Teste random

TEST1: *1.000.000 numere*

Bubble Sort: peste 1 minut

Quick Sort(pivot determinist – in mijloc): 329 milisecunde

Quick Sort(pivot determinist –mediana din 3): 233 milisecunde

Merge Sort: 398 milisecunde

Radix Sort: 227 milisecunde

Count Sort: 23 milisecunde

Sort din STL: 294 milisecunde

TEST2: *10.000.000 numere*

Bubble Sort: peste 1 minut

Quick Sort(pivot determinist – in mijloc): 8788 milisecunde

Quick Sort(pivot determinist –mediana din 3): 2496 milisecunde

Merge Sort: 4426 milisecunde

Radix Sort: 2340 milisecunde

Count Sort: 224 milisecunde

Sort din STL: 3171 milisecunde

TEST3: 1.000.000 numere || max_number=1000

Bubble Sort: peste 1 minut

Quick Sort(pivot determinist – in mijloc): 2230 milisecunde

Quick Sort(pivot determinist –mediana din 3): 205 milisecunde

Merge Sort: 373 milisecunde

Radix Sort: 176 milisecunde

Count Sort: 23 milisecunde

Sort din STL: 339 milisecunde

TEST4: 1.000.000 numere || max_number=100

Bubble Sort: peste 1 minut

Quick Sort(pivot determinist – in mijloc): 20460 milisecunde

Quick Sort(pivot determinist –mediana din 3): 197 milisecunde

Merge Sort: 342 milisecunde

Radix Sort: 116 milisecunde

Count Sort: 31 milisecunde

Sort din STL: 254 milisecunde

TEST5: 1.000.000 numere || max_number=100000

Bubble Sort: peste 1 minut

Quick Sort(pivot determinist – in mijloc): 338 milisecunde

Quick Sort(pivot determinist –mediana din 3): 227 milisecunde

Merge Sort: 388 milisecunde

Radix Sort: 225 milisecunde

Count Sort: 24 milisecunde

Sort din STL: 304 milisecunde

TEST6: 10.000 numere || max_number=100000

Bubble Sort: 725 milisecunde

Quick Sort(pivot determinist – in mijloc): 2 milisecunde

Quick Sort(pivot determinist –mediana din 3): 2 milisecunde

Merge Sort: 2 milisecunde

Radix Sort: 3 milisecunde

Count Sort: 1 milisecunda

Sort din STL: 1 milisecunda

TEST7: 40.000.000 numere

Bubble Sort: peste 1 minut

Quick Sort(pivot determinist – in mijloc): 109725 milisecunde

Quick Sort(pivot determinist –mediana din 3): 10458 milisecunde

Merge Sort: 18777 milisecunde

Radix Sort: 9625 milisecunde

Count Sort: 935 milisecunde

Sort din STL: 13928 milisecunde

BubbleSort este cel mai lent algoritm de sortare. Pentru un vector cu peste 21000 de elemente devine foarte lent, peste 3100 de milisecunde. BubbleSort este mai eficient atunci cand vectorul este in mare parte ordonat crescator, astfel incat trebuie sa faca doar cateva interschimbari. BubbleSort este cel mai ineficient atunci cand vectorul este ordonat descrescator.

QuickSort este un algoritm rapid bazat pe comparatii, dar este influentat de alegerea pivotului si de ordinea elementelor din vector. QuickSort-ul cu pivotul ales in mediana din 3 este mai rapid decat QuickSort-ul cu pivotul ales in mijloc in toate testele. QuickSort este un algoritm eficient pentru vectori care au mai putin de 40000000 de elemente,depasind timpul de 10000 de milisecunde pentru sortarea vectorului.

MergeSort este un algoritm eficient pentru vectori care au mai putin de 22000000 de elemente, depasind timpul de 10100 de milisecunde. Imparte vectorul ales mereu in 2 parti pana ajunge ca vectorul respectiv sa 1 element sau 2, fiind constant indiferent de ordinea elementelor in vector.

RadixSort este foarte rapid, folosind shiftari. Pentru vectori cu mai mult de 42000000 de elemente, sortarea depaseste timpul de 10000 de milisecunde. RadixSort poate sorta doar numere naturale, fiind influentat de marimea numerelor si a vectorului. Totodata este influentat si de baza in care este realizat.

CountSort este un algoritm foarte rapid de sortare, fiind influentat doar de marimea vectorului. Poate sorta doar numere naturale. Foloseste un vector de frecventa in care retine de cate ori apare fiecare element, dupa care le afiseaza in ordine crescatoare, fiind foarte eficient din punct de vedere al timpului.

Sort-ul din STL poate fi utilizat pentru toate cazurile:numere mari, vectori cu foarte multe elemente, numere negative, numere rationale; fiind eficient.

Teste speciale

Bubblesort: alegem un vector sortat descrescator

Quicksort: alegem un vector care are valoarea 0 pe fiecare pozitie unde va fi pivotul in timpul sortarii