


# **COMPARAREA ALGORITMILOR DE SORTARE**

# DESPRE PROIECT

Acest proiect vizeaza compararea algoritmilor de sortare pe un numar arbitrar de teste diferite, pentru a vedea ce algoritm este optim pentru un anume set de date de intrare.

Inputul se face din fisier, in care se afla in aceasta ordine, numarul de teste, si pentru fiecare test numarul de elemente si maximul sirului care trebuie sortat.

Sirurile sunt generate de o functie care genereaza aleator numere intre 0 si maximul citit de la tastatura.



# ALGORITMII UTILIZATI

- Bubblesort;
- CountSort;
- MergeSort;
- RadixSort, cu bazele 2,16 si 256;
- C++NativeSort;
- Quicksort, pentru care am folosit 3 tipuri de pivot:
  - Pivot standard (ultimul element al sirului);
  - Mediana din 3;
  - Pivot generat random;







# N=10000 ( $10^4$ )

Bubble sort-ul devine clar, foarte ineficient in comparatie cu celelalte sortari, cel mai rapid pentrua cest input ramananc count sort-ul

```
-----
Testul 4| numar elemente: 10000| maxim: 10
-----
>BubbleSort:                               Sortare reusita | 0.879971 secunde
>QuickSort:
  -Pivot standard(ultimul element): Sortare reusita | 0.115005 secunde
  -Pivot mediana(din 3):                   Sortare reusita | 0.002001 secunde
  -Pivot random:                           Sortare reusita | 0.002011 secunde
>CountSort:                               Sortare reusita | 0 secunde
>MergeSort:                               Sortare reusita | 0.040001 secunde
>C++NativeSort:                           Sortare reusita | 0.002002 secunde
>RadixSort:
  -Baza 2:                                 Sortare reusita | 0.003999 secunde
  -Baza 16:                                Sortare reusita | 0.001 secunde
  -Baza 256:                               Sortare reusita | 0.00101 secunde
-----
```

# N=100000 ( $10^5$ )

Bubble sort-ul nu mai ruleaza sub 100 de secunde, deci este considerat extrem de ineficient pentru acest input, iar count sort-ul ramane cel mai eficient

```
-----
Testul 5| numar elemente: 100000| maxim: 10
-----
>BubbleSort:                                Ineficient pentru n> 5*10^4

>QuickSort:
  -Pivot standard(ultimul element): Sortare reusita | 11.6866 secunde
  -Pivot mediana(din 3):                Sortare reusita | 0.025998 secunde
  -Pivot random:                        Sortare reusita | 0.021 secunde

>CountSort:                                Sortare reusita | 0.002 secunde

>MergeSort:                                Sortare reusita | 0.377988 secunde

>C++NativeSort:                           Sortare reusita | 0.022997 secunde

>RadixSort:
  -Baza 2:                               Sortare reusita | 0.038999 secunde
  -Baza 16:                              Sortare reusita | 0.009 secunde
  -Baza 256:                             Sortare reusita | 0.009999 secunde
-----
```



# N=1000000 ( $10^6$ )

Putem observa ca si quick sort-ul depaseste pragul de 100 de secunde.

```
-----  
Testul 6| numar elemente: 1000000| maxim: 10  
-----  
>BubbleSort:                                Ineficient pentru n> 5*10^4  
  
>QuickSort:  
-Pivot standard(ultimul element): Ineficient pentru n> 10^6  
  
-Pivot mediana(din 3):                      Sortare reusita | 0.217994 secunde  
  
-Pivot random:                              Sortare reusita | 0.230993 secunde  
  
>CountSort:                                Sortare reusita | 0.017001 secunde  
  
>MergeSort:                                Sortare reusita | 3.72588 secunde  
  
>C++NativeSort:                             Sortare reusita | 0.300988 secunde  
  
>RadixSort:  
-Baza 2:                                    Sortare reusita | 0.354978 secunde  
  
-Baza 16:                                   Sortare reusita | 0.085998 secunde  
  
-Baza 256:                                  Sortare reusita | 0.084998 secunde  
-----
```

**N=10000000 (  $10^7$  )**

Merge sort-ul se apropie de pragul de 100 de secunde, iar count sort-ul ramane cel mai eficient, in timp ce restul algoritmilor au timpi asemanatori de rulare.

```
Testul 7| numar elemente: 10000000| maxim: 10
```

```
>BubbleSort:          Ineficient pentru  $n > 5 \cdot 10^4$ 
```

```
>QuickSort:
```

- Pivot standard(ultimul element): Ineficient pentru  $n > 10^6$

```
-Pivot mediana(din 3):          Sortare reusita | 2.53392 secunde
```

```
-Pivot random:          Sortare reusita | 2.57792 secunde
```

```
>CountSort:          Sortare reusita | 0.167005 secunde
```

```
>MergeSort:          Sortare reusita | 41.8787 secunde
```

```
>C++NativeSort:          Sortare reusita | 3.4949 secunde
```

```
>RadixSort:
```

[illegible][illegible]

```
-Baza 256:          Sortare reusita | 1.04197 secunde
```

# N=100000000 ( $10^8$ )

Timpul de sortare incepe sa creasca, count sort-ul ramanand aproape constant.

```
-----
Testul 1| numar elemente: 100000000| maxim: 10
-----
>BubbleSort:                               Ineficient pentru n> 5*10^4

>QuickSort:
  -Pivot standard(ultimul element): Ineficient pentru n> 10^6

  -Pivot mediana(din 3):                    Sortare reusita | 28.0357 secunde

  -Pivot random:                            Sortare reusita | 27.993 secunde

>CountSort:                                Sortare reusita | 1.64896 secunde


>MergeSort:                                Ineficient pentru n> 10^7

>C++NativeSort:                             Sortare reusita | 38.7261 secunde
```

Cat despre radix sort-uri, este folosita prea multa memorie RAM din cauza bucket-urilor folosite, si ruleaza pe sisteme mai performante, pe care a scos timpi decenti, dar nu ar fi corecta comparatia intre 2 sisteme diferite.

# DEPENDENTA ALGORITMILOR DE MAXIMUL DIN SIR N-UL FIIND CONSTANT 100

De la  $\text{maxim}=10$  pana la  $\text{maxim}=10^5 - 1$ , timpii de sortare sunt egali si minimi,  
Diferentele incep abia la  $\text{maxim} \geq 10^5$







# UN TEST IN CARE $N=10^8$ , IAR MAXIM=23123

```
-----  
Testul 8| numar elemente: 100000000| maxim: 23123  
-----
```

```
>BubbleSort:                      Absolut ineficient
```

```
>QuickSort:
```

```
-Pivot standard(ultimul element): Absolut ineficient
```

```
-Pivot mediana(din 3):             Sortare reusita | 22.3132 secunde
```

```
-Pivot random:                     Sortare reusita | 21.6553 secunde
```

```
>CountSort:                        Sortare reusita | 1.04751 secunde
```

```
>MergeSort:                        Absolut ineficient
```

```
>RadixSort:
```

```
-Baza 2:                           Sortare reusita | 63.2171 secunde
```

```
-Baza 16:                          Sortare reusita | 17.6301 secunde
```

```
-Baza 256:                         Sortare reusita | 8.83417 secunde
```

```
>C++NativeSort:                   Sortare reusita | 20.1533 secunde
```

## PENTRU UN SIR CONSTANT DE $10^4$ ELEMENTE

Desi pare un test inutil,  
apare ceva ciudat,  
bubble sort-ul este cel  
mai rapid, iar quicksort-  
ul foarte lent.

```
Testul 1| numar elemente: 10000| maxim: 32568
```

```
>BubbleSort:          Sortare reusita | 0 secunde
```

```
>QuickSort:
```

```
-Pivot standard(ultimul element): Sortare reusita | 0.813975 secunde
```

```
-Pivot mediana(din 3):          Sortare reusita | 0.001013 secunde
```

```
-Pivot random:          Sortare reusita | 0.000999 secunde
```

```
>CountSort:      Sortare reusita | 0.002009 secunde
```

```
>MergeSort:      Sortare reusita | 0.037999 secunde
```

```
>C++NativeSort:          Sortare reusita | 0.002009 secunde
```

```
>RadixSort:
```

[illegible]

```
-Baza 16:          Sortare reusita | 0.004001 secunde
```

```
-Baza 256:          Sortare reusita | 0.002991 secunde
```



## PENTRU UN SIR DESCRESCATOR

```
Testul 1| numar elemente: 10000| maxim: 10000
```

```
>BubbleSort:      Sortare reusita | 1.38799 secunde
```

```
>QuickSort:
```

```
-Pivot standard(ultimul element): Sortare reusita | 0.469002 secunde
```

```
-Pivot mediana(din 3):      Sortare reusita | 0.001015 secunde
```

```
-Pivot random:      Sortare reusita | 0 secunde
```

```
>CountSort:          Sortare reusita | 0.001013 secunde
```

```
>MergeSort:      Sortare reusita | 0.046408 secunde
```

```
>C++NativeSort:          Sortare reusita | 0.000999 secunde
```

```
>RadixSort:
```

[illegible]

```
-Baza 16:          Sortare reusita | 0.004 secunde
```

```
-Baza 256:                Sortare reusita | 0.003001 secunde
```

# COMPLEXITATI:

CASES	Bubble	Merge	Count	Quick std	Quick median	Quick random	Native	Radix 2	Radix 16	Radix 256
Best	n	nlogn	n + max	nlogn	nlogn	nlogn	nlogn	Unk	Unk	Unk
Average	n <sup>2</sup>	nlogn	n + max	nlogn	nlogn	nlogn	nlogn	Unk	Unk	Unk
Worst	n <sup>2</sup>	nlogn	n + max	nlogn	nlogn	nlogn	nlogn	Unk	Unk	Unk
Mem	1	n	n + max	logn	logn	logn	logn	n+ 2 <sup>2</sup>	n+ 2 <sup>4</sup>	n+ 2 <sup>8</sup>

# MENTIUNI

- Random pivot Quicksort este o loterie, deoarece pivotul ales aleator nu este mereu unul optim, sau macar average.
- Radix sort ar functiona mai rapid pe teste cu  $n$  foarte mare, daca baza ar fi o putere mai mare a lui 2, dar va consuma foarte multa memorie din cauza bucketurilor
- Bubble sort nu merita implementat pentru siruri lungi, dar in acelasi timp este foarte “fiabil” pentru siruri mici.
- C++NativeSort este surprinzator de eficient pentru siruri lungi, dar nu merita folosit pentru numere foarte mari.

# RADIX VS COUNT

In urma analizei facuta pe sortari, pot afirma ca ar trebui sa folosim radix sort-ul pentru numere mari ( $>10^6$ ), intrucat vectorul de frecventa nu poate fi suficient de mare, iar pentru numere mai mici, count sort-ul, nu numai ca este considerabil mai rapid pe orice lungime a sirului, dar si este mai economic din punct de vedere al memoriei.

# QUICKSORTS

Alegerea pivotului este o controversa pentru multi, existand numeroase implementari ale acesteia, printre cele mai eficiente se numara mediana din 3, din 5, mediana medianelor.

Concret, pe input-urile testate, mediana din 3 a fost cel mai eficient pivot, urmat de pivotul ales random, care nu este stabil, si de pivotul ales static .

RESULTATE GENERALE
 (SECONDE)

N= Max=	10 10	10^2 10^2	10^3 10^3	10^4 10^4	10^5 10^5	10^6 10^6	10^7 10^7	10^8 10^8
Bubble	0	0	0.009	1.014	>100	>100	>100	>100
Native	0	0	0.001	0.002	0.029	0.349	3.972	45.87
Merge	0	0	0.003	0.036	0.366	3.669	42.42	>100
Count	0	0	0	0.001	0.011	0.087	1.18	14.94
Radix 2	0	0	0.001	0.016	0.173	1.738	23.625	>100
Radix 16	0	0	0	0.004	0.048	0.426	5.799	66.24
Radix 256	0	0	0	0.002	0.03	0.27	2.939	34.281
Quick	0	0	0	0.003	0.029	>100	>100	>100
Quick med	0	0	0	0.001	0.022	0.265	2.965	33.889
Quick rnd	0	0	0	0.002	0.022	0.253	2.975	33.969