

PROIECT SORTĂRI

Sora Andreea-Ioana

Seria 13

Grupa 134

*Structuri de Date – Lect. Dr. Marius Dumitran
Facultatea de Matematică și Informatică
Universitatea din București*

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe



Nume: Sora Andreea-Ioana; Grupa: 134 - Proiect sortari

-----Metode de sortare:-----

- 1.BubbleSort
- 2.CountSort
- 3.MergeSort
- 4.InsertionSort
- 5.RadixSort-LSD
- 6.QuickSort-pivot ultimul element
- 7.QuickSort-pivot mediana din 3
- 8.Sortare nativa a limbajului C++

-----Tipuri de teste:-----

- 1.Vector aproape sortat crescator
- 2.Vector aproape sortat descrescator
- 3.Vector complet random
- 4.Vector constant (format doar din elementul maxim)
- 5.Vector cu elemente double

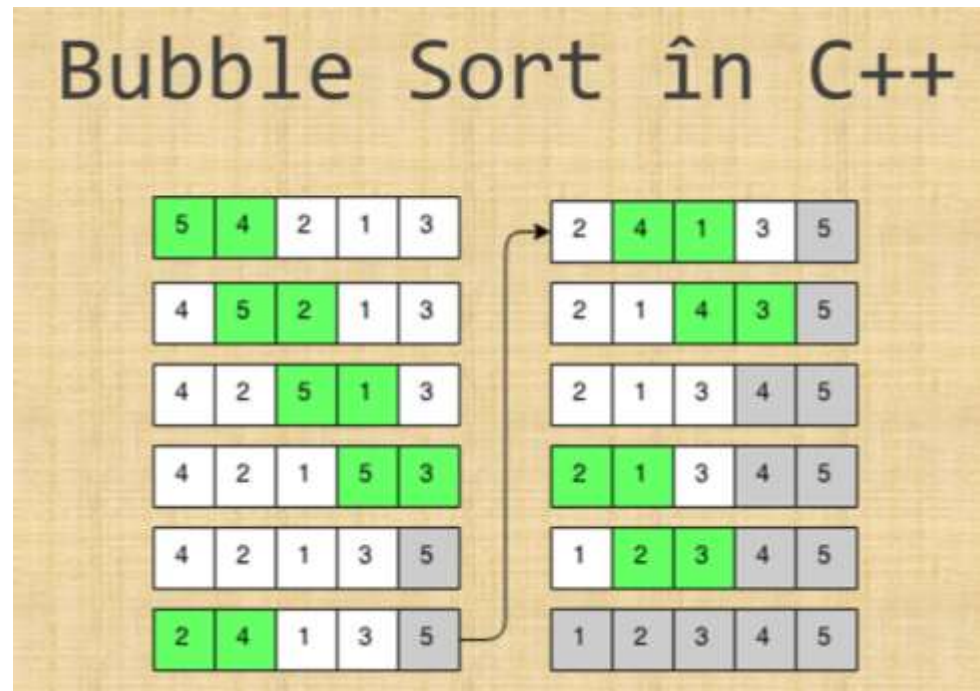
Introduceti numarul tipului de test:

BubbleSort

Complexitate: $O(n^2)$

Idee:

Parcurgem vectorul, iar pentru oricare două elemente vecine care nu sunt în ordinea dorită, le interschimbăm valorile.



CountSort

Complexitate:

-timp: $O(n+\max)$

-spațiu: $O(\max)$

Idee:

Parcurgem vectorul și reținem frecvențele valorilor într-un vector freqv. După acest pas, parcurgem freqv de la 0 la max, iar pentru fiecare element freqv[i] întâlnit, adăugăm în vectorul rezultat i de freqv[i] ori.

Input Data

0	4	2	2	0	0	1	1	0	1	0	2	4	2
---	---	---	---	---	---	---	---	---	---	---	---	---	---

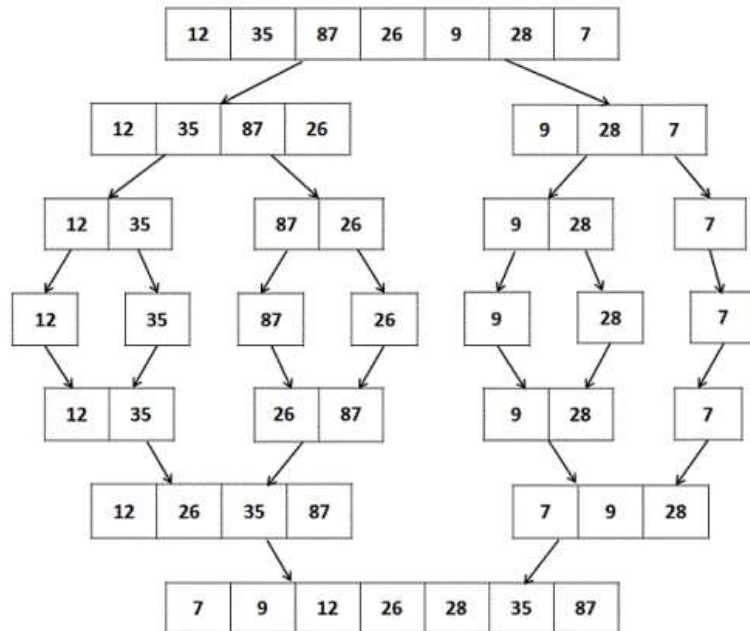
Count Array

0	1	2	3	4
5	3	4	0	2

Sorted Data

0	0	0	0	0	1	1	1	2	2	2	2	4	4
---	---	---	---	---	---	---	---	---	---	---	---	---	---

MergeSort



10/20/2016

Merge Sort

Complexitate: $O(n \log n)$

Idee:

Se împarte vectorul în jumătate și se interclasează independent fiecare parte formându-se un subșir ordonat din vector, mai mare, care, la rândul lui se va interclasa cu subșirul corespunzător obținut în același mod etc.

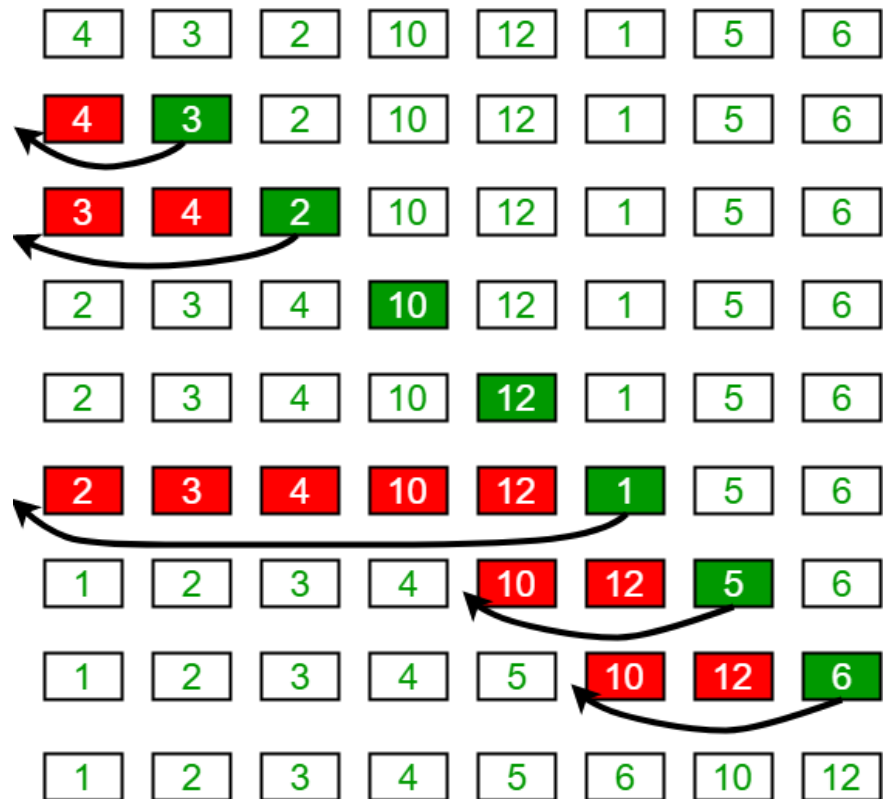
InsertionSort

Complexitate: $O(n^2)$

Idee:

Parcurgem vectorul și inserăm elementul $v[i]$ în secvența din stânga sa, elementele mutându-se spre dreapta.

Insertion Sort Execution Example



RadixSort – Varianta LSD (Least significant digit)

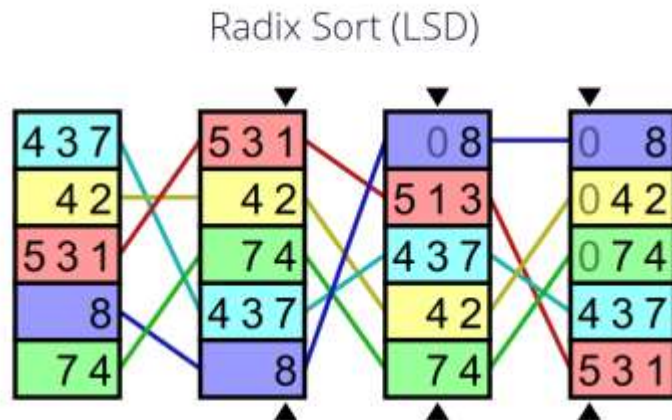
Complexitate:

-timp: $O(nk)$; k -lungimea medie a fiecărei chei;

-spațiu: $O(n+b)$

Idee:

Se grupează elementele pe rând după cea mai nesemnificativă cifră (de la coadă la cap), urmând ca la final vectorul să fie sortat.



Growing with the Web

QuickSort

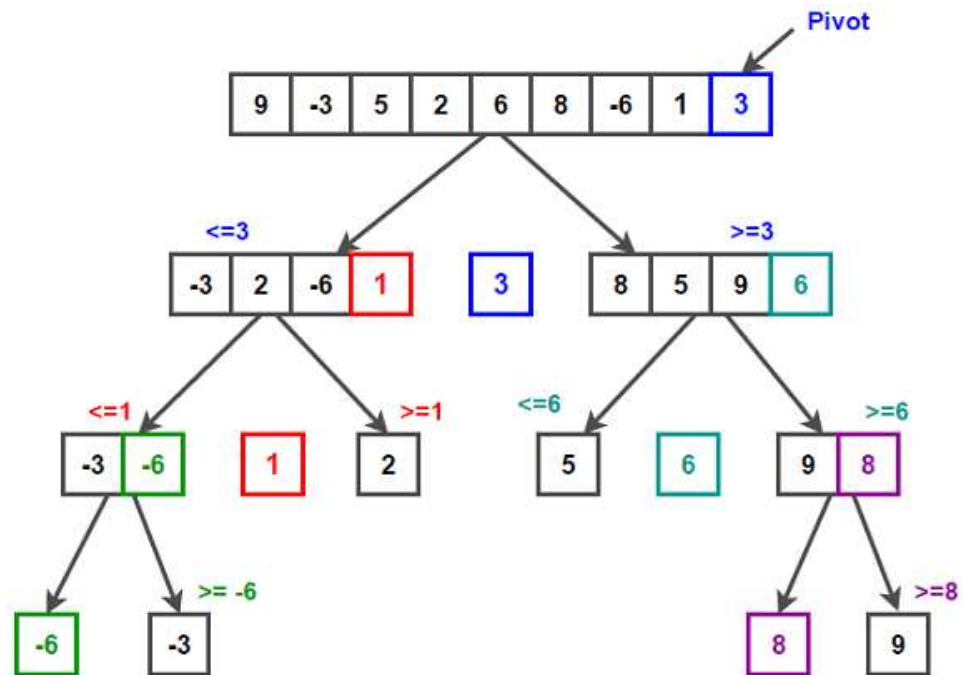
Complexitate: $O(n^2)$ (worst case) / $O(n \log n)$ (best case)

Idee:

Se alege un element drept pivot, iar subvectorul din stânga va fi format din elementele mai mici decât pivotul, iar în dreapta mai mari decât pivotul.

Pivotul poate fi ales ca fiind:

- ultimul/primul sau elementul din mijloc (cazuri nu tocmai bune);
- un element random;
- mediana din 3 (caz favorabil).

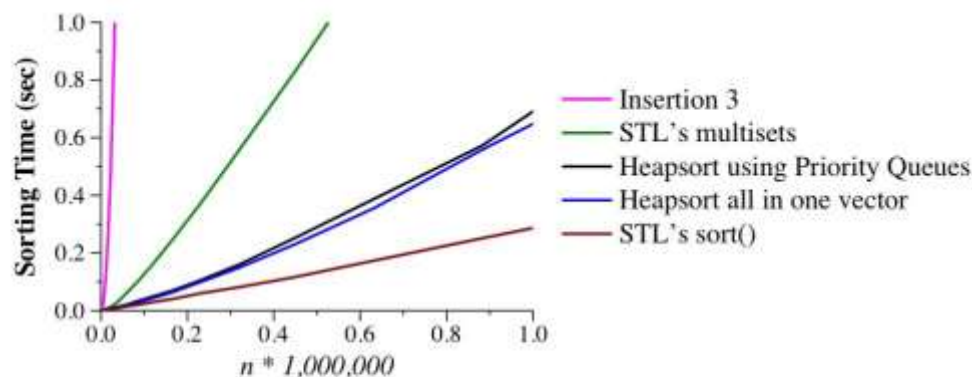


Sortarea nativă a limbajului C++ (sort)

Funcția sort este declarată în header-ul algorithm, ea folosind un hibrid între QuickSort, HeapSort și InsertionSort.



STL sorting in C++



Tipuri de teste:

- ✓ Vector aproape sortat crescător;
- ✓ Vector aproape sortat descrescător;
- ✓ Vector random;
- ✓ Vector constant (din elementul maxim);
- ✓ Vector de double.

În continuare vom analiza timpul de rulare pentru fiecare algoritm, mai în detaliu pentru un vector random.

Observație: Timpii nu vor fi întotdeauna la fel, acest lucru va fi influențat și de forma vectorului generat!!

Lungime=10/Maxim=10

```
C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe
Introduceti numarul de elemente din vector: 10
Introduceti elementul maxim(possibil) din vector: 10
-----
Timp sortare bubblesort:0.000001300 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.000000700 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.000001400 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:0.000000600 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.000003800 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.000001200 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.000001300 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.000002100 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:
```

Pentru început, toți algoritmi se mișcă foarte rapid, însă evident putem observa diferențe între CountSort/InsertionSort și celelalte metode de sortare. Pentru un algoritm de complexitate $O(n^2)$, InsertionSort prezintă un timp foarte bun pentru un număr așa mic de elemente.

Lungime=10/Maxim=1000000

C:\Users\Andreea\Desktop\codeblocks\proiect-sd\bin\Debug\proiect-sd.exe

— □ ×

```
Introduceti numarul de elemente din vector: 10
Introduceti elementul maxim(posibil) din vector: 1000000
-----
Timp sortare bubblesort:0.000001300 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.007461900 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.000001600 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:0.000000500 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.000007600 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.000001500 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.000001300 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.000002100 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:
```

Acest test evidențiază faptul că, în ciuda numărului mic de elemente, dacă maximul este un număr mare, CountSort își crește foarte mult timpul de executare în comparație cu vectorul de 10 elemente (10 maximul). Ceilalți algoritmi au un timp similar.

Lungime=100/Maxim=100

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe

```
Introduceti numarul de elemente din vector: 100
Introduceti elementul maxim(possibil) din vector: 100
-----
Timp sortare bubblesort:0.000100400 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.000003700 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.000013800 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:0.000017600 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.000019300 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.000017700 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.000016800 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.000017500 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:
```

Se observă o creștere destul de bruscă între BubbleSort pe 10 elemente și pe 100 de elemente. Începe să se stabilizeze și diferența între RadixSort și QuickSort. CountSort pare să fie cel mai rapid, prezentând un timp foarte bun.

Lungime=1000/Maxim=1000

```
C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe
Introduceti numarul de elemente din vector: 1000
Introduceti elementul maxim(posibil) din vector: 1000
-----
Timp sortare bubblesort:0.007108500 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.000024300 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.000166800 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:0.001115900 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.000144700 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.000236600 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.000177900 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.000249900 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar: _
```

În continuare are loc o creștere pentru algoritmi cu complexitatea $O(n^2)$.

Lungime=10000/Maxim=10000

```
C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe
Introduceti numarul de elemente din vector: 10000
Introduceti elementul maxim(posibil) din vector: 10000
-----
Timp sortare bubblesort:0.767566700 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.000261400 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.002105800 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:0.098647100 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.001691900 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.002366600 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.002277700 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.002286800 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar: _
```

Analog cu slide-ul precedent.

Lungime=100000/Maxim=100000

```
C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe
Introduceti numarul de elemente din vector: 100000
Introduceti elementul maxim(posibil) din vector: 100000
-----
Timp sortare bubblesort:77.721121100 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.002658200 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.025808700 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:9.776032600 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.018984400 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.027143500 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.026687000 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.026430000 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar: _
```

La acest număr de elemente, consider că devin total ineficiente BubbleSort și InsertionSort, sortarea luându-le câteva secunde. CountSort rămâne fruntaș în clasament.

Lungime=1000000/Maxim=1000000

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe

— □ ×

```
-----
Introduceti numarul tipului de test: 3
Introduceti numarul de elemente din vector: 1000000
Introduceti elementul maxim(posibil) din vector: 1000000
-----
Bubblesort ar dura prea mult pe calculatorul meu pentru acest numar de elemente!
-----
Timp sortare countsort:0.038556300 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.307666400 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Insertionsort ar dura prea mult pe calculatorul meu pentru acest numar de elemente!
-----
Timp sortare radixsort_lsd:0.189566400 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.321107300 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.318667400 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.302047100 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:
```

Pentru un milion de elemente, soluțiile optime rămân RadixSort și CountSort. Toate celelalte sortări sunt oarecum similare în timpul de executare.

Lungime=10000000/Maxim=10000000

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe

```
-----
Introduceti numarul tipului de test: 3
Introduceti numarul de elemente din vector: 10000000
Introduceti elementul maxim(posibil) din vector: 10000000
-----
Bubblesort ar dura prea mult pe calculatorul meu pentru acest numar de elemente!
-----
Timp sortare countsort:0.465392300 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:3.561609900 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Insertionsort ar dura prea mult pe calculatorul meu pentru acest numar de elemente!
-----
Timp sortare radixsort_lsd:2.519031600 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:3.814195100 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:3.684512600 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:3.483786500 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:
```

La acest pas, CountSort devine cel mai eficient, având un timp de mai puțin de jumătate de secundă, în comparație cu toți ceilalți care trec de două secunde.

Lungime=10^8/Maxim=10000000

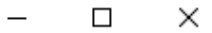
```
C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe

-----
Introduceti numarul tipului de test: 3
Introduceti numarul de elemente din vector: 100000000
Introduceti elementul maxim(possibil) din vector: 1000000
-----
Bubblesort ar dura prea mult pe calculatorul meu pentru acest numar de elemente!
-----
Timp sortare countsort:2.008656000 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:37.807919200 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Insertionsort ar dura prea mult pe calculatorul meu pentru acest numar de elemente!
-----
Timp sortare radixsort_lsd:22.033698000 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:51.197158600 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:50.739668100 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:34.850960700 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----

Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar: _
```

În concluzie, pentru 10^8 elemente toți algoritmi ating un interval de timp destul de ridicat, CountSort fiind de departe cel mai rapid.

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe



Nume: Sora Andreea-Ioana; Grupa: 134 - Proiect sortari

-----Metode de sortare:-----

- 1.BubbleSort
- 2.CountSort
- 3.MergeSort
- 4.InsertionSort
- 5.RadixSort-LSD
- 6.QuickSort-pivot ultimul element
- 7.QuickSort-pivot mediana din 3
- 8.Sortare nativa a limbajului C++

-----Tipuri de teste:-----

- 1.Vector aproape sortat crescator
- 2.Vector aproape sortat descrescator
- 3.Vector complet random
- 4.Vector constant (format doar din elementul maxim)
- 5.Vector cu elemente double

Introduceti numarul tipului de test: 3

Introduceti numarul de elemente din vector: 1000000000

Introduceti elementul maxim(possibil) din vector: 100000

Introduceti un numar mai mic de elemente(maxim $2 \cdot (10^8)!!$)

Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:

Alte tipuri de teste: double

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe

```
4.InsertionSort
5.RadixSort-LSD
6.QuickSort-pivot ultimul element
7.QuickSort-pivot mediana din 3
8.Sortare nativa a limbajului C++
-----Tipuri de teste:-----
1.Vector aproape sortat crescator
2.Vector aproape sortat descrescator
3.Vector complet random
4.Vector constant (format doar din elementul maxim)
5.Vector cu elemente double
-----
Introduceti numarul tipului de test: 5
Introduceti numarul de elemente din vector: 100000
Introduceti elementul maxim(possibil) din vector: 100000
-----
Timp sortare bubblesort:80.018182700 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Countsot nu ar avea sens pe elemente double (formarea vectorului de frecventa)!
-----
Timp sortare mergesort:0.026977900 secunde
Vectorul a fost sortat corect folosind mergesort !
-----
Timp sortare quicksort:0.026102400 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar: _
```

Aşa cum şi pe elemente numere naturale BubbleSort devenea ineficientă de la pragul de 10^5 în sus inclusiv, acest lucru este valabil şi pentru vectorul de double. Celelalte sunt similare cu vectorul de numere naturale.

Alte tipuri de teste: vector constant

```
C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe
Introduceti numarul de elemente din vector: 10
Introduceti elementul maxim(possibil) din vector: 10
-----
Timp sortare bubblesort:0.000000700 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.000000400 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.000001300 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:0.000000300 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.000003600 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.000001000 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.000001100 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.000001300 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe
Introduceti numarul de elemente din vector: 100
Introduceti elementul maxim(possibil) din vector: 100
-----
Timp sortare bubblesort:0.000035500 secunde
Vectorul a fost sortat corect folosind bubblesort!
-----
Timp sortare countsort:0.000001300 secunde
Vectorul a fost sortat corect folosind countsort!
-----
Timp sortare mergesort:0.000008500 secunde
Vectorul a fost sortat corect folosind mergesort!
-----
Timp sortare insertion:0.000000900 secunde
Vectorul a fost sortat corect folosind insertionsort!
-----
Timp sortare radixsort_lsd:0.000010600 secunde
Vectorul a fost sortat corect folosind radixsort_lsd!
-----
Timp sortare quicksort:0.000020300 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = ultimul element)!
-----
Timp sortare quicksort:0.000020400 secunde
Vectorul a fost sortat corect folosind quicksort (pivot = mediana din 3)!
-----
Timp sortare nativa:0.000006800 secunde
Vectorul a fost sortat corect folosind sortarea nativa a limbajului c++!
-----
Continuati executarea programului? Apasati tasta 1 in caz afirmativ, 0 in caz contrar:
```

Între vectorii cu 10 și 100 de elemente se poate observa cum Radix, unde în primul test cu 10 elemente este mai slab decât Quick, devine mai rapid în cel cu 100 de elemente, trecere care în cazul numerelor random se întâmplă la un număr mai mare de elemente. Pentru un vector constant, QuickSortul nu pare să fie alegerea cea mai bună.

Alte tipuri de teste: vector aproape sortat
crescător/descrescător

Pentru aceste tipuri de teste, timpii de rulare
sunt similari cu cei analizați mai sus.

Sfârșit!

C:\Users\Andreea\Desktop\codeblocks\proiect_sd\bin\Debug\proiect_sd.exe

— □ ×

```
0 zi buna!  
Process returned 0 (0x0)   execution time : 27.368 s  
Press any key to continue.
```