

Prezentare algoritmi de sortare

Am implementat in c++ urmatorii algoritmi:

- BubbleSort
- CountSort
- RadixSort
- MergeSort
- QuickSort

Dupa ce am masurat timpii de rulare pentru fiecare algoritm pe mai multe teste am putut observa diferenta dintre ei in functie de datele de intrare.

Pentru putine numere toti algoritmii sunt foarte rapizi.

```
TEST 1
N = 100 NMax = 1000

bubbleSort: elapsed time = 0 CORECT
countSort: elapsed time = 0 CORECT
radixSort: elapsed time = 0 CORECT
mergeSort: elapsed time = 0 CORECT
quickSort: elapsed time = 0 CORECT

Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

De la 10^4 deja apar diferente intre ei, iar pentru 10^5 bubbleSort avand complexitatea $O(n^2)$ este foarte lent, de aceea nu o sa il las sa sorteze mai mult de atat

```
TEST 1
N = 10000 NMax = 1000

bubbleSort: elapsed time = 0.130106 CORECT
countSort: elapsed time = 0 CORECT
radixSort: elapsed time = 0 CORECT
mergeSort: elapsed time = 0.0009961 CORECT
quickSort: elapsed time = 0.0012691 CORECT

Process returned 0 (0x0)   execution time : 0.149 s
Press any key to continue.
```

```
TEST 1
N = 100000 NMax = 1000

bubbleSort: elapsed time = 15.0347 CORECT
countSort: elapsed time = 0.0010228 CORECT
radixSort: elapsed time = 0.0038009 CORECT
mergeSort: elapsed time = 0.0237692 CORECT
quickSort: elapsed time = 0.0112416 CORECT

Process returned 0 (0x0)   execution time : 15.097 s
Press any key to continue.
```

CountSort este foarte rapid pentru numere mici, dar daca marim Nmax creste si timpul de rulare. Acum este mult mai lent chiar si decat bubbleSort.

Complexitatea lui CountSort este $O(n + \max)$ pentru timp si $O(\max)$ pentru spatiu, ceea ce il face ineficient pentru numere mari.

```
TEST 1
N = 10000 NMax = 1000000000

bubbleSort: elapsed time = 0.13041 CORECT
countSort: elapsed time = 12.426 CORECT
radixSort: elapsed time = 0.000974 CORECT
mergeSort: elapsed time = 0.0019946 CORECT
quickSort: elapsed time = 0.000997 CORECT

Process returned 0 (0x0)   execution time : 12.598 s
Press any key to continue.
```

RadixSort (implementat lsd in baza 10) are timpii de executare mai rapizi decat quicksort pentru numere mici, iar pentru numere mari timpii de rulare sunt comparabili, dar radixSort foloseste memorie in plus.

RadixSort are complexitatea $O(n \log \max)$ pentru timp si $O(n + b)$ pentru spatiu.

```
TEST 1
N = 100000000 NMax = 1000000000

bubbleSort: NU POT SORTA

countSort: elapsed time = 8.6577 CORECT
radixSort: elapsed time = 1.31011 CORECT
mergeSort: elapsed time = 2.83778 CORECT
quickSort: elapsed time = 1.12951 CORECT

Process returned 0 (0x0)   execution time : 14.332 s
Press any key to continue.
```

In toate testele pe care le-am facut mergeSort ruleaza mai lent decat radixSort si quicksort, avantajul sau insa este ca are complexitatea $O(n \log n)$ indiferent de marimea vectorului sau de intervalul din care apartin numerele.

Astfel, in testele de mai jos vedem ca mergeSort are aproximativ acelasi timp de rulare pentru $N_{\max} = 10^4$ cat si pentru $N_{\max} = 10^9$ spre deosebire de ceilalti algoritmi unde vedem variatii mai mari.

```

TEST 1
N = 10000000 NMax = 1000000000

bubbleSort: NU POT SORTA

countSort: elapsed time = 6.29118 CORECT

radixSort: elapsed time = 0.770939 CORECT

mergeSort: elapsed time = 2.17118 CORECT

quickSort: elapsed time = 0.930512 CORECT

Process returned 0 (0x0)   execution time : 10.430 s
Press any key to continue.

```

```

TEST 1
N = 10000000 NMax = 10000

bubbleSort: NU POT SORTA

countSort: elapsed time = 0.0897597 CORECT

radixSort: elapsed time = 0.426849 CORECT

mergeSort: elapsed time = 1.95377 CORECT

quickSort: elapsed time = 4.25659 CORECT

Process returned 0 (0x0)   execution time : 6.975 s
Press any key to continue.

```

QuickSort are complexitatea medie $O(n \log n)$ insa poate varia foarte mult in functie de cum alegem pivotul dupa care este impartit vectorul. In implementarea mea, am ales ca pivot ultimul element iar algoritmul este foarte rapid pe majoritatea testelor de mai sus. Problema apare cand vectorul este constant sau deja sortat deoarece complexitatea ajunge la $O(n^2)$. Mai jos am testat cu doar 10^4 elemente iar quicksort este mai lent chiar si decat bubbleSort, deoarece vectorul este constant.

```

TEST 1
N = 10000 NMax = 1000

bubbleSort: elapsed time = 0.0299197 CORECT

countSort: elapsed time = 0 CORECT

radixSort: elapsed time = 0 CORECT

mergeSort: elapsed time = 0.0019908 CORECT

quickSort: elapsed time = 0.0469261 CORECT

Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.

```

Avantajul la quicksort este sortarea efectuata in-place, fara a folosi memorie aditionala.

Am comparat timpii de rulare si cu functia sort() din c++ care foloseste introSort. IntroSort este un algoritm hibrid, acesta combinand mai multi algoritmi (quickSort, heapSort, insertionSort).

```

TEST 1
N = 1000000 NMax = 1000000

bubbleSort: NU POT SORTA

countSort: elapsed time = 0.0289491 CORECT

radixSort: elapsed time = 0.0518615 CORECT

mergeSort: elapsed time = 0.212457 CORECT

quickSort: elapsed time = 0.0817819 CORECT

introSort: elapsed time = 0.0628467 CORECT

```

```

TEST 1
N = 10000000 NMax = 100000000

bubbleSort: NU POT SORTA

countSort: elapsed time = 1.8421 CORECT

radixSort: elapsed time = 0.682151 CORECT

mergeSort: elapsed time = 2.19715 CORECT

quickSort: elapsed time = 0.964455 CORECT

introSort: elapsed time = 0.714128 CORECT

```

