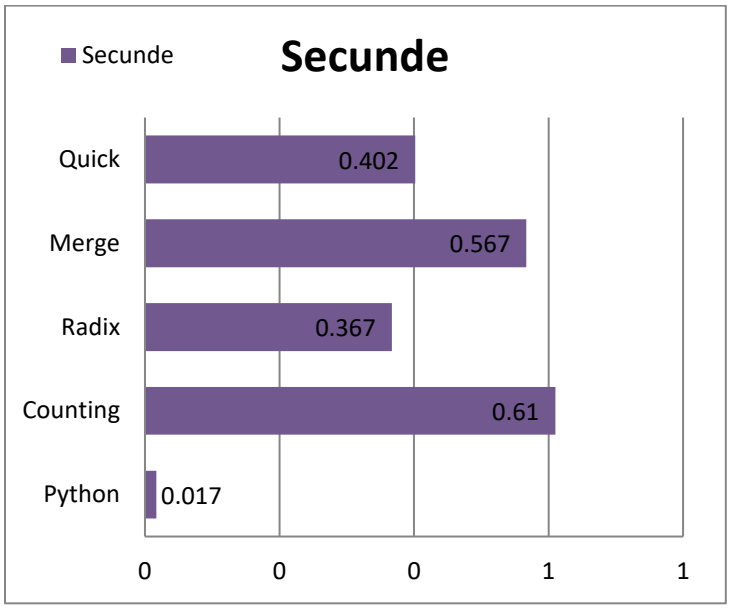
  ➢ Numere naturale de dimensiuni mici $[\ 0, n^4]$



Python Sort	0.014
Bubble Sort	862.944
Counting Sort	0.020
Radix Sort	0.209
Merge Sort	0.539
Quick Sort	0.455

*Bubble Sort nu va fost introdusă în nici un grafic, având un timp de rulare considerabil de mare.

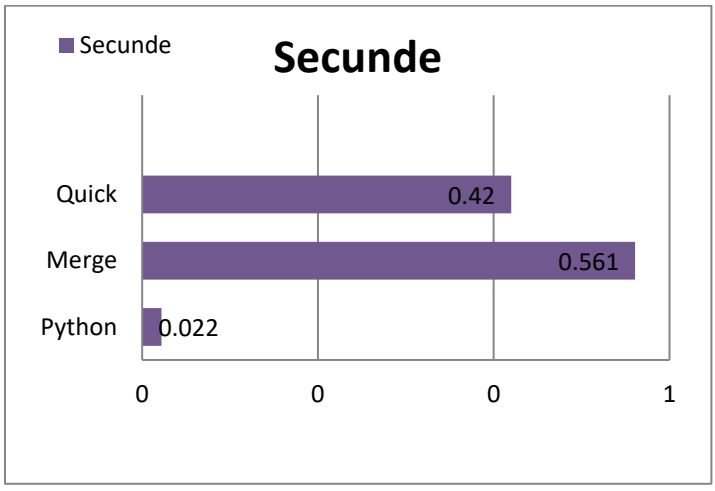
- Numere naturale de dimensiuni mari $[0, n^8]$



Python Sort	0.017
Bubble Sort	880.110
Counting Sort	0.610
Radix Sort	0.367
Merge Sort	0.567
Quick Sort	0.402

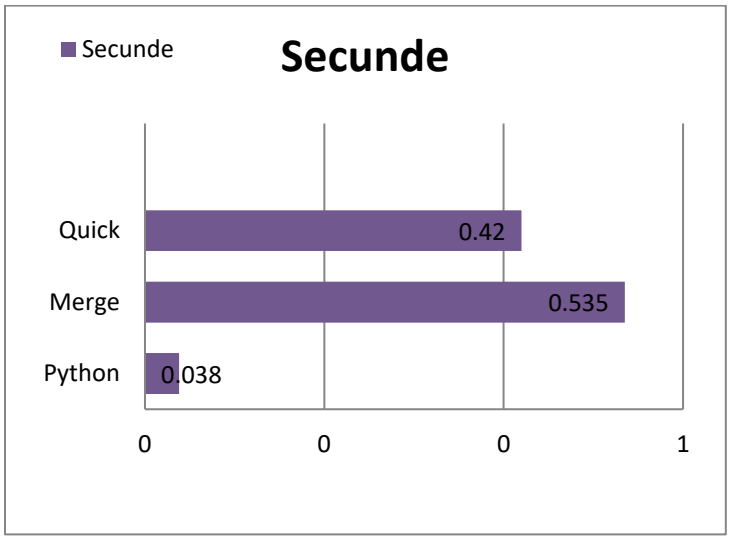
Deși majoritatea algoritmilor au avut o creștere a timpului mică, Counting Sort a avut o creștere de 3 ori mai lentă, devenind de la cel mai rapid la cel mai lent algoritm de sortare din listă (după Bubble Sort).

- Numere reale de dimensiuni mici$[0, n^4]$



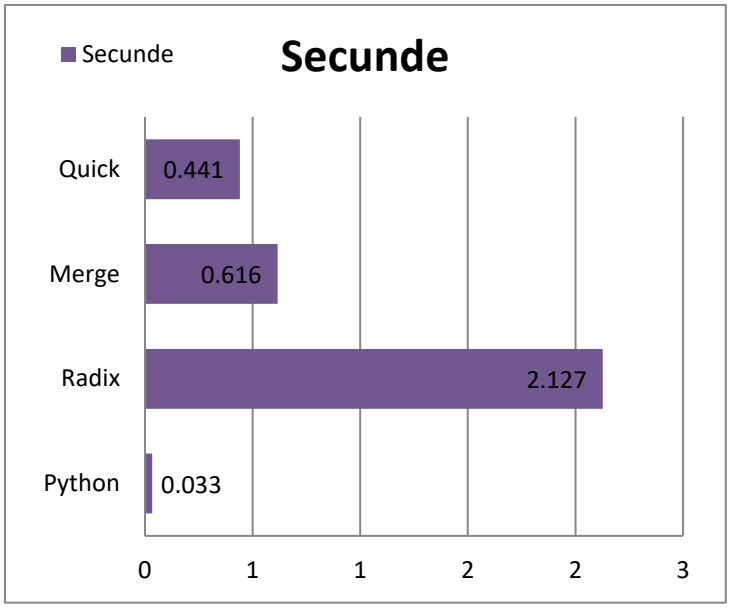
Python Sort	0.022
Bubble Sort	939.491
Merge Sort	0.561
Quick Sort	0.420

- Numere reale de dimensiuni mari $[0, n^8]$



Python Sort	0.038
Bubble Sort	941.489
Merge Sort	0.535
Quick Sort	0.420

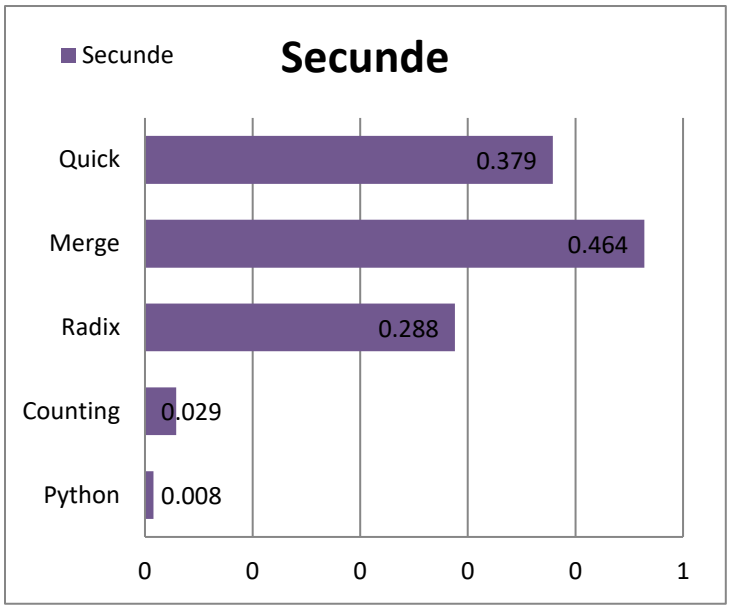
- Șiruri de caractere



Python Sort	0.033
Bubble Sort	1102.144
Radix Sort	2.127
Merge Sort	0.616
Quick Sort	0.441

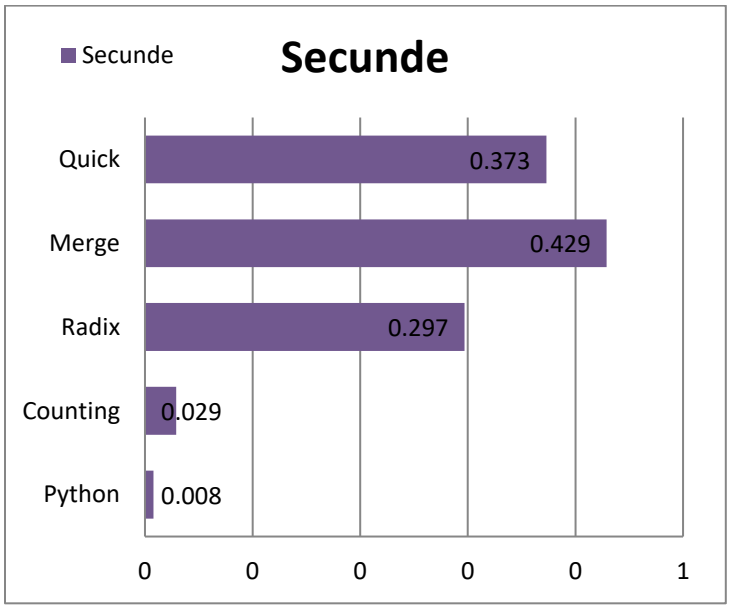
Timpul de rulare al algoritmului Radix Sort este datorat construcției lui. Am ales un caz general, unde sirurile de caractere pot avea atât litere mari, cât și mici, astfel Radix are de căutat într-un vector de frecvență cu 52 de elemente, lucru ce îi încetinește viteza de rulare.

- Viteza de rulare a sortarilor pe liste aproape sortate



Python Sort	0.008
Bubble Sort	437.069
Counting Sort	0.029
Radix Sort	0.288
Merge Sort	0.464
Quick Sort	0.379

- Viteza de rulare a sortarilor pe liste sortate descredcator



Python Sort	0.008
Bubble Sort	1246.979
Counting Sort	0.029
Radix Sort	0.297
Merge Sort	0.429
Quick Sort	0.373

Ultimele două tabele arată faptul că timpul de sortare nu este afectat de tipul listei, în afară de algoritmul Bubble Sort care are o diferentă de 3 ori între o listă aproape sortată și o listă sortată descrescător.