Proiect Sortari SD

realizat de Velciu Razvan Gabriel

Codul sursa folosit pentru acest proiect a fost scris in Python 3.11.1.

Am implementat urmatorii algoritmi de sortare:

- Insertion sort
- Counting sort
- Radix sort in baza 10, 2^{16}, 2^{8} realizat cu operatii pe biti, exceptand cel in baza 10
- Merge sort
- Shell sort
- Bubble sort

De asemenea, in testele pe care le-am efectuat am introdus si algoritmul de sortare al limbajului.

Au fost executate urmatoarele teste pentru a stabili diferentele dintre acesti algoritmi (N reprezinta numarul de elemente din vector si Max reprezinta care e cea mai mare valoare care poate aparea in vector) :

Vectorul sortat este generat random.

N = 1000 Max = 1000 :

```
Sortare corecta 0.024021863937377930 seconds insertionSort
Sortare corecta 0.0 seconds countSort
Sortare corecta 0.0010008811950683594 seconds radixSort
Sortare corecta 0.004004001617431641 seconds mergeSort
Sortare corecta 0.0020020008087158203 seconds shellSort
Sortare corecta 0.05705165863037109 4 seconds bubbleSort
Sortare corecta 0.0020027160644531250 seconds radixSort_8
Sortare corecta 0.015013694763183594 seconds radixSort_16
Sortare corecta 0.0 seconds Default Python Sort
```

```
N =  10000  Max =  1000 :

Sortare corecta 2.523293972015381  seconds insertionSort

Sortare corecta 0.0020015239715576172  seconds countSort

Sortare corecta 0.008006572723388672  seconds radixSort

Sortare corecta 0.05605101585388183 6  seconds mergeSort

Sortare corecta 0.036032676696777344  seconds shellSort

Sortare corecta 6.686097860336304  seconds bubbleSort

Sortare corecta 0.021018266677856445  seconds radixSort_8

Sortare corecta 0.023019790649414062  seconds radixSort_16

Sortare corecta 0.0010020732879638672  seconds Default Python Sort
```

```
N =  100000  Max =  1000 :

Sortare corecta 265.60192346572876  seconds insertionSort

Sortare corecta 0.01601576805114746  seconds countSort

Sortare corecta 0.08307552337646484  seconds radixSort

Sortare corecta 0.6786224842071533  seconds mergeSort

Sortare corecta 0.5242280960083008  seconds shellSort

Sortare corecta 721.6923620700836  seconds bubbleSort

Sortare corecta 0.2392165660858154З  seconds radixSort_8

Sortare corecta 0.08607864379882812  seconds radixSort_16

Sortare corecta 0.014012813568115234  seconds Default Python Sort
```

```
Vectorul sortat este  generat aproape sortat.
N =  1000  Max =  1000 :
Sortare corecta 0.0020017623901367188  seconds insertionSort
Sortare corecta 0.0 seconds countSort
Sortare corecta 0.0010006427764892578  seconds radixSort
Sortare corecta 0.005005598068237305  seconds mergeSort
Sortare corecta 0.0010004043579101562  seconds shellSort
Sortare corecta 0.03503131866455078  seconds bubbleSort
Sortare corecta 0.0020017623901367188  seconds radixSort_8
Sortare corecta 0.015013694763183594  seconds radixSort_16
Sortare corecta 0.0 seconds Default Python Sort
```

```
N =  10000  Max =  1000 :
Sortare corecta 0.013010501861572266  seconds insertionSort
Sortare corecta 0.002002000808715820З  seconds countSort
Sortare corecta 0.007005929946899414  seconds radixSort
Sortare corecta 0.05404973030090332  seconds mergeSort
Sortare corecta 0.016015291213989258  seconds shellSort
Sortare corecta 3.740398406982422  seconds bubbleSort
Sortare corecta 0.018016576766967773  seconds radixSort_8
Sortare corecta 0.021018266677856445  seconds radixSort_16
Sortare corecta 0.0 seconds Default Python Sort
```

```
N =  100000  Max =  1000 :
Sortare corecta 0.1481325626373291  seconds insertionSort
Sortare corecta 0.018015384674072266  seconds countSort
Sortare corecta 0.10209274291992188  seconds radixSort
Sortare corecta 0.6686065196990967  seconds mergeSort
Sortare corecta 0.19717907905578613  seconds shellSort
Sortare corecta 412.63363432884216  seconds bubbleSort
Sortare corecta 0.20350265502929688  seconds radixSort_8
Sortare corecta 0.08403396606445312  seconds radixSort_16
Sortare corecta 0.0020022392272949 22  seconds Default Python Sort
```

Vectorul sortat este generat cu elementele ordonate descrescator.

```
N =  10000  Max =  1000 :
Sortare corecta 5.292120456695557  seconds insertionSort
Sortare corecta 0.002001523971557617  seconds countSort
Sortare corecta 0.008006811141967773  seconds radixSort
Sortare corecta 0.05004572868347168  seconds mergeSort
Sortare corecta 0.02101874351501465  seconds shellSort
Sortare corecta 9.248378276824951  seconds bubbleSort
Sortare corecta 0.01901721954345703  seconds radixSort_8
Sortare corecta 0.021018028259277344  seconds radixSort_16
Sortare corecta 0.0 seconds Default Python Sort
```

```
N =  1000  Max =  1000 :
Sortare corecta 0.05404806137084961  seconds insertionSort
Sortare corecta 0.0010006427764892578  seconds countSort
Sortare corecta 0.001001596450805664  seconds radixSort
Sortare corecta 0.0040035247802734375  seconds mergeSort
Sortare corecta 0.0020017623901367188  seconds shellSort
Sortare corecta 0.08807921409606934  seconds bubbleSort
Sortare corecta 0.0020020008087158203  seconds radixSort_8
Sortare corecta 0.016015291213989258  seconds radixSort_16
Sortare corecta 0.0 seconds Default Python Sort
```

```
N =  100000  Max =  1000 :
Sortare corecta 555.8084895610809  seconds insertionSort
Sortare corecta 0.029026746749877930  seconds countSort
Sortare corecta 0.1271157264709472 7  seconds radixSort
Sortare corecta 0.6991040706634521  seconds mergeSort
Sortare corecta 0.3433103561401367  seconds shellSort
Sortare corecta 1009.7146062850952  seconds bubbleSort
Sortare corecta 0.1999821662902832  seconds radixSort_8
Sortare corecta 0.0840005874633789  seconds radixSort_16
Sortare corecta 0.0020012855529785156  seconds Default Python Sort
```

Se poate observa la insertionSort si bubbleSort cum devin din ce in ce mai lente cu cresterea numarului de elemente intr-o maniera foarte sesizabila. Acest lucru se datoareaza tuturor comparatiilor pe care trebuie sa le faca, avand o complexitate de O(n^2).

Se observa totusi ca in cazul in care vectorul este sortat descrescator timpul de executie este cel mai ridicat la ele, acesta fiind Worst Case-ul acestora. Diferenta este foarte sesizabila, algoritmii devenind neutilizabil de lenti comparativ cu celelalte.

https://we.tl/t-bepudcLUQV

In acest link am adaugat mai multe teste pe care le-am rulat pe algoritmii acestia de sortare, cu Max de la 10^3 pana la 10^8 si cu vectori generati random, aproape sortati si sortati descrescator. In aceste teste se observa cresterea exponentiala a countSortului din cauza acelui vector de frecventa. RadixSorturile s-au comportat destul de bine in fata tuturor testelor, insa algoritmul de sortare al limbajului Python a castigat in fiecare test.