Algoritmi de sortare

Popel Emil-Bogdan

Bubble sort

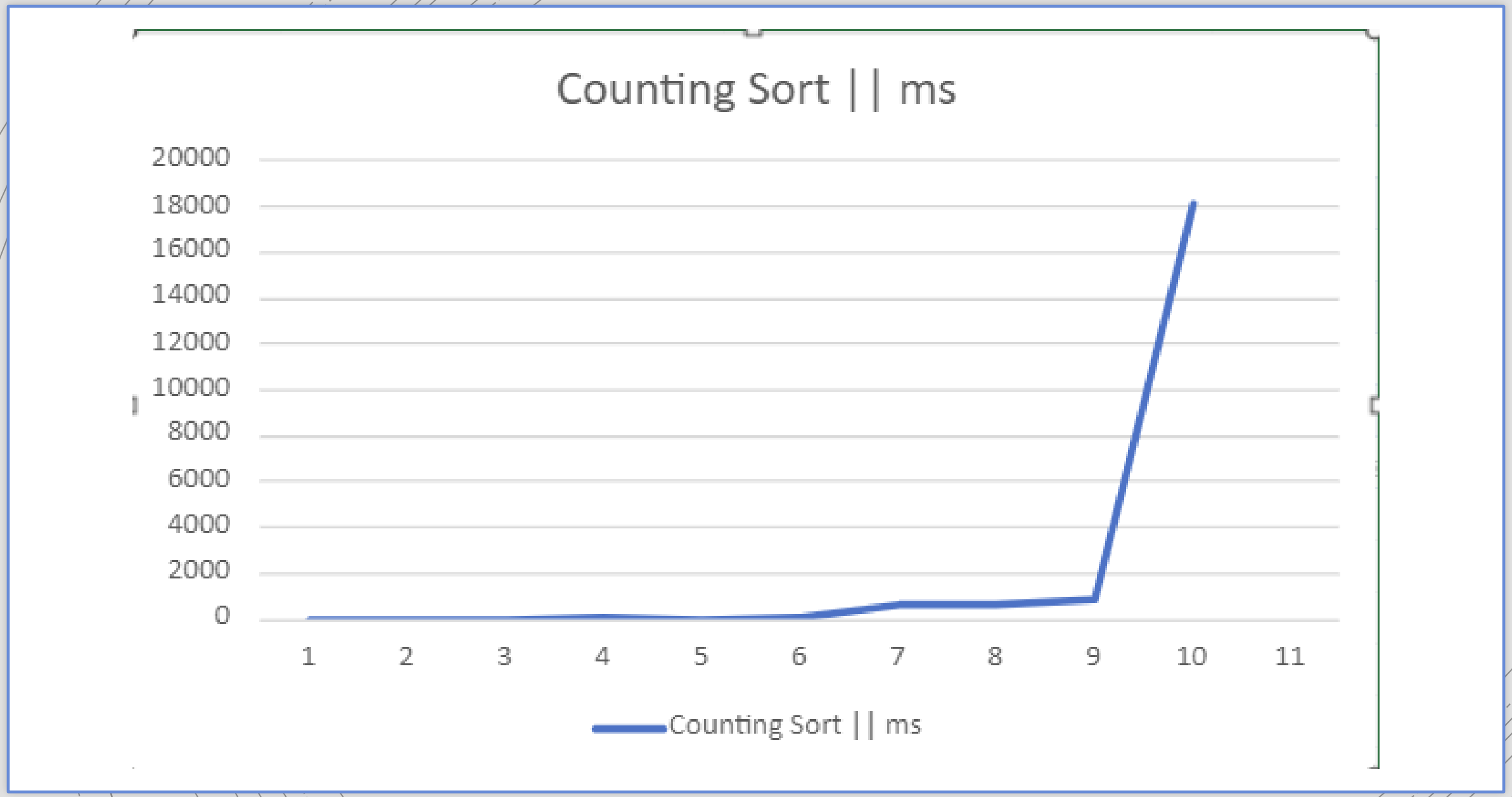
Bubble sort este cel mai simplu **algoritm de sortare cunoscut. Isi atinge limitele atunci cand:**

-vectorul de sortat este sortat deja - O(n)

-vectorul este inversat - O(n^2)

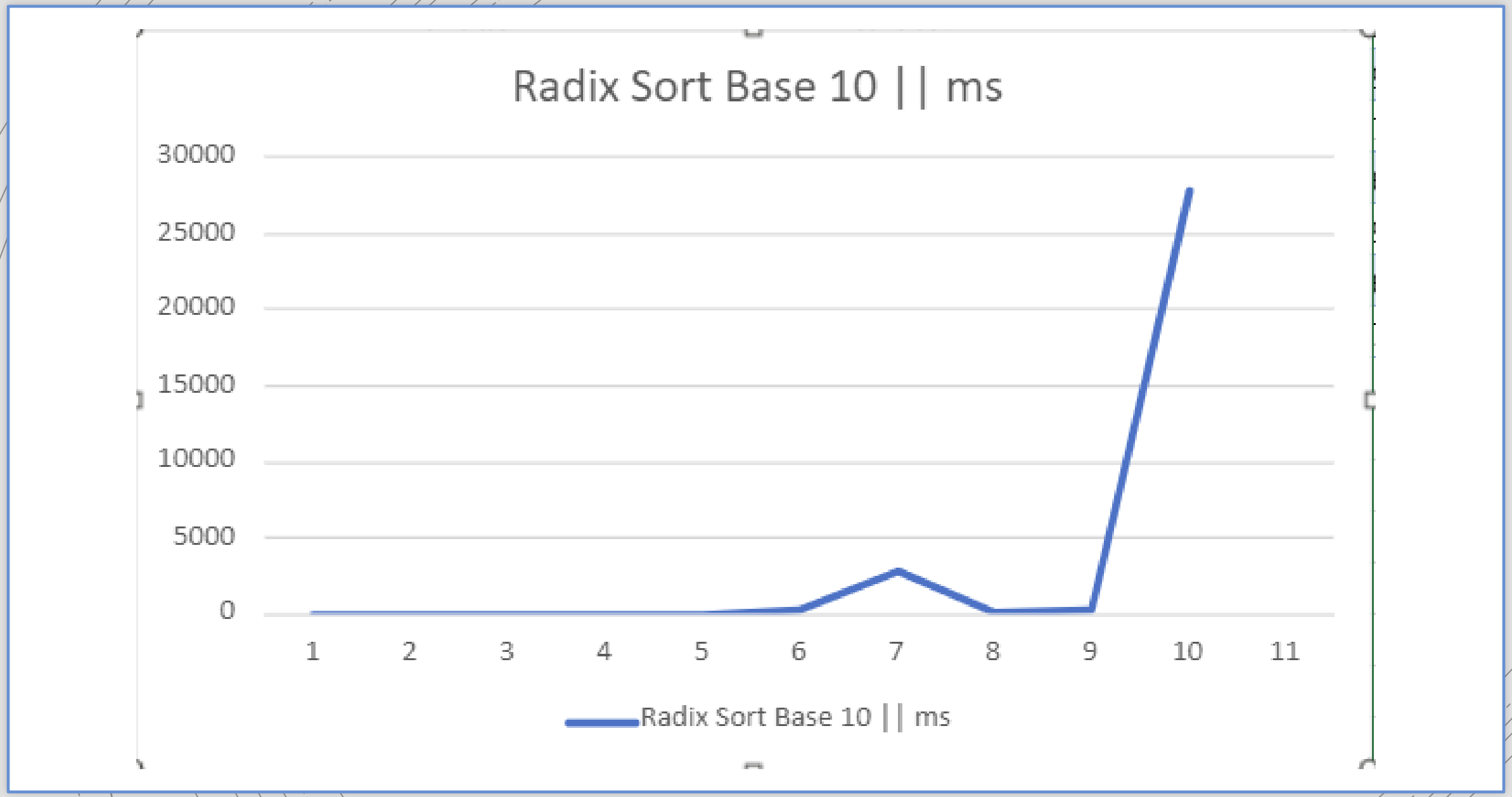
Counting Sort

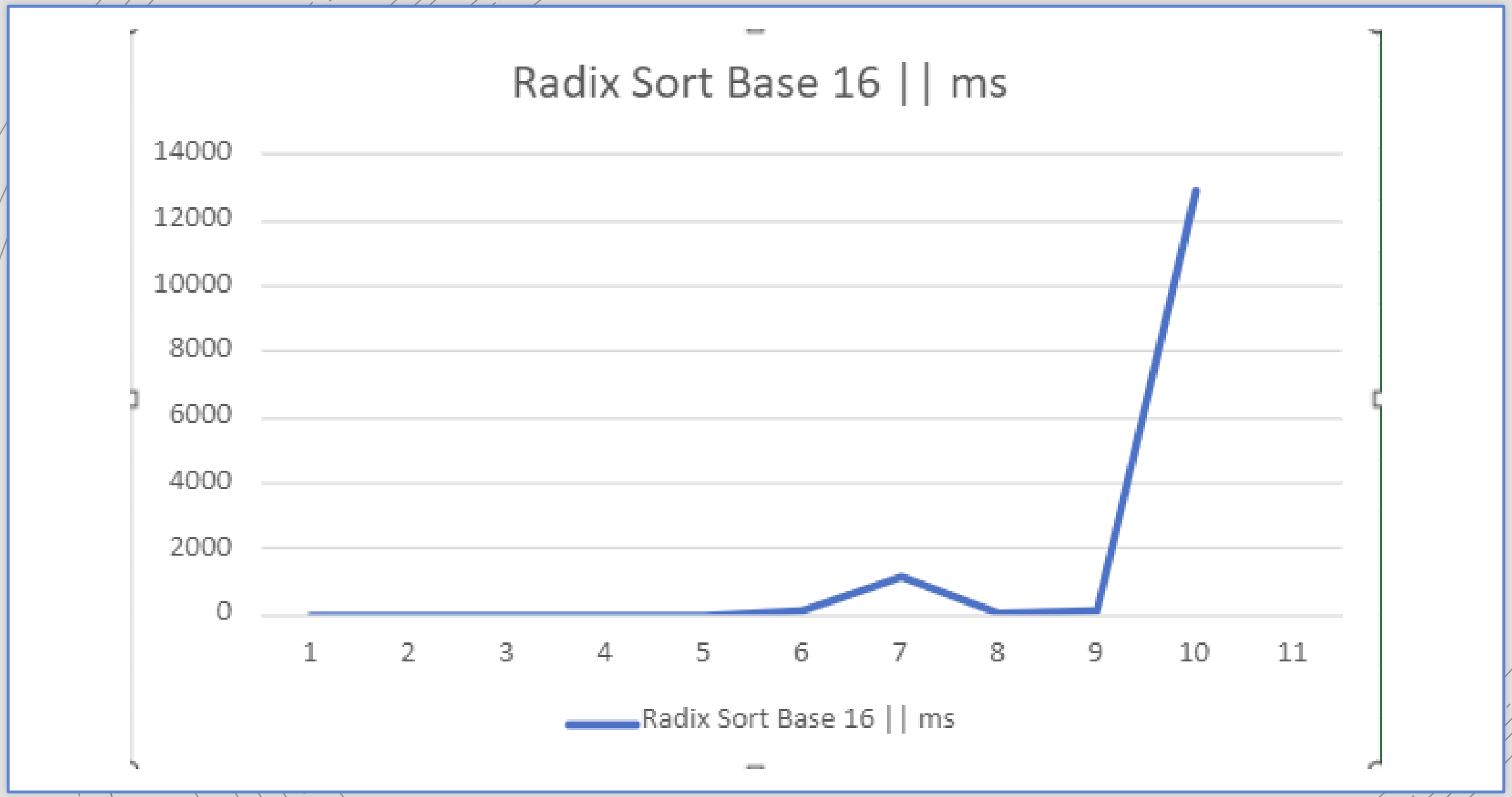
- Counting sort nu este un algoritm bazat pe comparatii. Acesta functioneaza cel mai bine atunci cand diferenta dintre cel mai mare element de sortat si cel mai mic element nu depaseste cu mult numarul elementelor ce trebuie a fi sortate.
- N = numarul de elemente supuse sortarii;
- K = MaxElement – MinElement;

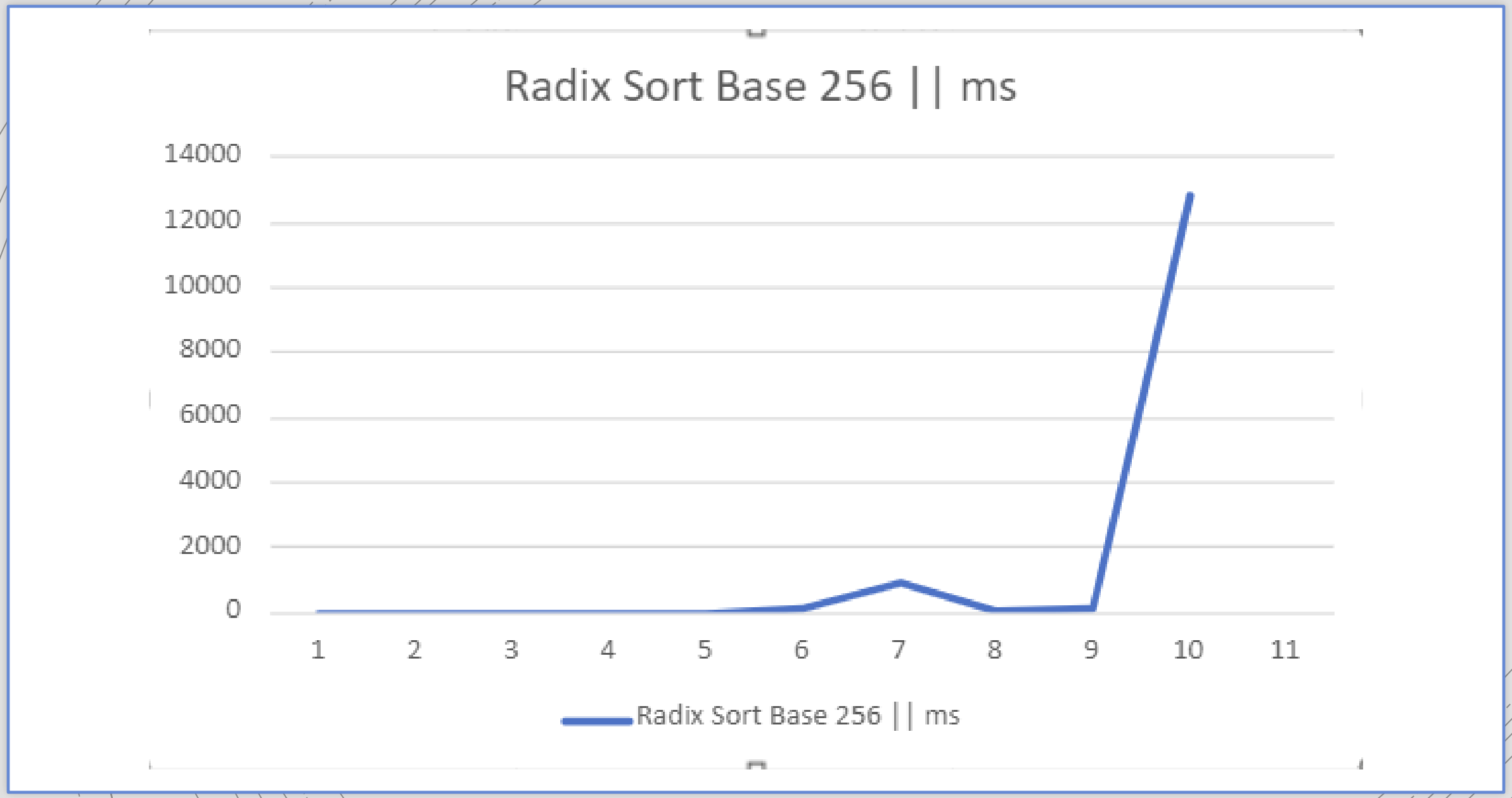

Counting Sort || ms
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10
11
Counting Sort || ms

Radix Sort

- Radix Sort este un algoritm bazat pe impartirea elementelor in "Bucket-uri" in functie de "radix". Cu alte cuvinte, elementul supus sortarii este introdus intr-un bucket in functie de cifra unitatilor la care a ajuns algoritmul. Schimbarea bazei cu care lucreaza algoritmul poate imbunatati semnificativ timpii de rulare. Din testele efectuate, reiese faptul ca baza 256 este cea mai rapida.


Radix Sort Base 10 || ms
30000
25000
20000
15000
10000
5000
0
1
2
3
4
5
6
7
8
9
10
11
Radix Sort Base 10 || ms


Radix Sort Base 16 || ms
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10
11
Radix Sort Base 16 || ms

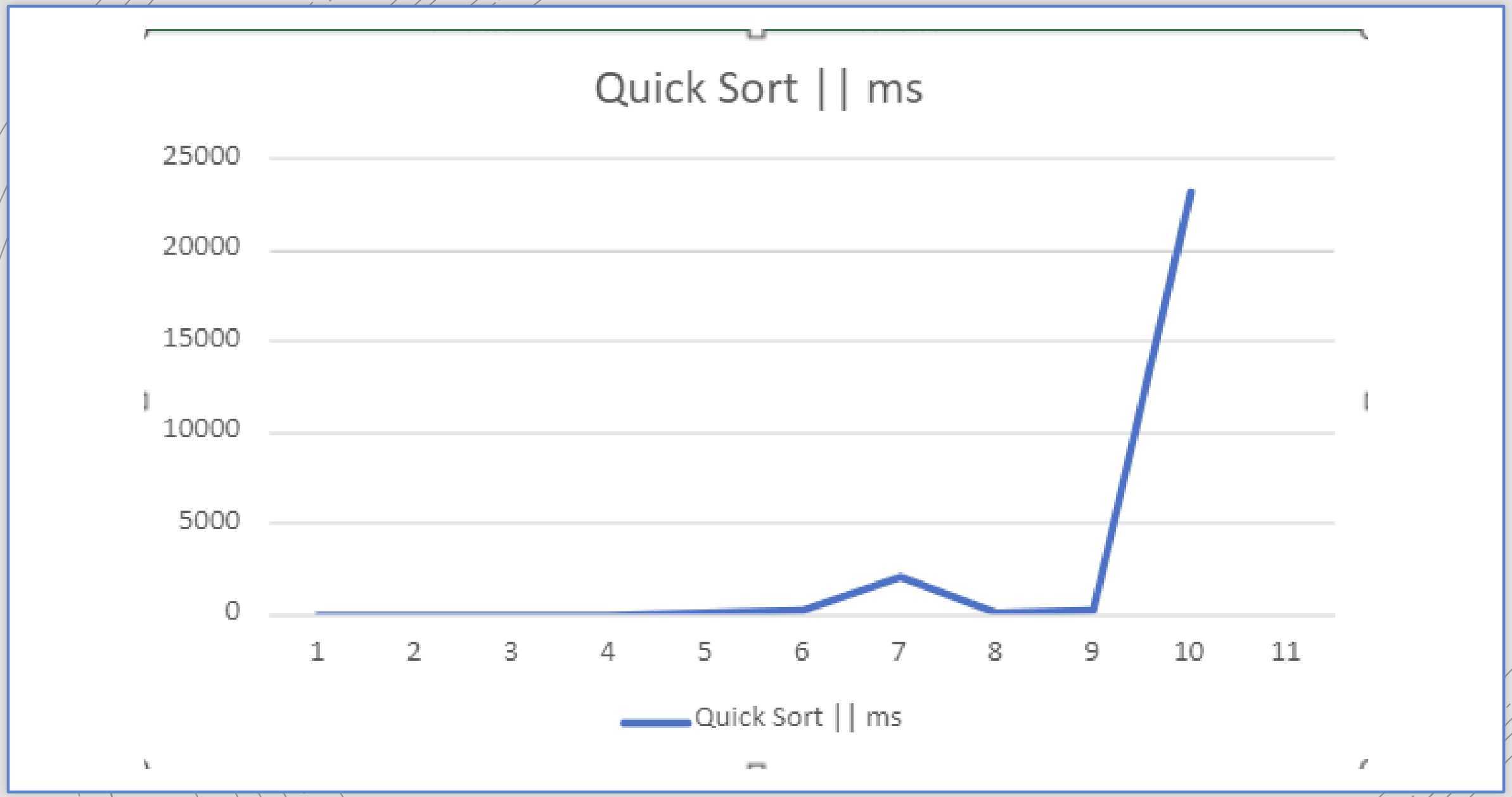

Radix Sort Base 256 || ms
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10
11
Radix Sort Base 256 || ms

Merge Sort

- Merge sort este un algoritm de sortare recursiv care preia un array si il imparte in jumatati, pe care le sorteaza, in final punandu-le cap la cap pentru a obtine array-ul sortat. Merge Sort reuseste sa imparta array-ul in 2 jumatati si sa le uneasca dupa sortare in timp liniar.

Quick Sort

- Quick Sort, asemenea Merge Sort-ului, este un algoritm de sortare de tip "Divide and Conquer". Cea mai importanta parte din implementarea Quick Sort-ului este alegerea pivotului si realizarea partitiilor in functie de pivot, element dupa care vor fi rearanjate restul elementelor.
- Cel mai rau caz: Algoritmul alege de fiecare data cel mai mic/cel mai mare element din partitie.
- Cel mai bun caz: Algoritmul alege de fiecare data elementul din mijloc.


Quick Sort || ms
25000
20000
15000
10000
5000
0
1
2
3
4
5
6
7
8
9
10
11
Quick Sort || ms

Insertion Sort

Functioneaza cel mai bine pe seturi de date mici.

Este un algoritm usor de urmarit si de implementat. Presupune alegerea unui Pivot in functie de care vor fi aranjate restul elementelor.

Isi pierde din eficienta rapid odata cu cresterea cantitatii de date.

TimSort

- TimSort este un algoritm stabil, hibrid intre Insertion Sort si Merge Sort. Ideea sta in impartirea vectorului in mai multe diviziuni de dimensiuni mici care sunt sortate cu ajutorul Insertion Sort-ului si lipite printr-o functie "Merge". Acesta ofera avantajele fiecarui algoritm pe care il imbina.

TIMPI DE RULARE IN MILISECUNDE

Nr. Elemente/NMAX	Bubble Sort || ms	Counting Sort || ms	Radix Sort Base 10 || ms	Radix Sort Base 16 || ms	Radix Sort Base 256 || ms	Merge Sort || ms	Quick Sort || ms	Insertion Sort || ms	Tim Sort || ms	C++ Sort() function
10 / 100	0	0	0	0	0	0	0	0	0	0
100 / 1000	0.997	0.997	0	0	0	0	0	0	0	0
1000 / 100000	8.972	0	0	0	0	0.643	0.981	0.997	0	0
10000 / 10000000	887.626	62.835	1.996	0.997	0.997	9.973	0.997	121.693	3.991	1.994
100000 / 100	86595.191	4.987	10.988	3.989	2.991	107.712	11.968	11964.034	36.934	21.943
1000000 / 100000	Nu sorteaza	43.881	203.457	90.759	88.764	1085.099	176.53	Nu sorteaza	39.993	249.333
10000000 / 10000	Nu sorteaza	675.254	2723.483	1145.496	925.639	1103.025	2012.837	Nu sorteaza	Nu Sorteaza	264.892
1000000 / 100000000	90274.746	660.236	28.922	12.943	12.966	109.707	15.957	12438.758	39.893	22.943
10000000 / 100000000	Nu sorteaza	824.798	294.213	129.654	134.641	1113.025	172.567	Nu sorteaza	Nu Sorteaza	263.295
100000000 / 100000000	Nu sorteaza	18090.743	27845.594	12948.469	12879.732	Nu sorteaza	23287.744	Nu sorteaza	Nu Sorteaza	35664.727

Valorile din vectori sunt complet randomizate. Prima coloana contine numarul de elemente citite si valoarea maxima pe care o avea un element din vector, in aceasta ordine.

NrElemente / ValMax

TIMPI DE RULARE IN MILISECUNDE

DESCENDING FROM:	Bubble Sort || ms	Counting Sort || n	Radix Sort Base 10 || ms	Radix Sort Base 16 || ms	Radix Sort Base 256 || ms	Merge Sort || ms	Quick Sort || ms	Insertion Sort || ms	Tim Sort || ms	C++ Sort() function
100000000	Nu sorteaza	4503.949	31175.713	13092.009	13245.598	Nu sorteaza	6927823	Nu sorteaza	Nu sorteaza	15332.022
10000000	Nu sorteaza	443.816	2807.619	1123.996	1125.986	Nu sorteaza	662.231	Nu sorteaza	Nu sorteaza	1448128
1000000	Nu sorteaza	41.889	241.354	96.741	94.747	1150.923	60.837	Nu sorteaza	Nu sorteaza	116.669
100000	103457.122	3.99	19.946	8.976	7.96	105.717	3.99	25124.969	35.885	9.973
10000	1011296	0.99	1.982	0	0	9.973	0.999	254.32	2.996	0
1000	9.973	0	0.997	0	0	0.997	0.997	1.994	0	0
100	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

- De aceasta data, valorile din vector sunt in ordine descrescatoare.
- Coloana din stanga contine valorile de la care pornesc.
- Ex: vector = [100, 99, 98, 97,...., 1]