Sorting algorithms

- a brief comparison -

Ocnaru Mihai

GitHub

Summary

- Within this power point I want to illustrate the best usecase for the following sorting algorithms choosen
- The tests were based on positive integers
- They were performed using different parameters, such as number of elements and max-value

Table of content

Sorting algorithms used

Count sort

Quick sort

Radix sort

Merge sort

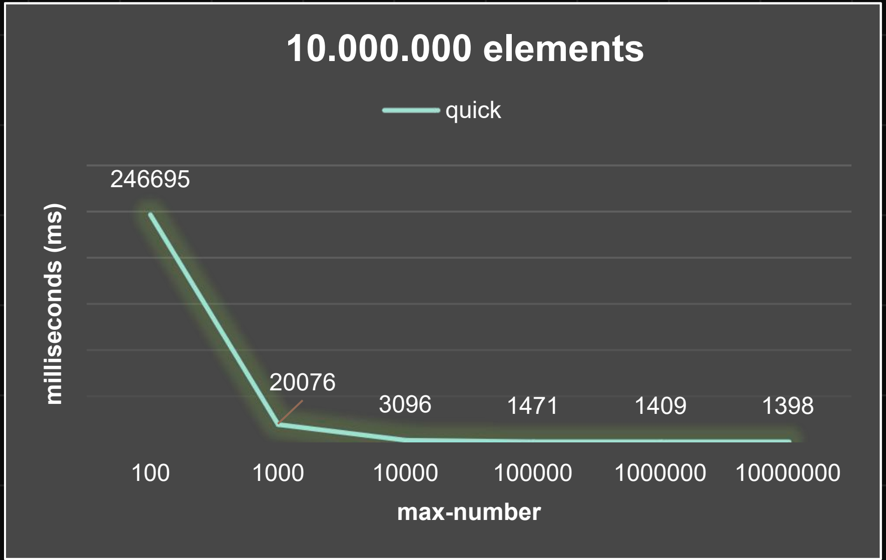
Shell sort

Quick sort

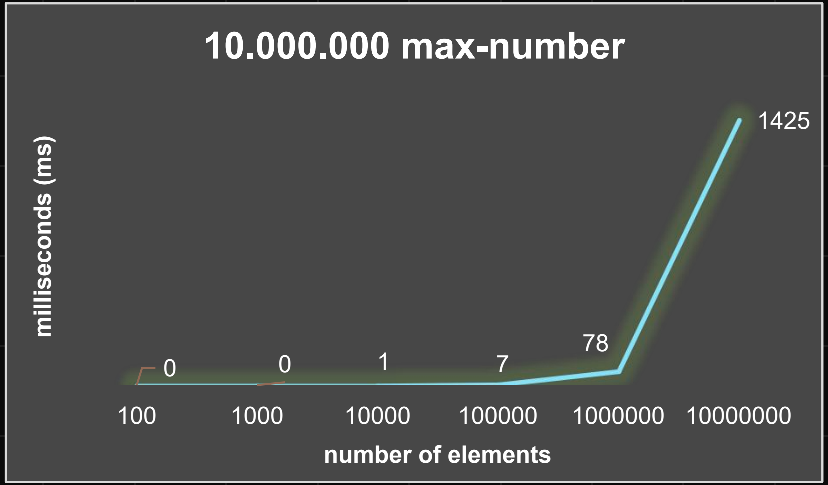
- Having a debateble Big-O complexity from $O(n \log n)$ to $O(n^2)$, quick sort is a divide and conquer algorithm, based on partitiong
- In my implementation, the pivot is chosen randomly

Worst case	Average case	Best case	Memory	Stable
n logn	n logn	n^2	n	no

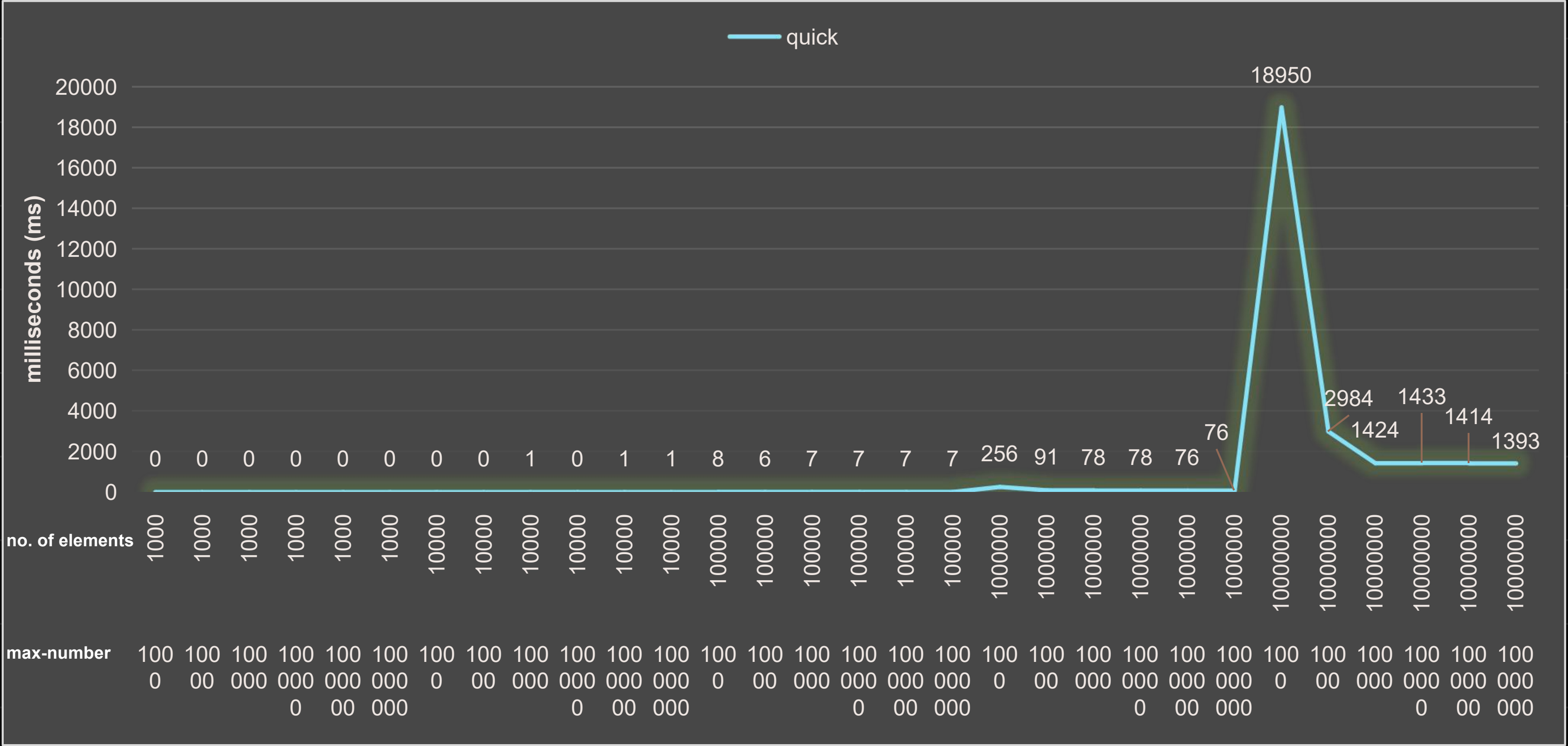
Quick sort



10.000.000 elements
quick
246695
20076
3096
1471
1409
1398
milliseconds (ms)
100
1000
10000
100000
1000000
10000000
max-number



10.000.000 max-number
1425
78
0
0
1
7
milliseconds (ms)
100
1000
10000
100000
1000000
10000000
number of elements



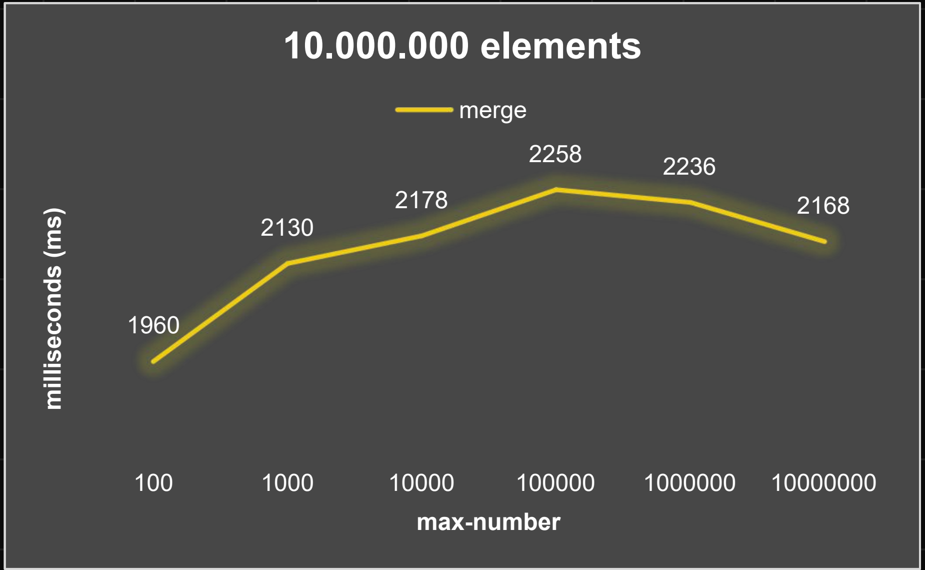
quick
milliseconds (ms)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
18950
2984
1433
1424
1414
1393
76
0 0 0 0 0 0 0 0 1 0 1 1 8 6 7 7 7 7 256 91 78 78 76
no. of elements
1000 1000 1000 1000 1000 1000 10000 10000 10000 10000 10000 10000 100000 100000 100000 100000 100000 100000 1000000 1000000 1000000 1000000 1000000 1000000 10000000 10000000 10000000 10000000 10000000 10000000
max-number
1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000

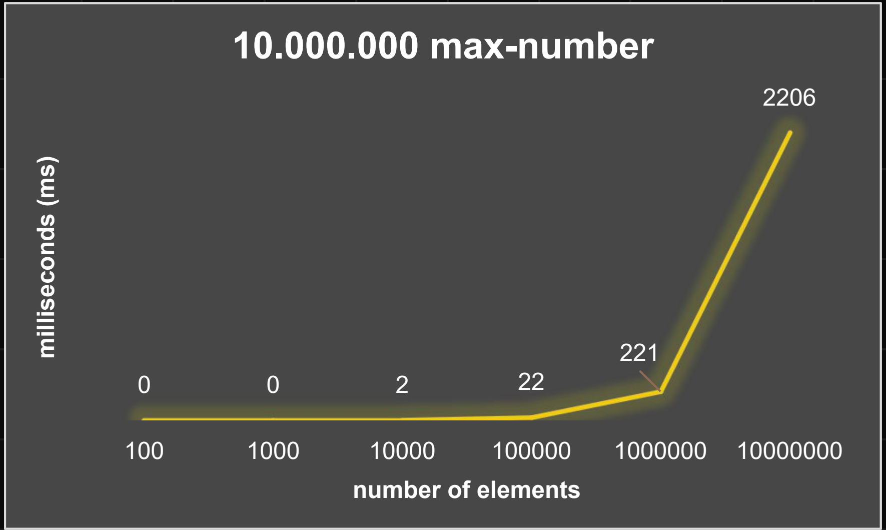
Merge sort

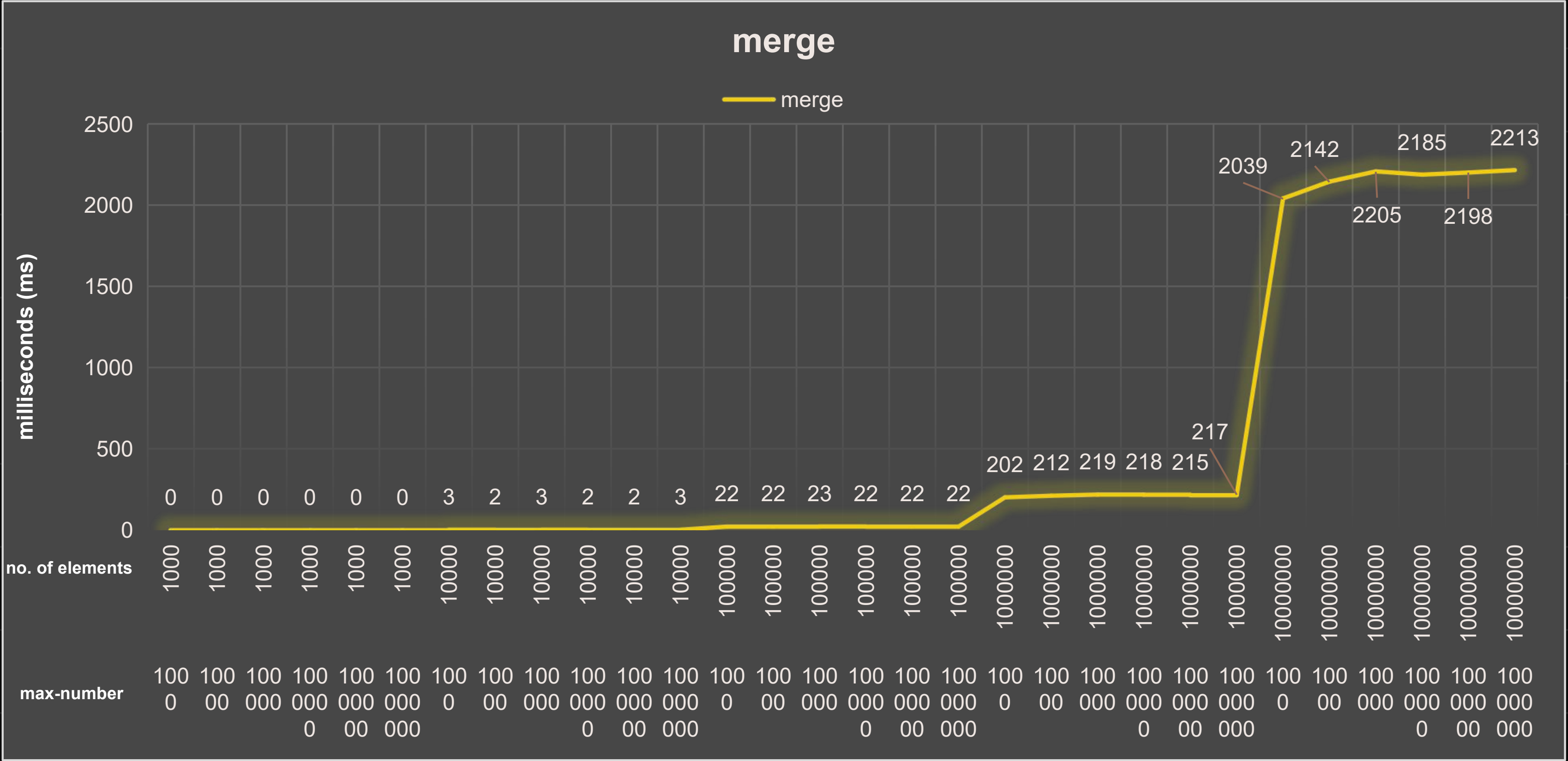
- A merging algorithm, with Big-O(n logn), easy to implement and fast

Worst case	Average case	Best case	Memory	Stable
n logn	n logn	n logn	n	yes

Merge sort


10.000.000 elements
merge
milliseconds (ms)
1960
2130
2178
2258
2236
2168
100
1000
10000
100000
1000000
10000000
max-number


10.000.000 max-number
milliseconds (ms)
0
0
2
22
221
2206
100
1000
10000
100000
1000000
10000000
number of elements



merge
merge
milliseconds (ms)
2500
2000
1500
1000
500
0
0 0 0 0 0 0 3 2 3 2 2 3 22 22 23 22 22 22 202 212 219 218 215 217 2039 2142 2205 2185 2198 2213
no. of elements
1000 1000 1000 1000 1000 1000 10000 10000 10000 10000 10000 10000 100000 100000 100000 100000 100000 100000 1000000 1000000 1000000 1000000 1000000 1000000 10000000 10000000 10000000 10000000 10000000 10000000
max-number
1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000

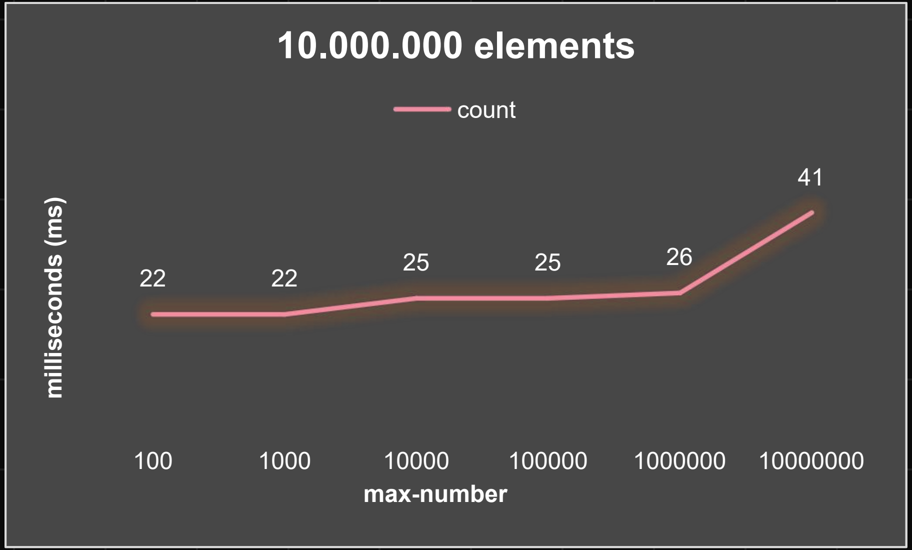
Count sort

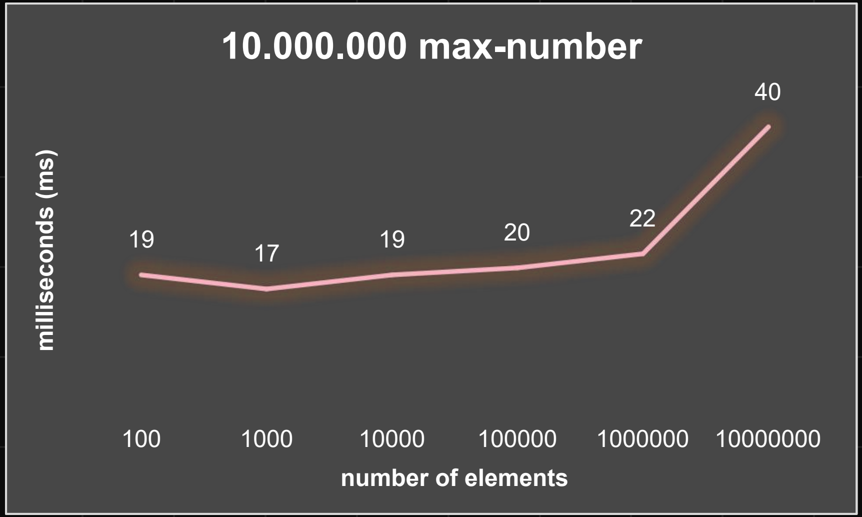
- Based on counting, an very efficient algorithm for unsigned integer numbers. It can be modify to work with negative and floats too.

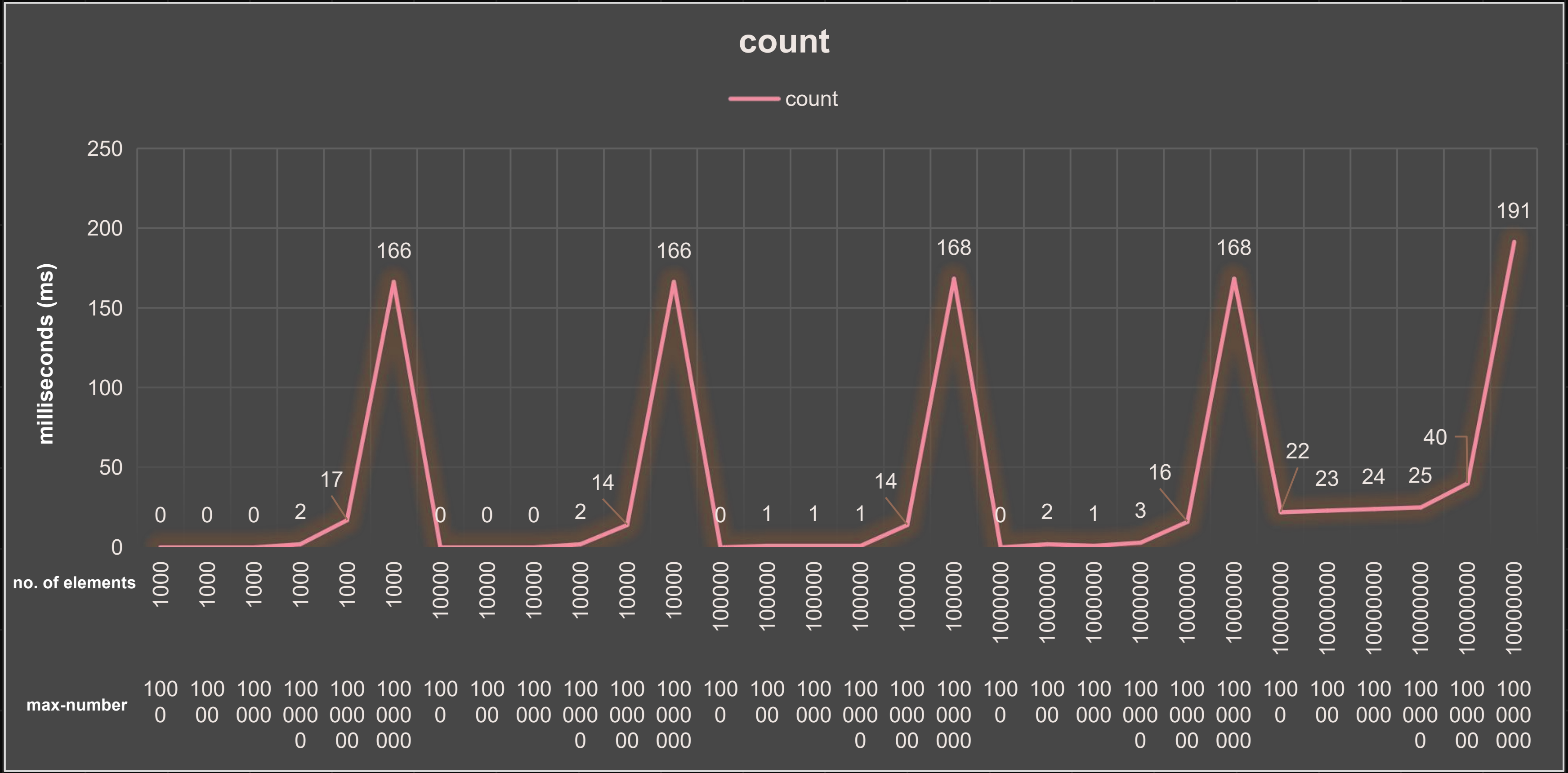
Worst case	Average case	Best case	Memory	Stable
-	n + r	n + r	n + r	yes

- n - represents the number of elements
- r - represents the max-number (range)

Count sort


10.000.000 elements
count
milliseconds (ms)
22
22
25
25
26
41
100
1000
10000
100000
1000000
10000000
max-number


10.000.000 max-number
milliseconds (ms)
19
17
19
20
22
40
100
1000
10000
100000
1000000
10000000
number of elements

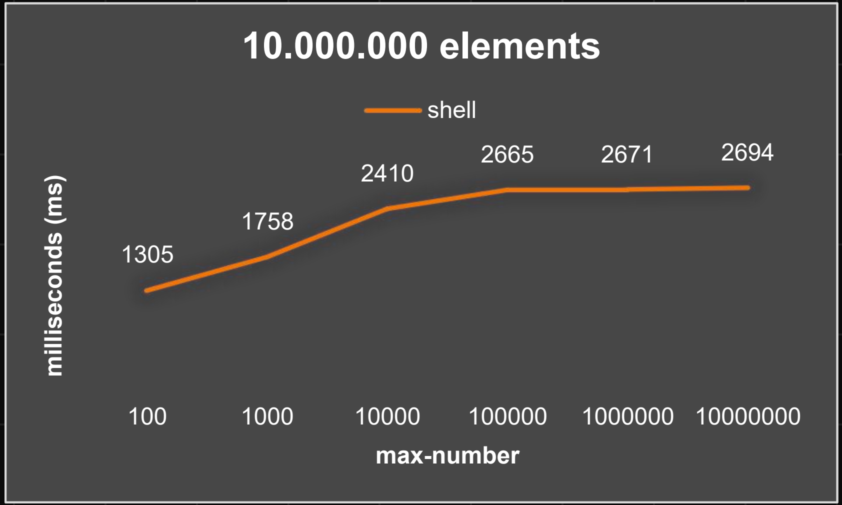

count
count
milliseconds (ms)
250
200
150
100
50
0
166
166
168
168
191
17
14
14
16
22
23
24
25
40
no. of elements
max-number

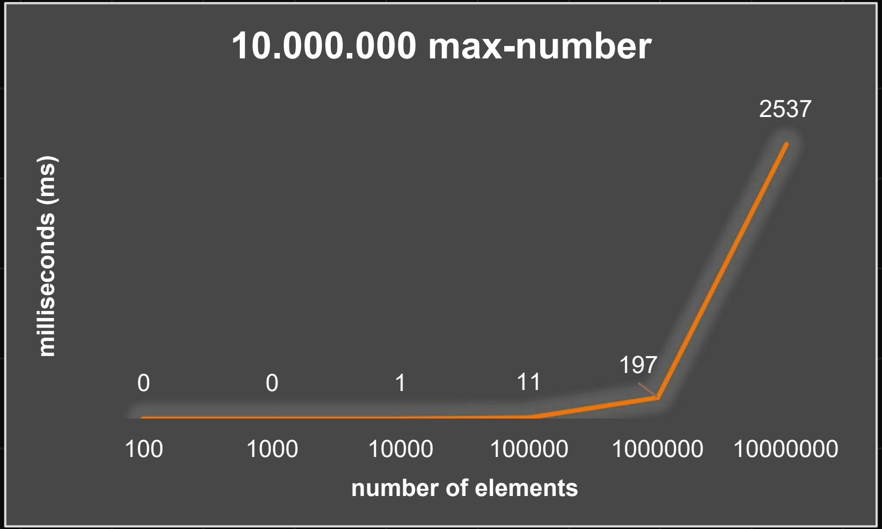
Shell sort

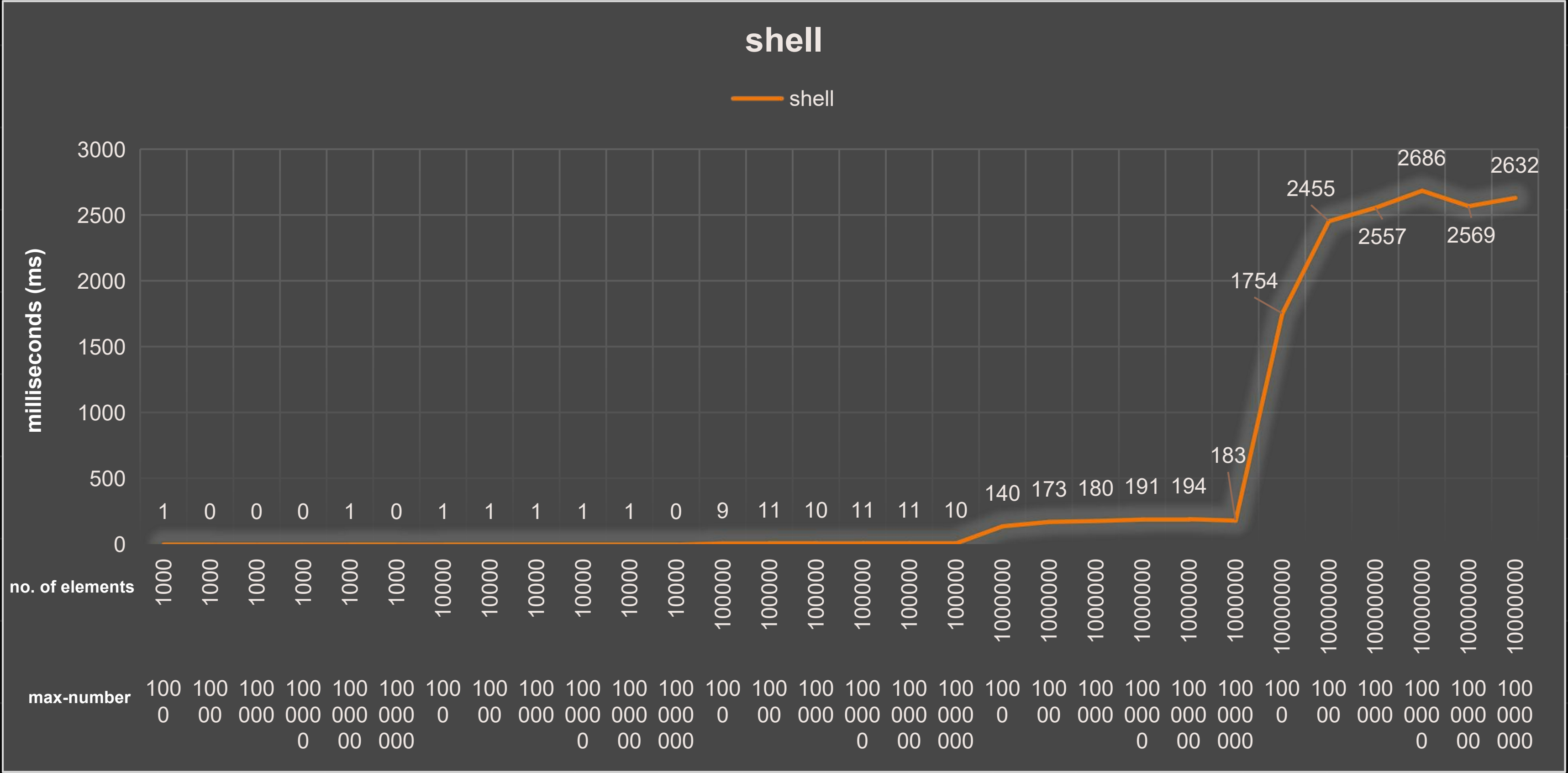
- An insertion algorithm, an improvement to bubble sort, that eliminates more than one inversion at a time.
- My implementation utilise the standard shell sort gap, dividing by 2

Worst case	Average case	Best case	Memory	Stable
n logn	$n^{4/3}$	$n^{3/2}$	1	no

Shell sort


10.000.000 elements
shell
milliseconds (ms)
1305
1758
2410
2665
2671
2694
100
1000
10000
100000
1000000
10000000
max-number


10.000.000 max-number
milliseconds (ms)
0
0
1
11
197
2537
100
1000
10000
100000
1000000
10000000
number of elements

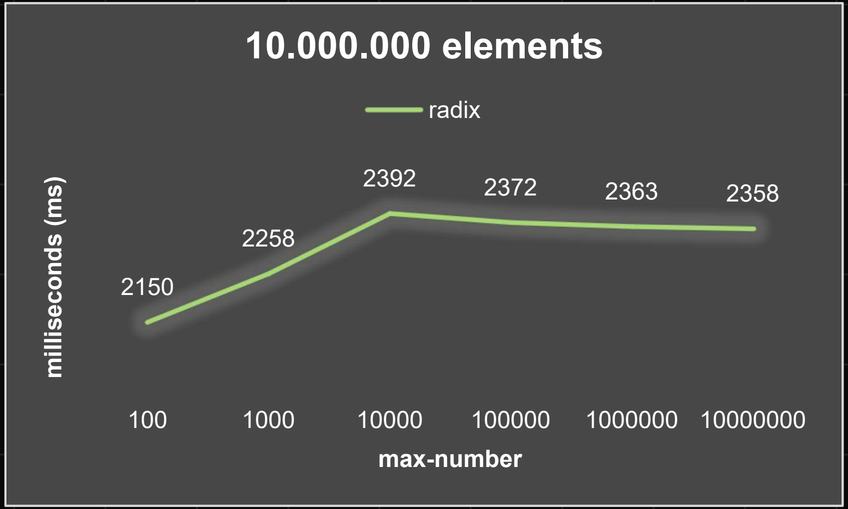

shell
shell
milliseconds (ms)
3000
2500
2000
1500
1000
500
0
1
0
0
0
1
0
1
1
1
1
1
0
9
11
10
11
11
10
140
173
180
191
194
183
1754
2455
2557
2686
2569
2632
no. of elements
1000
1000
1000
1000
1000
1000
10000
10000
10000
10000
10000
10000
100000
100000
100000
100000
100000
100000
1000000
1000000
1000000
1000000
1000000
1000000
10000000
10000000
10000000
10000000
10000000
10000000
max-number
1000
10000
100000
1000000
10000000
100000000
1000
10000
100000
1000000
10000000
100000000
1000
10000
100000
1000000
10000000
100000000
1000
10000
100000
1000000
10000000
100000000
1000
10000
100000
1000000
10000000
100000000

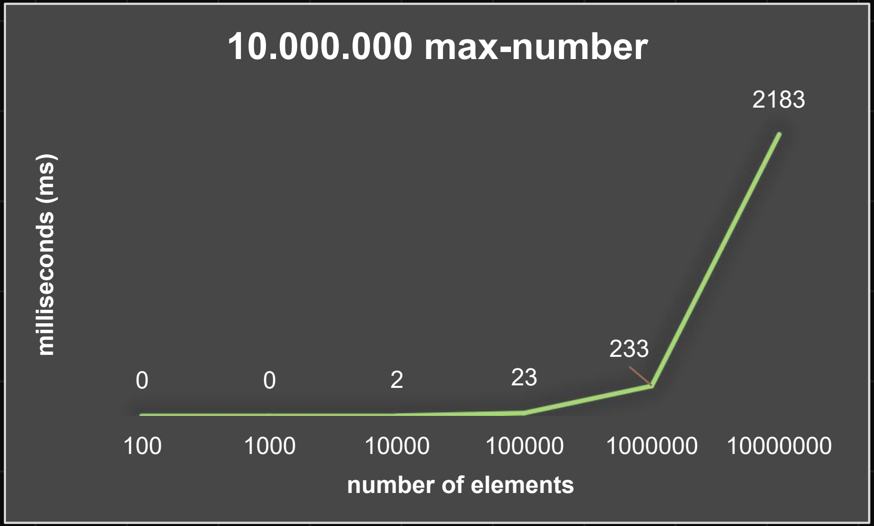
Radix sort

- Based on counting, this algorithm can offer different results based on the base used
- The tests include different bases. It was implemented as a LSD Radix.

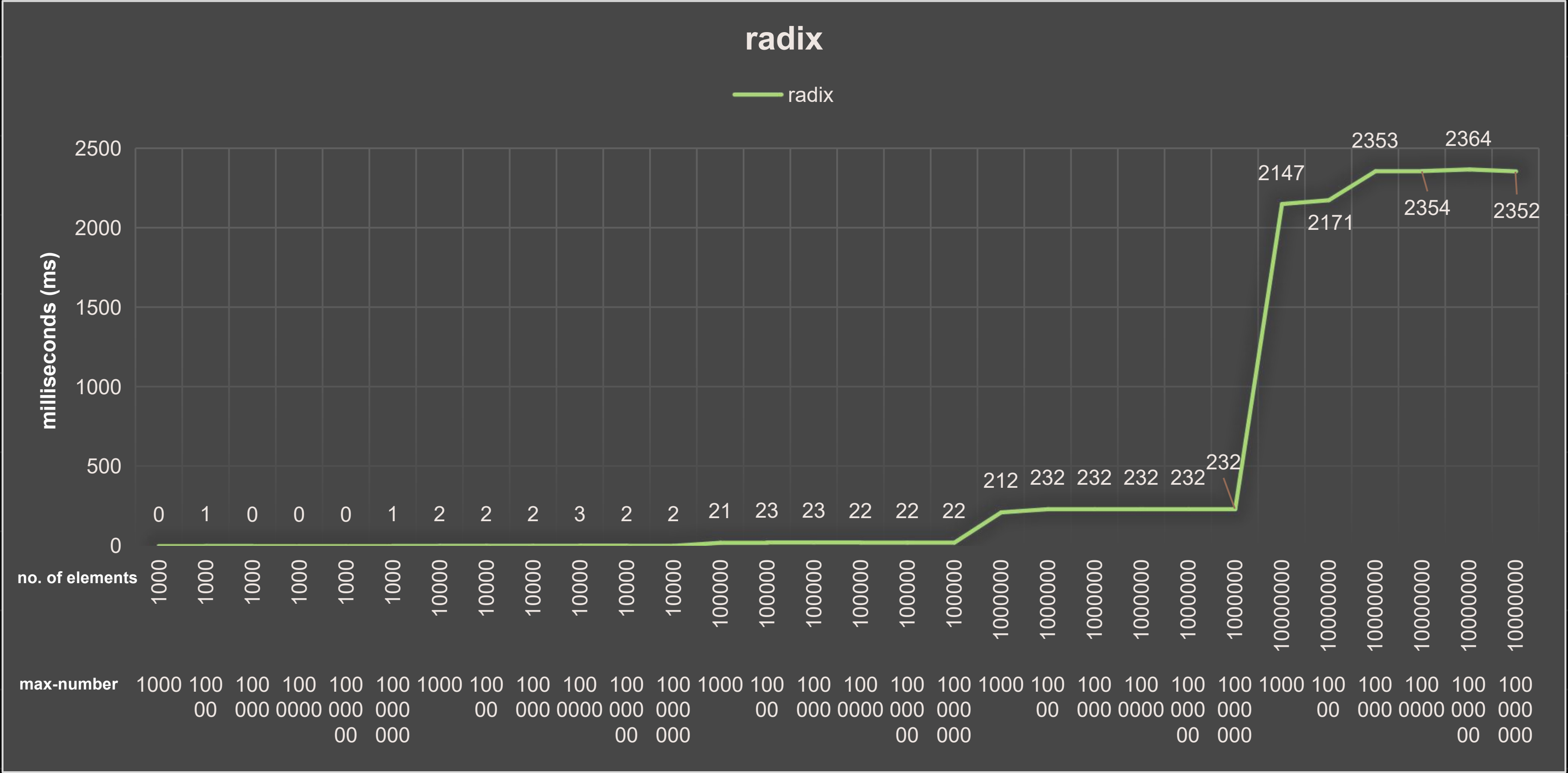
Worst case	Average case	Best case	Memory	Stable
n	n * no_of_digits	n * no_of_digits	$n + 2^{no\_of\_digits}$	yes

Radix sort

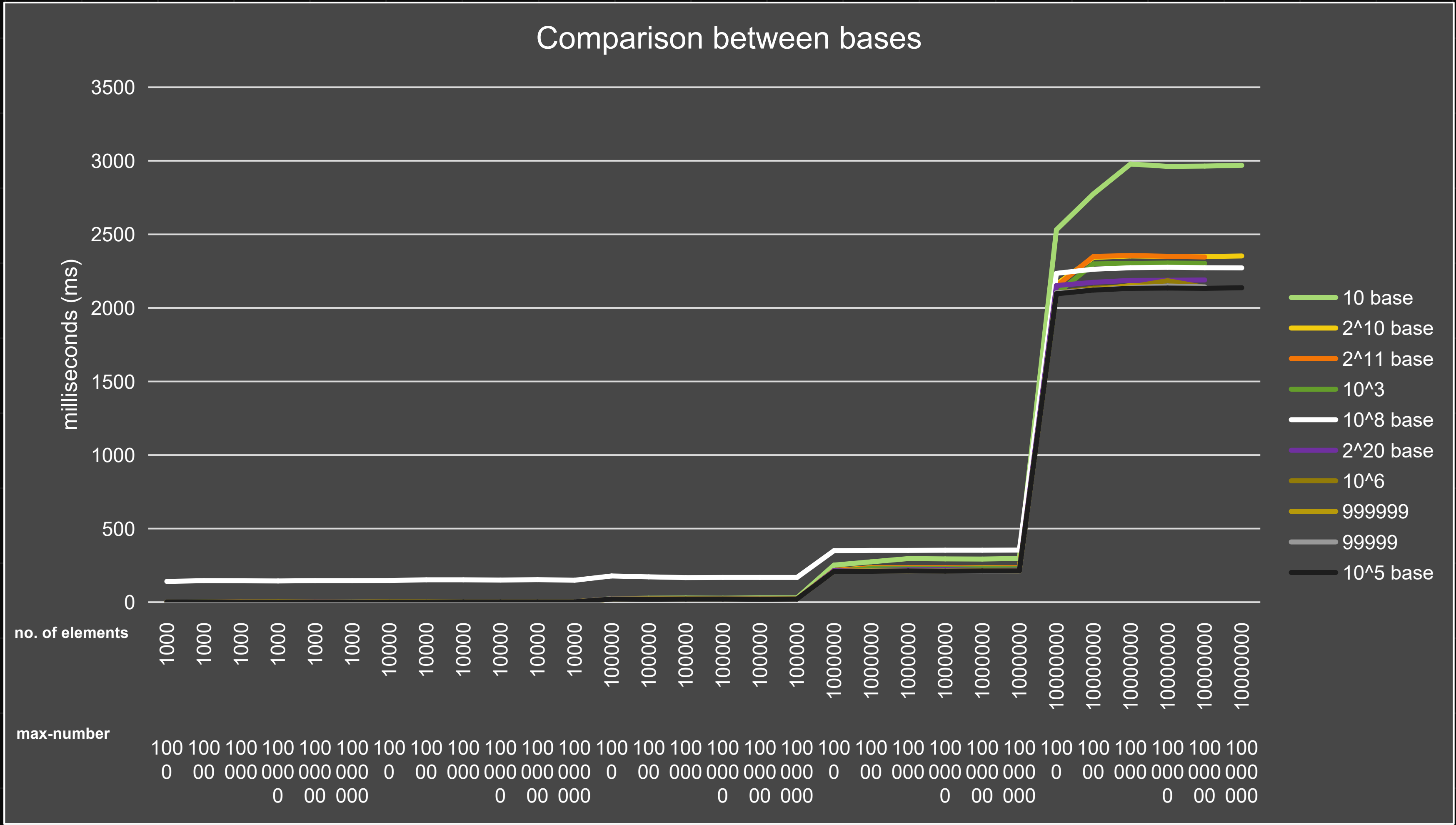




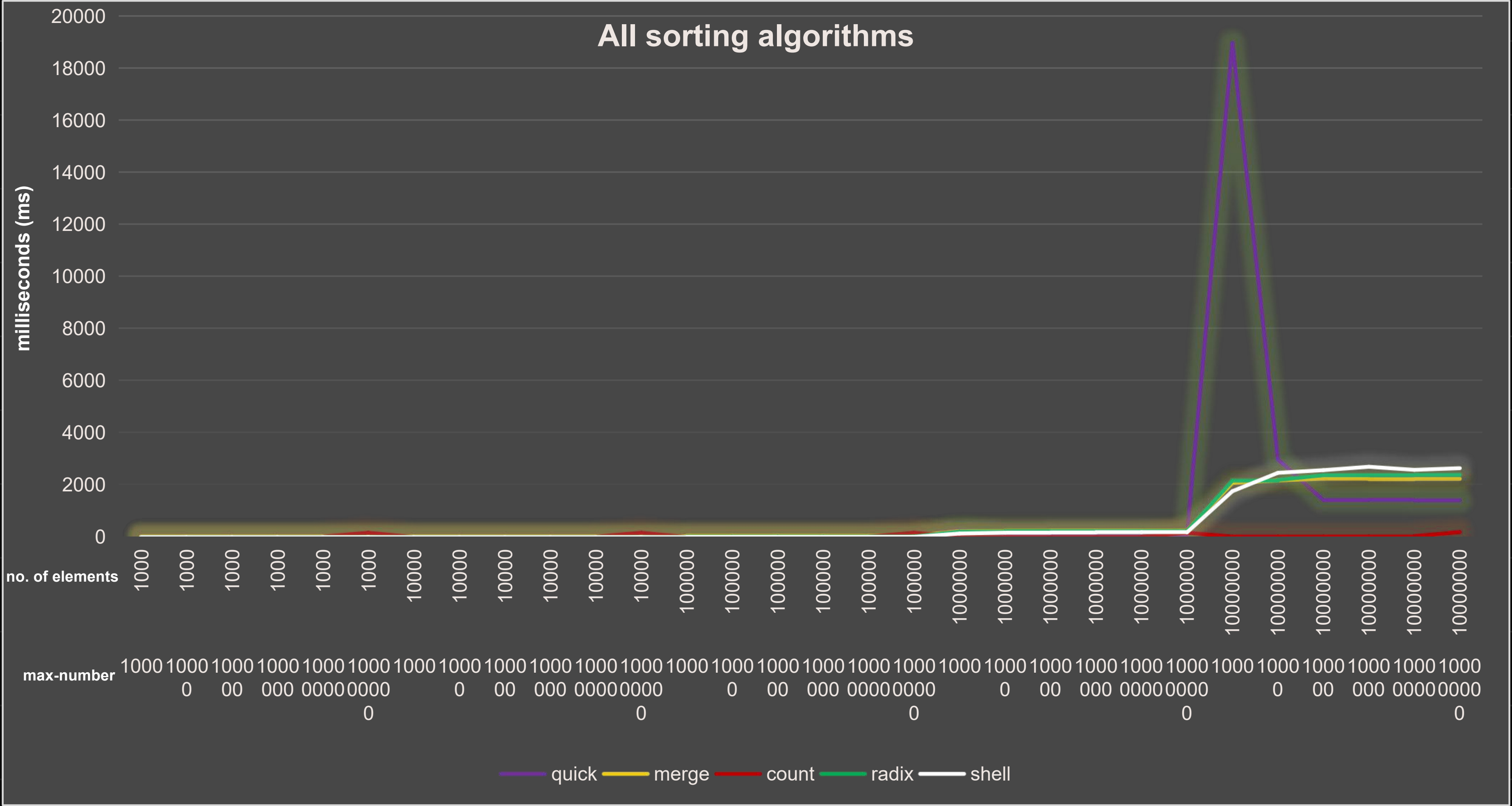
- Here and in the next slide the base raises exponentially from 2^{10} to 2^{15} by 2 on every step



radix
radix
milliseconds (ms)
2500
2000
1500
1000
500
0
0 1 0 0 0 1 2 2 2 3 2 2 21 23 23 22 22 22 212 232 232 232 232 232 2147 2171 2353 2354 2364 2352
no. of elements
1000 1000 1000 1000 1000 1000 10000 10000 10000 10000 10000 10000 100000 100000 100000 100000 100000 100000 1000000 1000000 1000000 1000000 1000000 1000000 10000000 10000000 10000000 10000000 10000000 10000000
max-number
1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000 1000 10000 100000 1000000 10000000 100000000



Comparison between bases
milliseconds (ms)
3500
3000
2500
2000
1500
1000
500
0
no. of elements
max-number
10 base
2^10 base
2^11 base
10^3
10^8 base
2^20 base
10^6
999999
99999
10^5 base



All sorting algorithms
milliseconds (ms)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
no. of elements
1000
10000
100000
1000000
10000000
max-number
1000
10000
100000
1000000
10000000
100000000
quick
merge
count
radix
shell

Bibliography

https://en.wikipedia.org/wiki/Sorting_algorithm

Thank you for your patience!