Christopher Kang

CS241

2/12/2016

Professor Rodriguez

Project 2 Sorting Algorithm Analysis

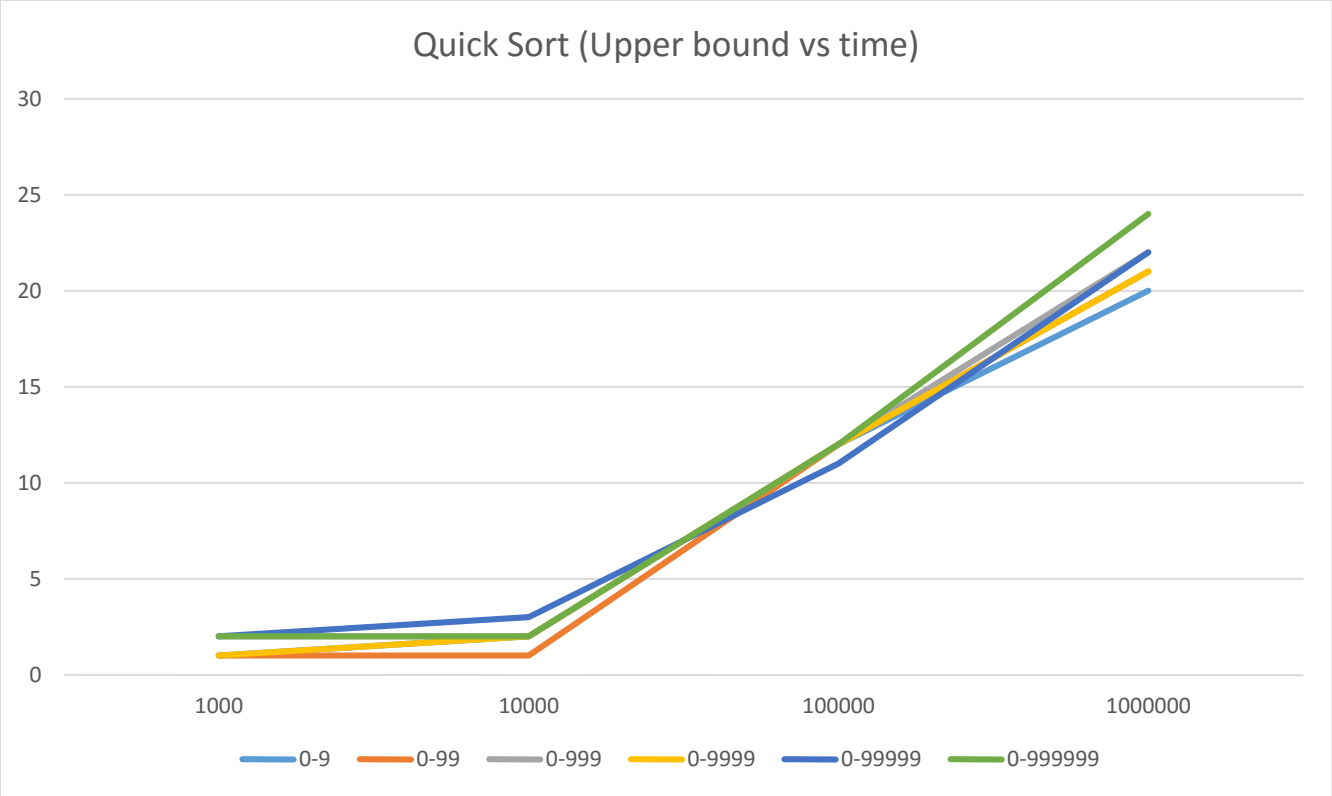
In this project, we were asked to run and analyze 8 different sorting algorithms. The sorting algorithms are: Bubble sort, Heap sort, Quick Sort, Selection sort, Insertion sort, Radix sort, Counting sort and Merge sort. My finding was no surprises, Heap sort with run time complexity O(n log n) turned out to be the superior sorting algorithm. Merge sort and quick sort were really close as well. What I did not expect was that Counting sort and Radix sort were really fast because I did not have too much hope with liner run time complexity of O (N + k). On the other hand, algorithms with average run time complexity O (N^2) such as selection sort and insertion sort took a lot longer to complete sorting when the number of elements are large, and bubble sort was the most painful to use above all other sorting algorithms.

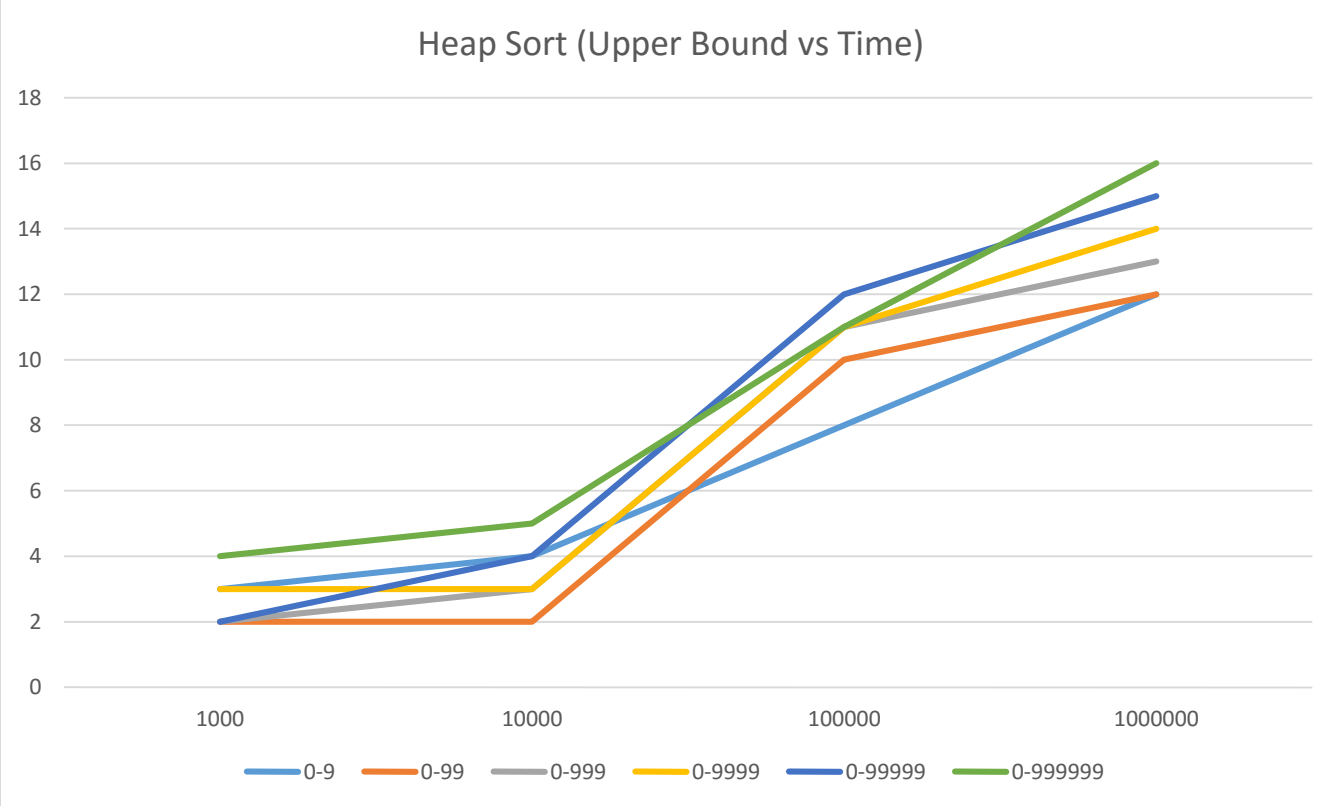
According to the data, we can see very little impact the upper bound of elements have to most of our sorting algorithms. For example, it took 28 seconds to sort 10000 elements from 0-9 elements and 30 seconds to sort 10000 elements in the range of 0-999999. However, we do see a big difference (in terms of % difference) when it comes to Counting sort and Radix sort. Because the run time complexity for these 2 algorithms are O (N + k), and we know that k is the range of elements. It is no surprise to see the time spent went up very quickly as we sort elements with higher range. Another interesting thing I observed is that Insertion sort is a bit slower than Merge

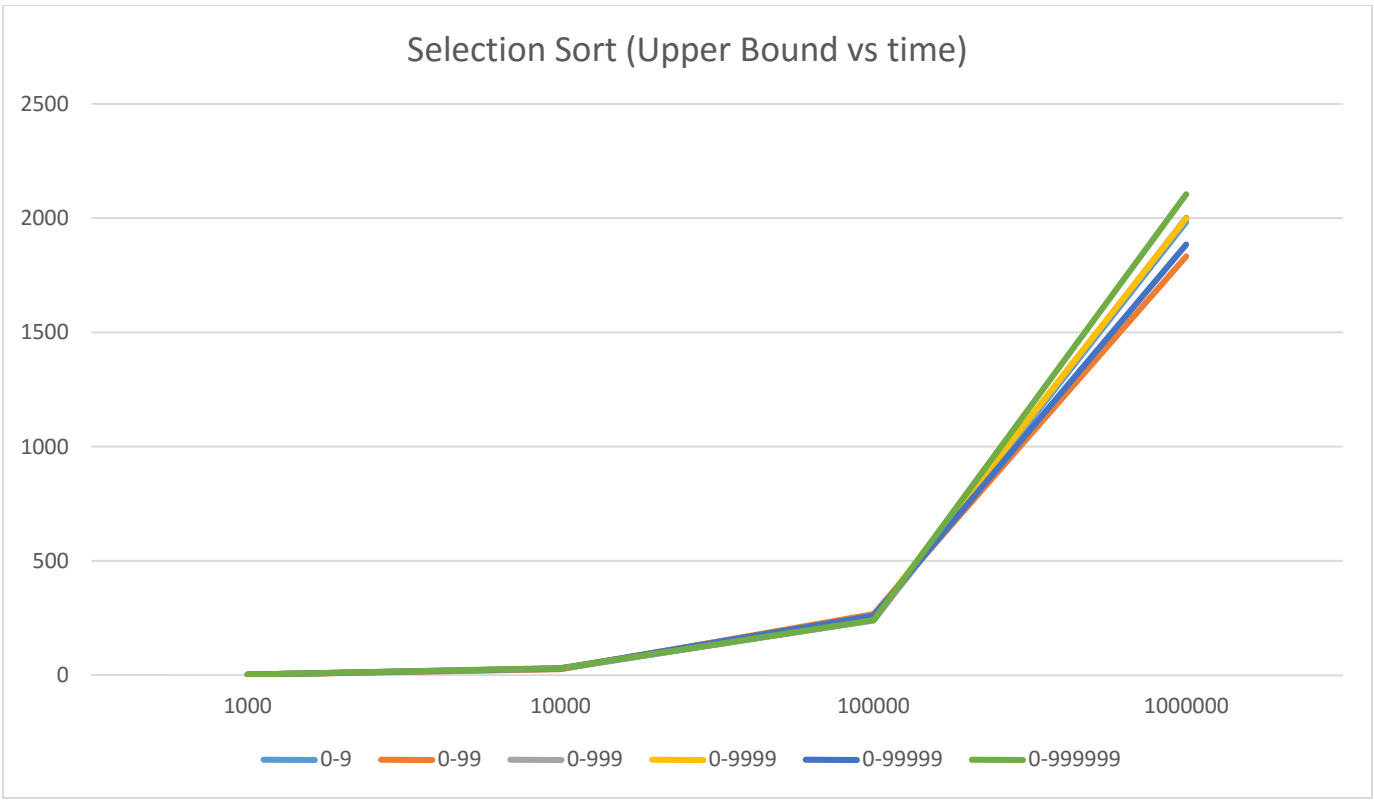
sort. The two have the same average case of run time complexity, but knowing how Insertion sort tends to do more work in general, it's satisfying to see the actual difference in action,

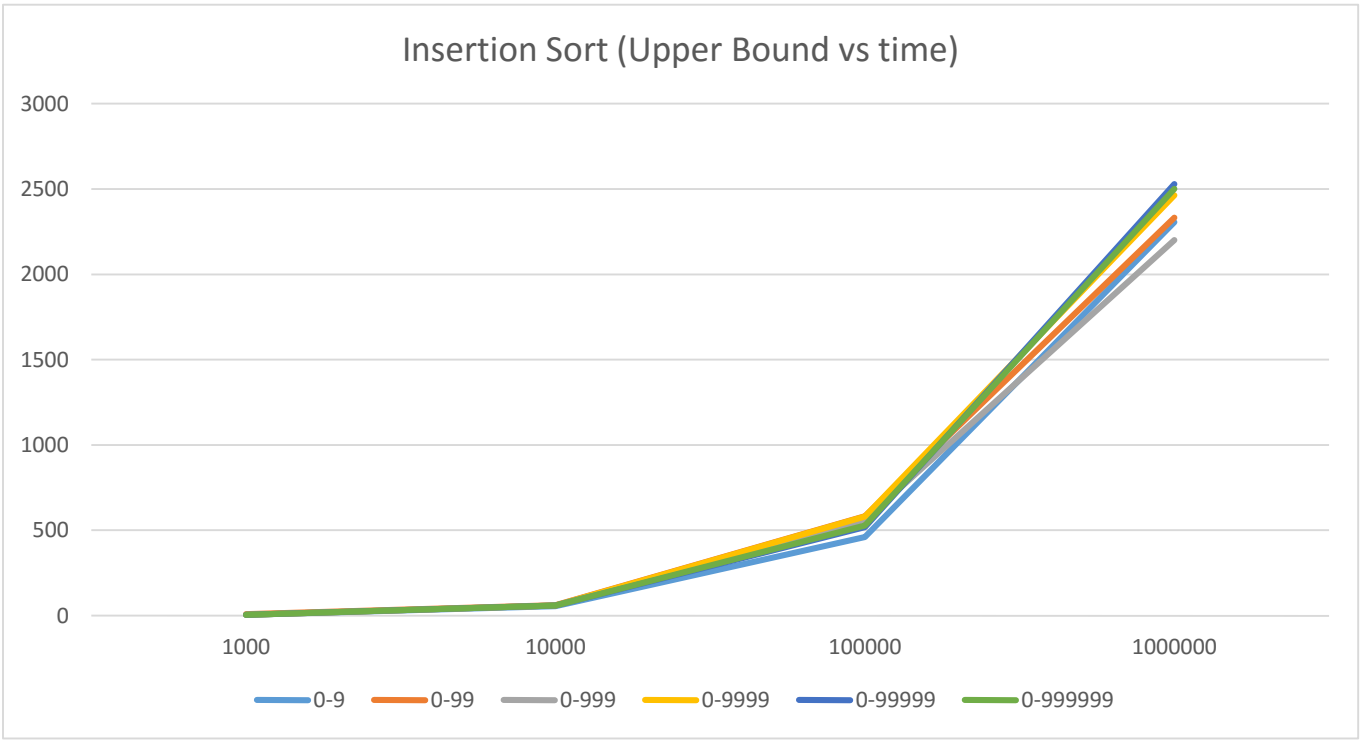
Something I learned from this experiment is the importance of average run time complexity. Because we always care about the best and worst cases of run time complexity, but in real life world which is not perfect (i.e. we actually have unsorted lists to sort), average run time complexity is just as important factor to consider when writing our programs. We can thus be more confident in Quick Sort even though it can possibly have worst case of O (n^2).

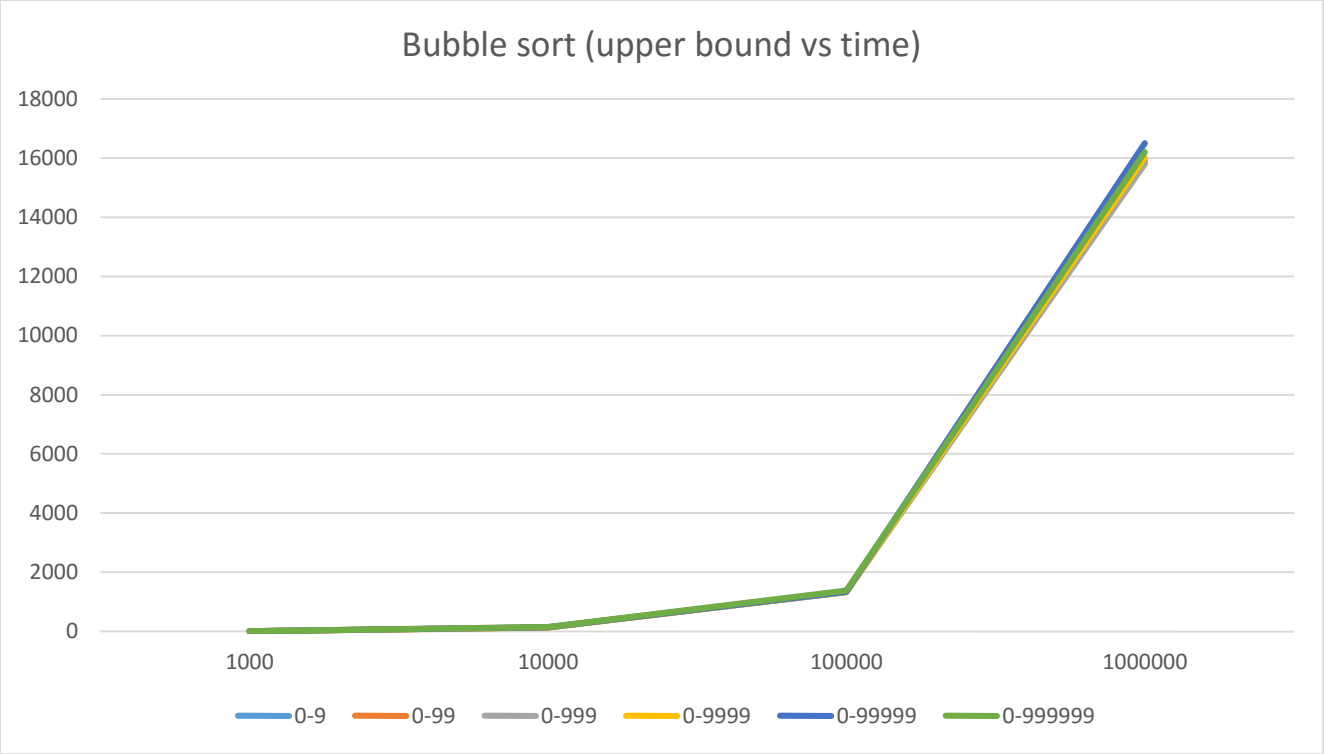
In conclusion, this project has helped me visualizing the run time complexity of different sorting algorithms. I now have a good idea of what sorting algorithm I should choose and what to avoid upon situation in the future. I'm also confident enough to say that I will certainly use Bubble sort…when coding for my worst enemy. I would also be sure to do similar testing to sorting algorithms in the future ensure they are true worthy of being implemented.

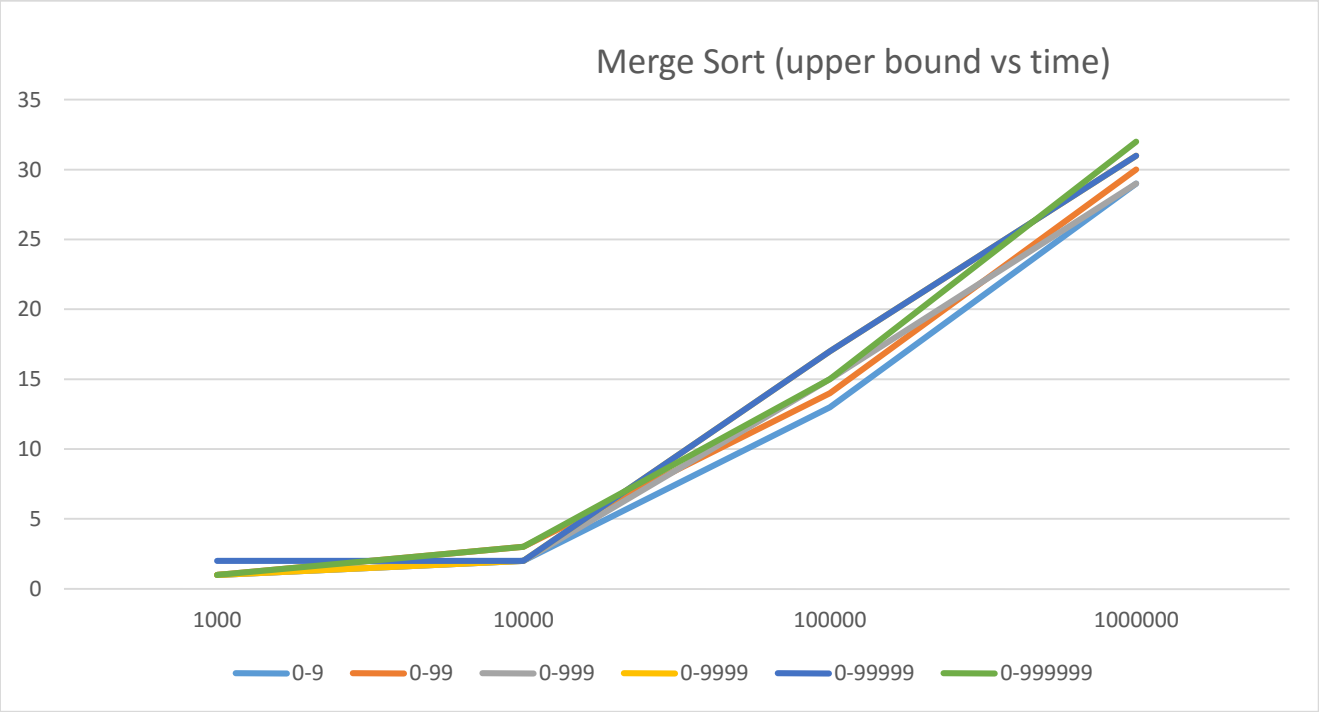

Quick Sort (Upper bound vs time)
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999

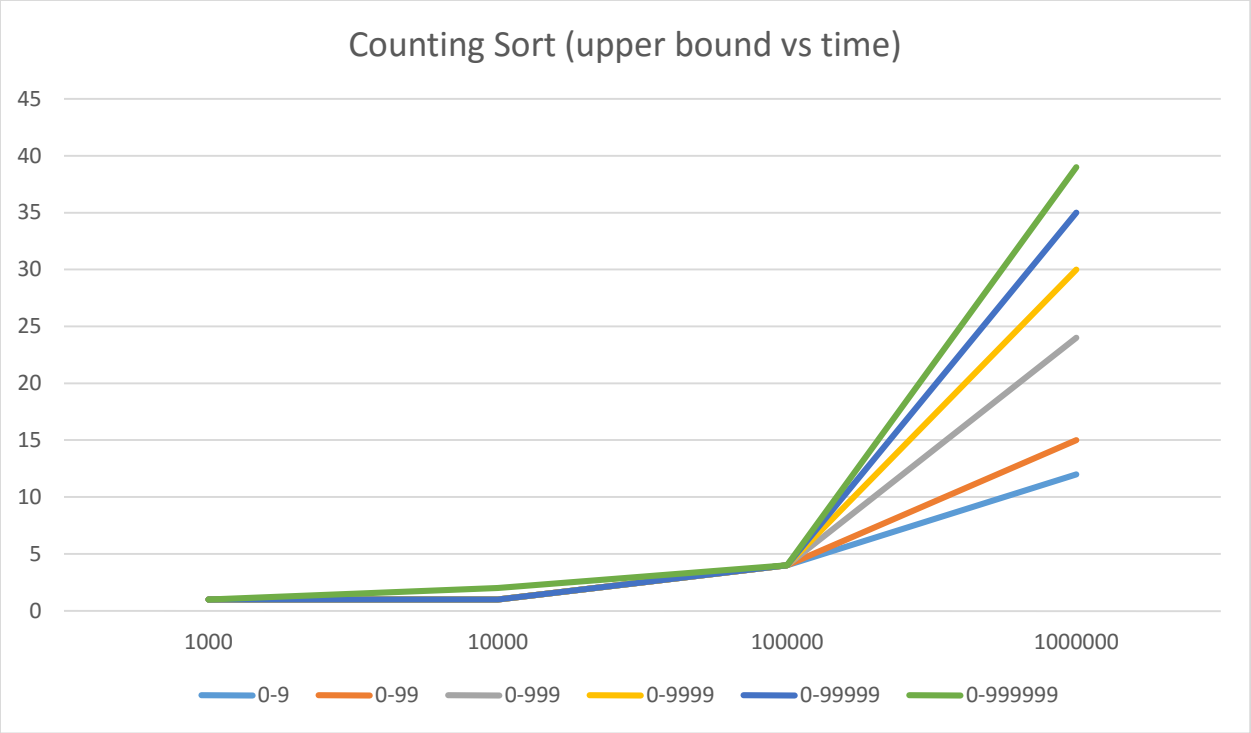

Heap Sort (Upper Bound vs Time)
18
16
14
12
10
8
6
4
2
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999

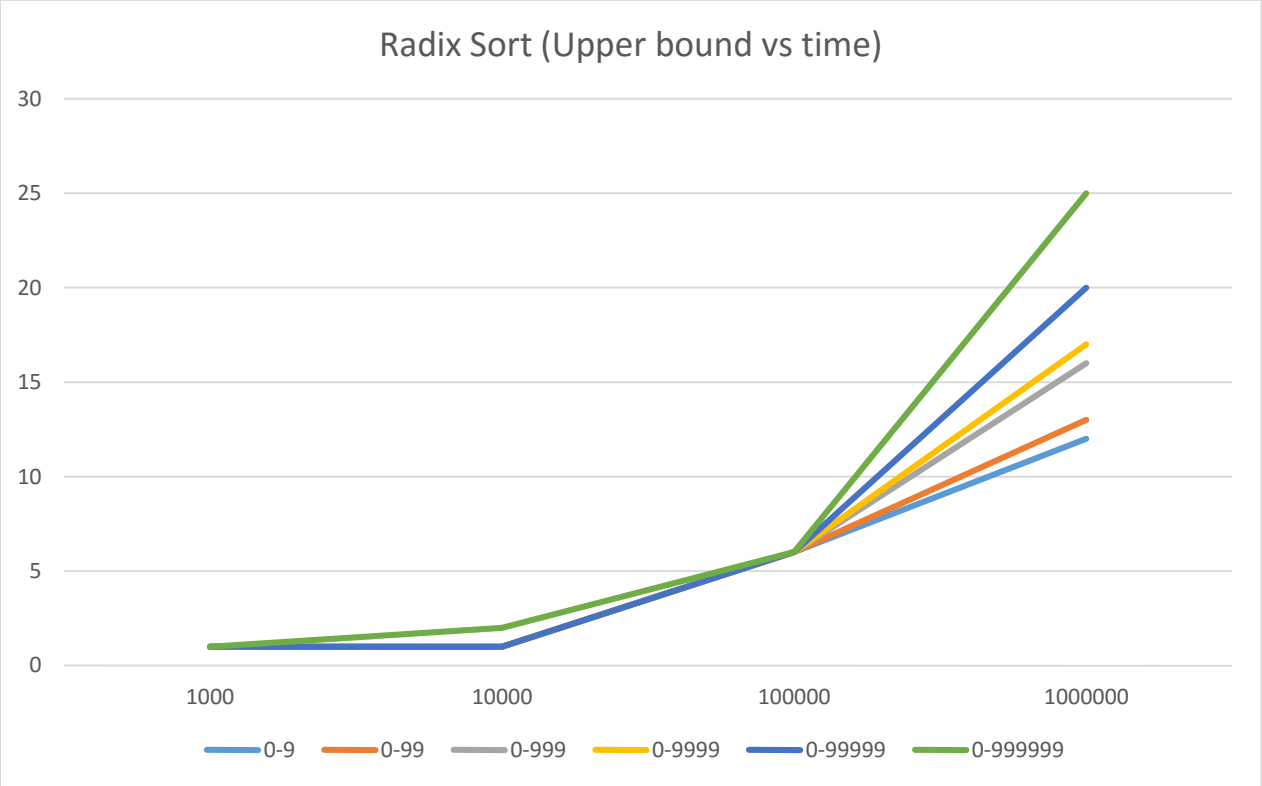

Selection Sort (Upper Bound vs time)
2500
2000
1500
1000
500
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999

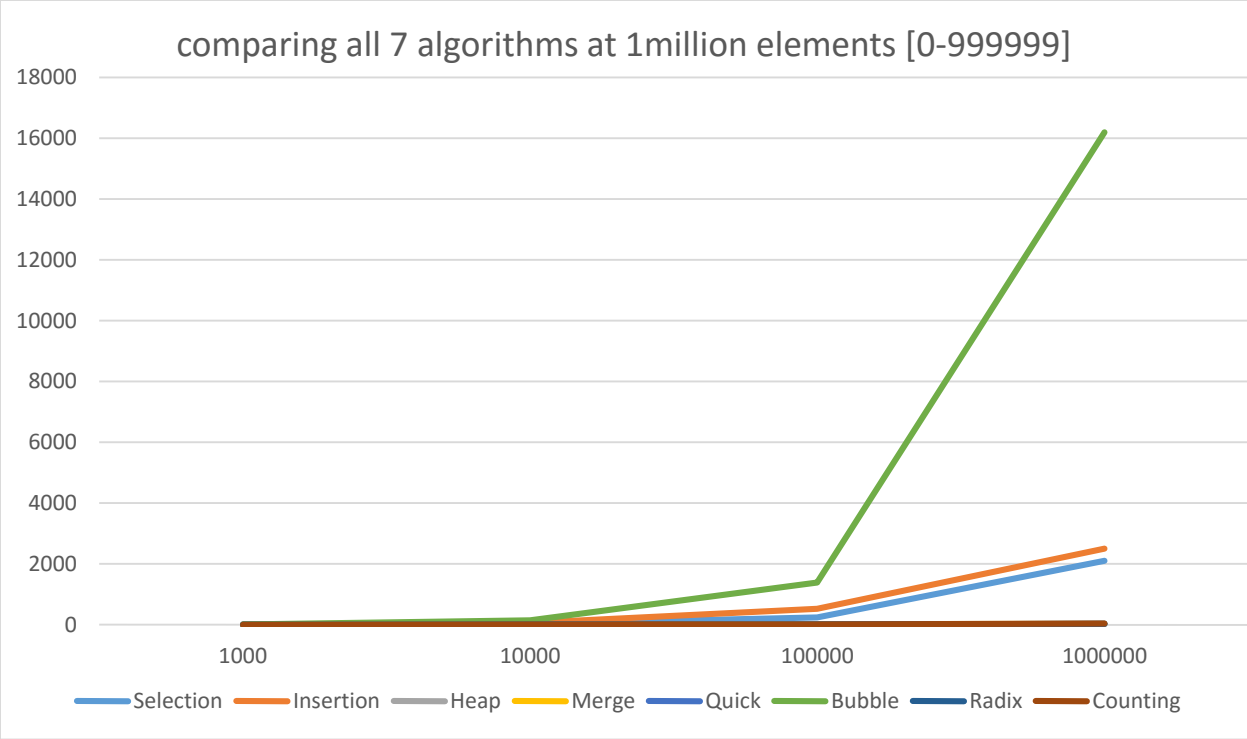

Insertion Sort (Upper Bound vs time)
3000
2500
2000
1500
1000
500
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Bubble sort (upper bound vs time)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Merge Sort (upper bound vs time)
35
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Counting Sort (upper bound vs time)
45
40
35
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Radix Sort (Upper bound vs time)
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


comparing all 7 algorithms at 1million elements [0-999999]
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1000
10000
100000
1000000
Selection
Insertion
Heap
Merge
Quick
Bubble
Radix
Counting

Selection

	1000	10000	100000	1000000
0-9	3	28	242	1982
0-99	2	27	267	1833
0-999	3	30	239	2003
0-9999	2	29	258	1998
0-99999	2	29	261	1884
0-999999	2	30	241	2104

insertion

	1000	10000	100000	1000000
0-9	5	56	460	2305
0-99	8	60	583	2333
0-999	5	59	562	2201
0-9999	5	59	581	2464
0-99999	5	58	518	2530
0-999999	5	58	528	2501

Heap

	1000	10000	100000	1000000
0-9	3	4	8	12
0-99	2	2	10	12
0-999	2	3	11	13
0-9999	3	3	11	14
0-99999	2	4	12	15
0-999999	4	5	11	16

Merge

	1000	10000	100000	1000000
0-9	1	2	13	29
0-99	1	3	14	30
0-999	1	2	15	29
0-9999	1	2	17	31
0-99999	2	2	17	31
0-999999	1	3	15	32

quick

	1000	10000	100000	1000000
0-9	1	2	12	20

0-99	1	1	12	21
0-999	2	2	12	22
0-9999	1	2	12	21
0-99999	2	3	11	22
0-999999	2	2	12	24

Bubble

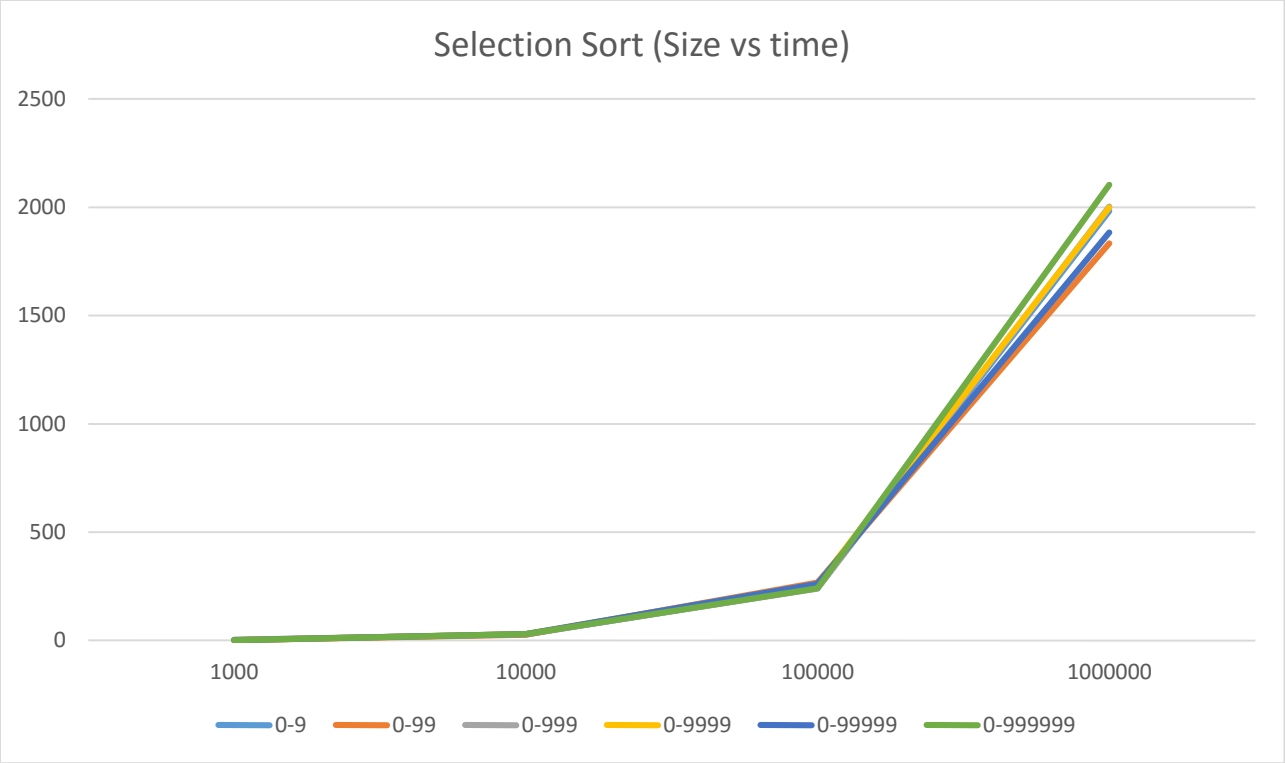
	1000	10000	100000	1000000
0-9	6	140	1354	15830
0-99	5	133	1322	15920
0-999	6	145	1327	15800
0-9999	6	124	1335	16013
0-99999	5	140	1338	16502
0-999999	6	146	1385	16203

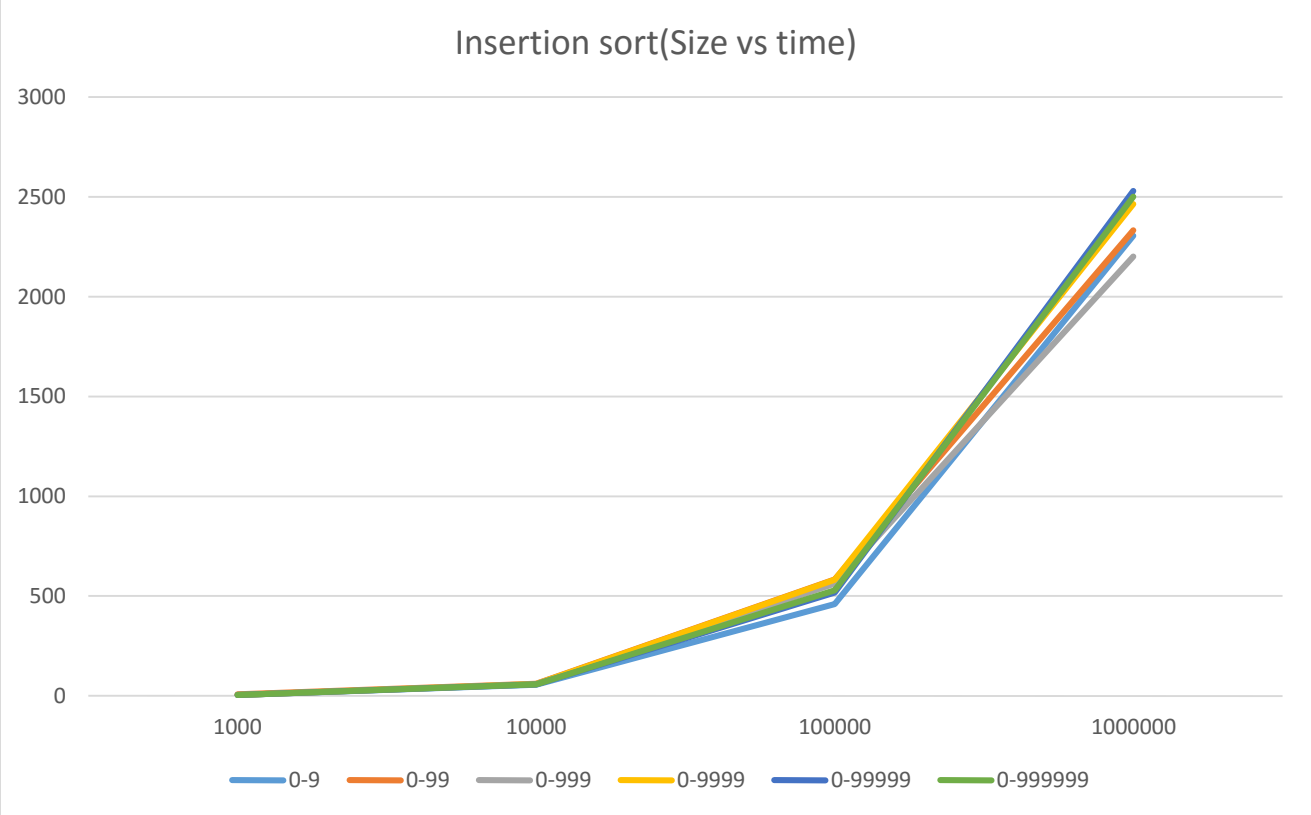
Counting

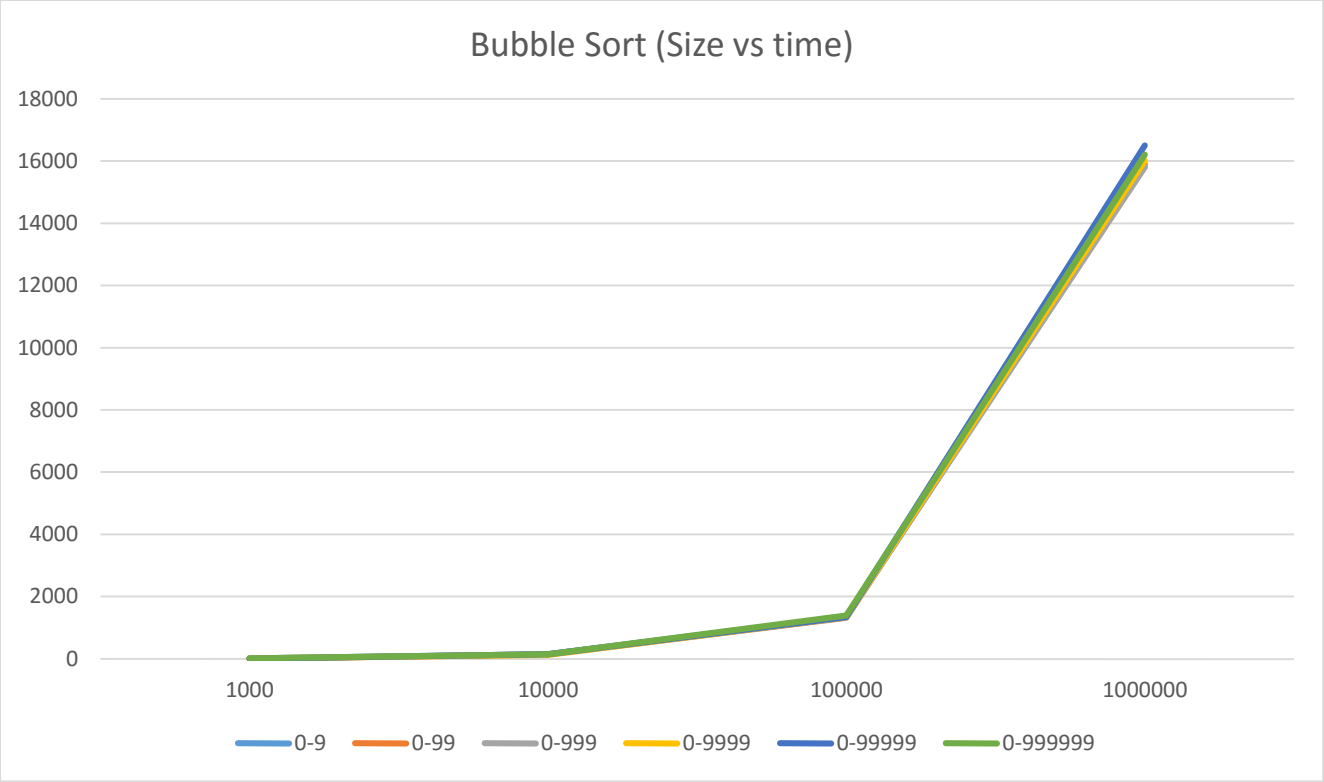
	1000	10000	100000	1000000
0-9	1	1	4	12
0-99	1	1	4	15
0-999	1	1	4	24
0-9999	1	1	4	30
0-99999	1	1	4	35
0-999999	1	2	4	39

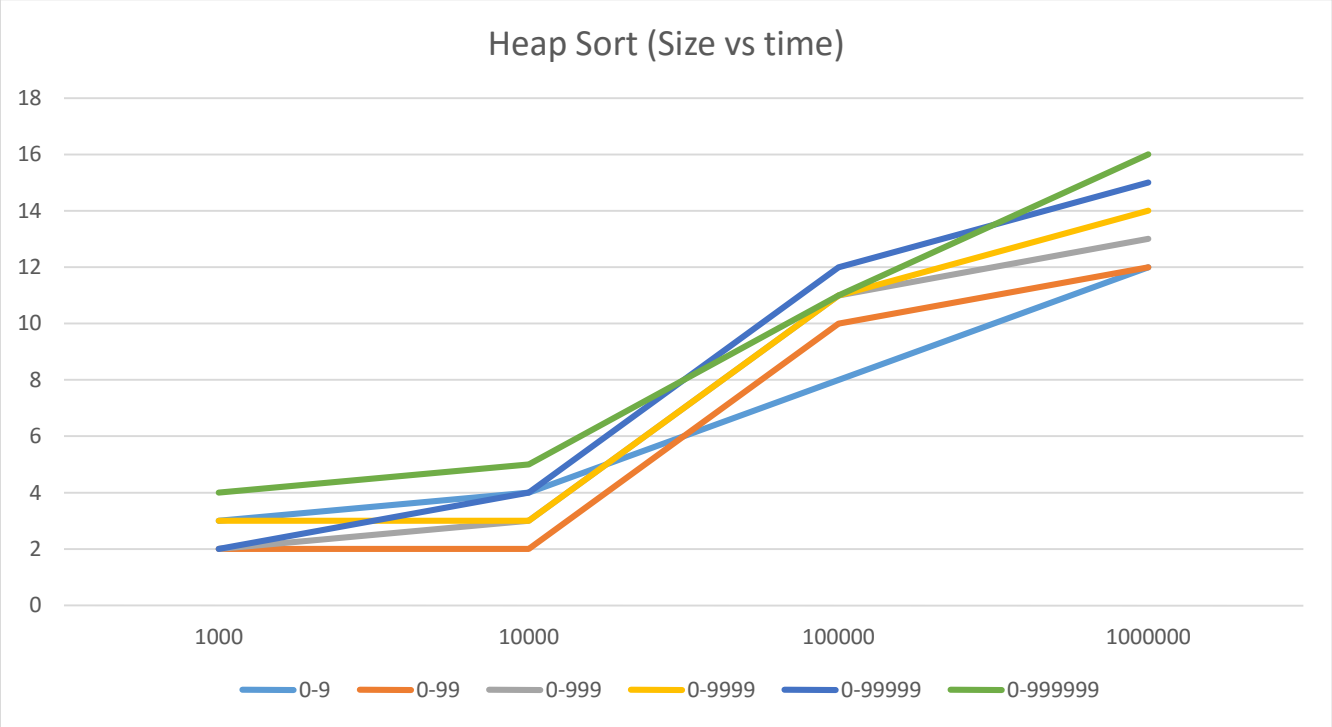
Radix

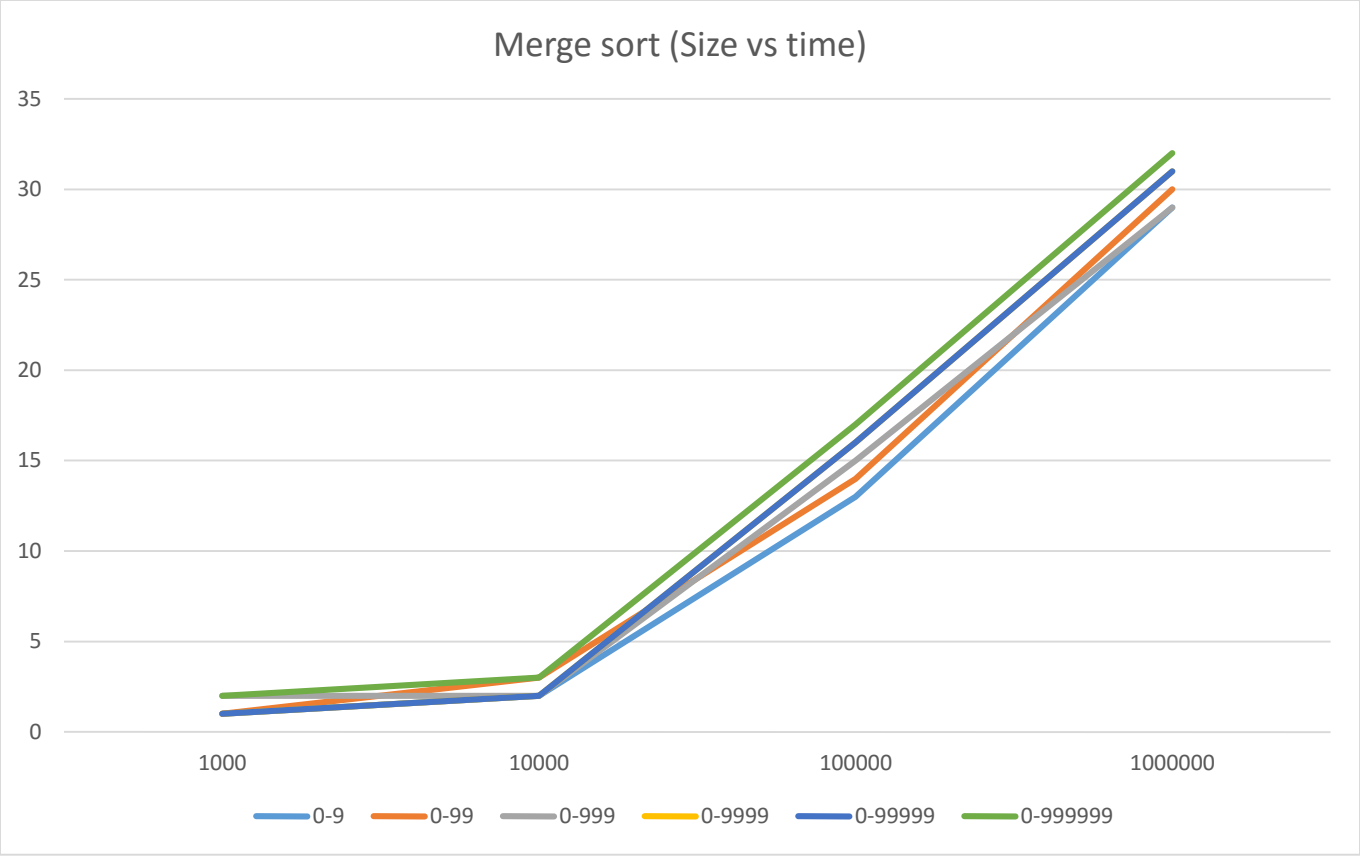
	1000	10000	100000	1000000
0-9	1	1	6	12
0-99	1	1	6	13
0-999	1	1	6	16
0-9999	1	1	6	17
0-99999	1	1	6	20
0-999999	1	2	6	25

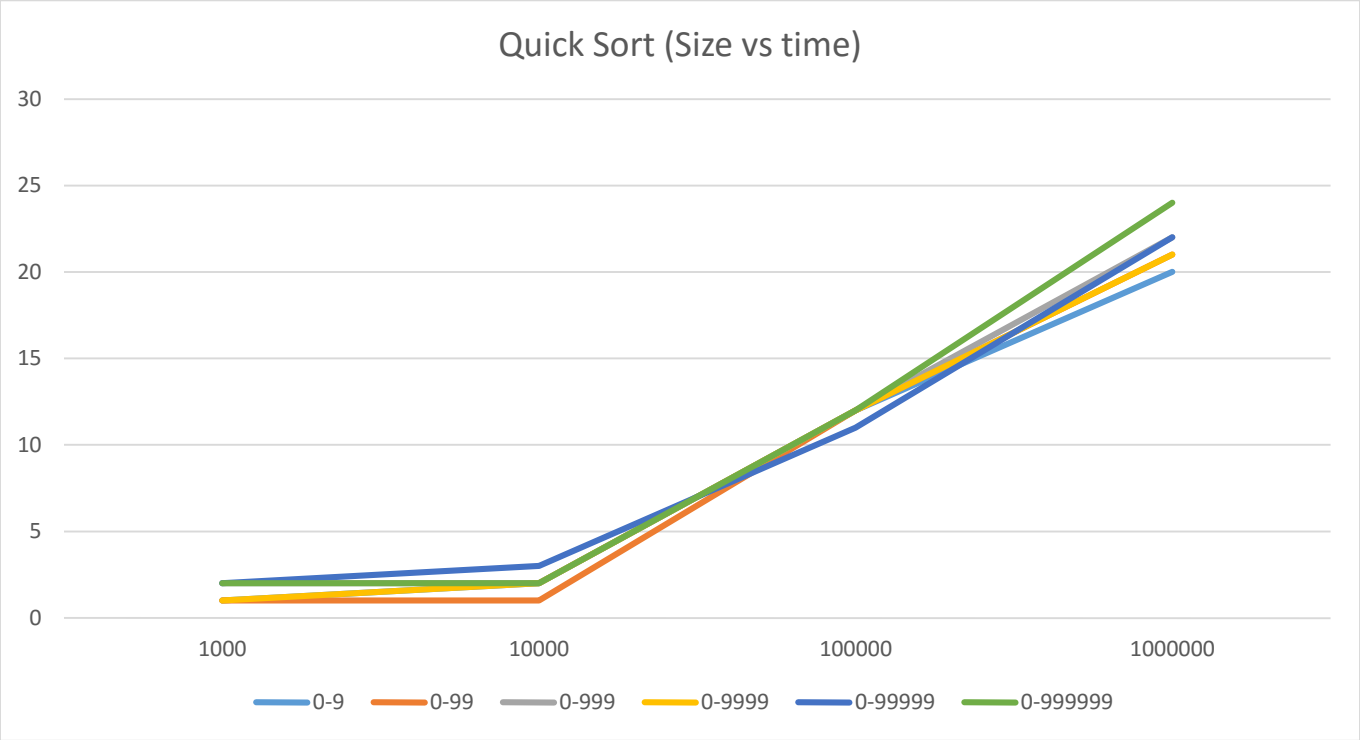

Selection Sort (Size vs time)
2500
2000
1500
1000
500
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999

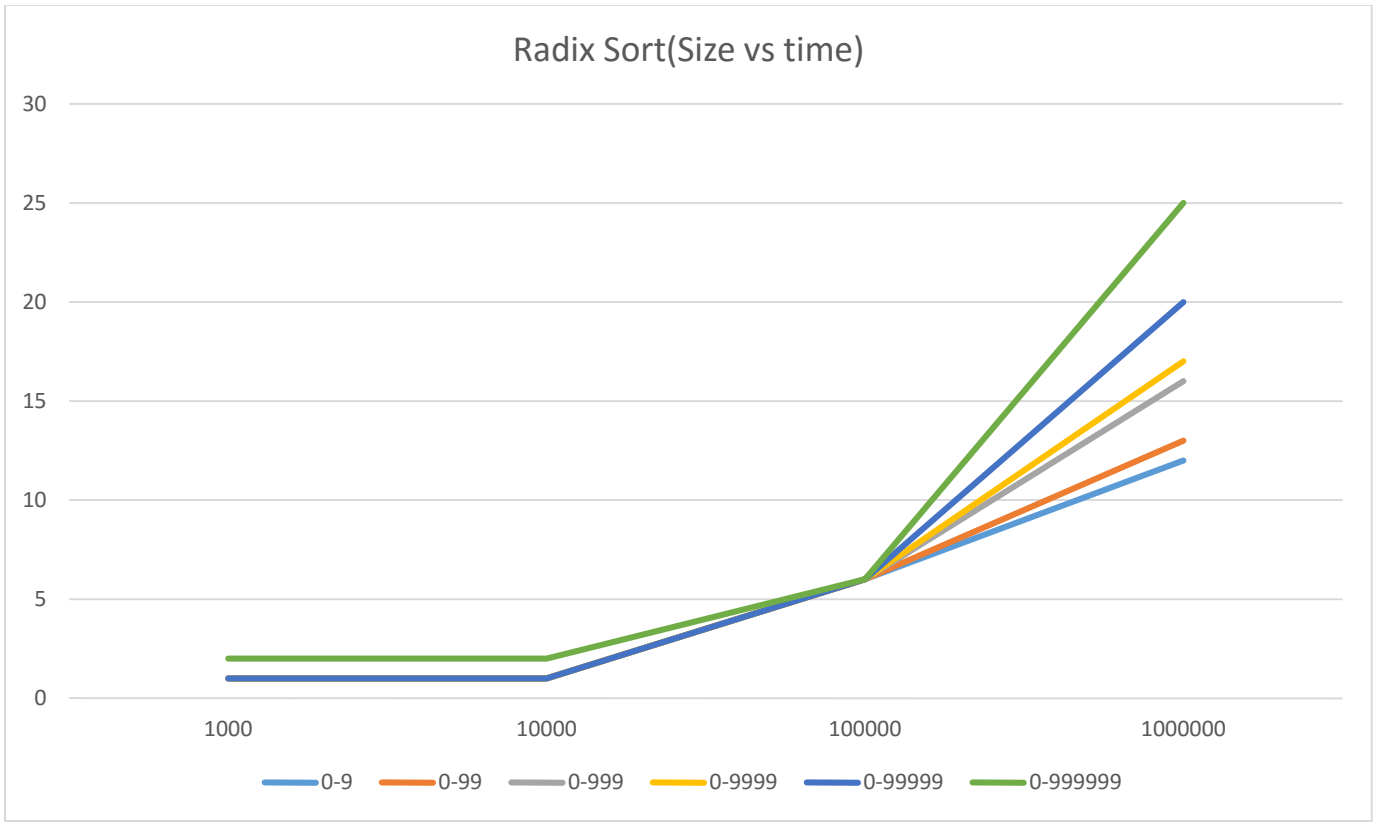

Insertion sort(Size vs time)
3000
2500
2000
1500
1000
500
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999

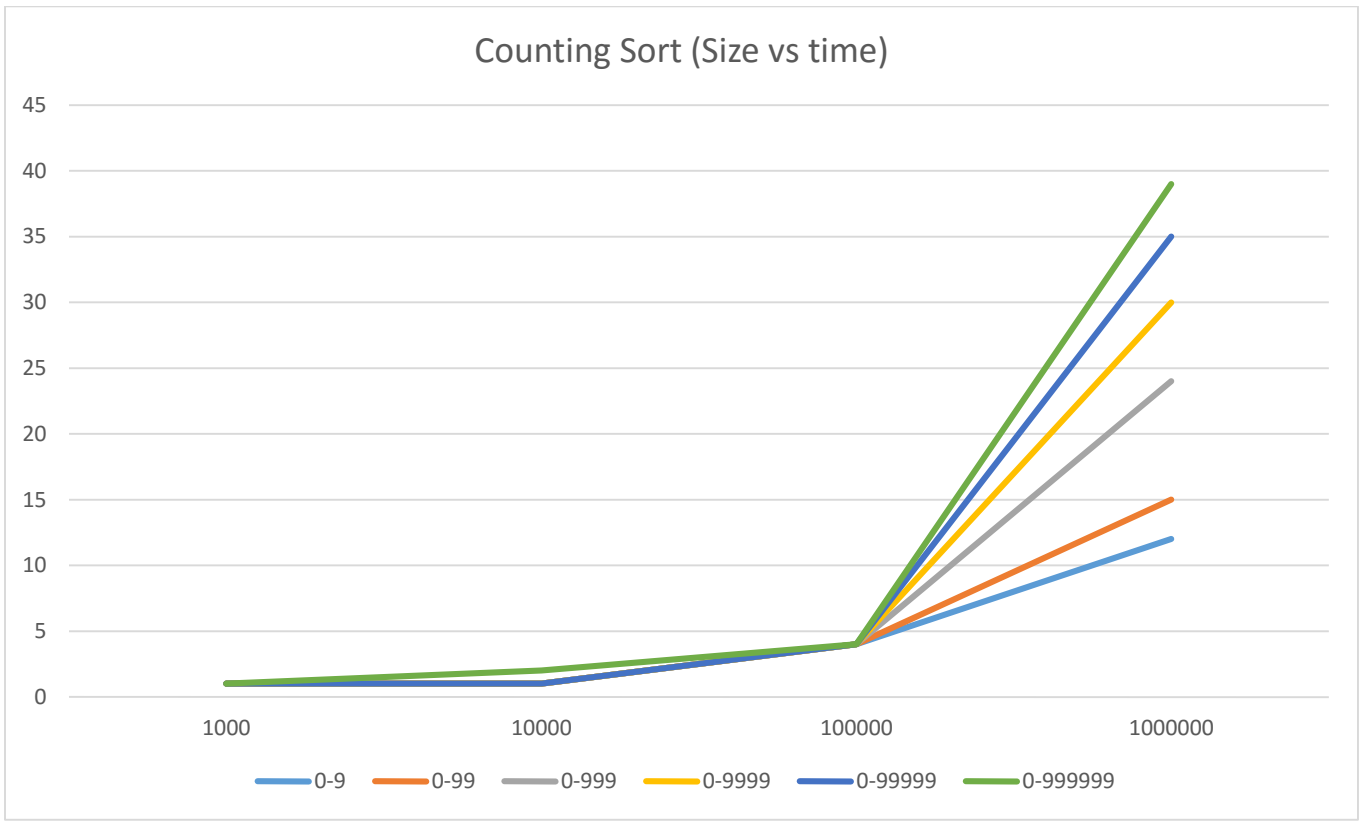

Bubble Sort (Size vs time)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999

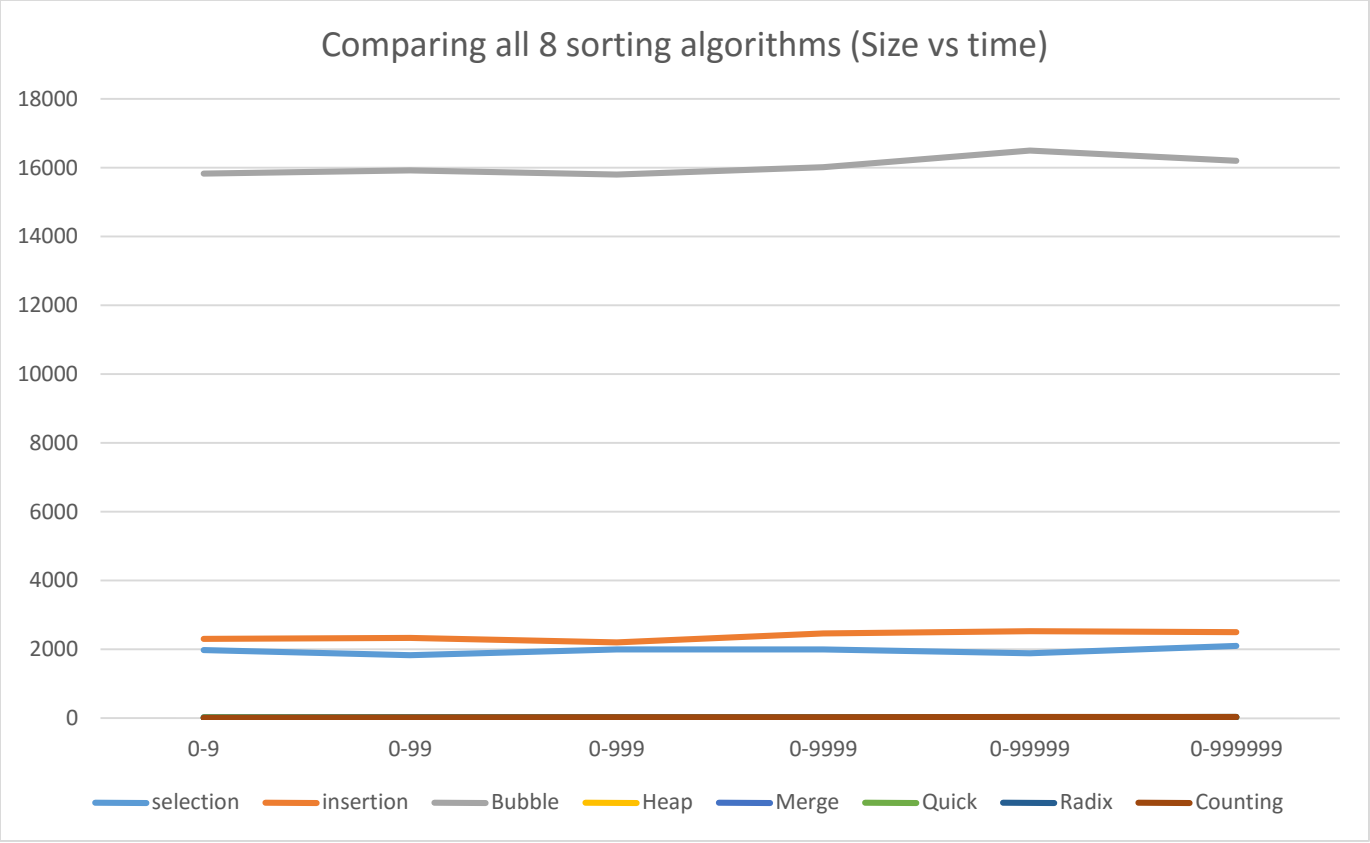

Heap Sort (Size vs time)
18
16
14
12
10
8
6
4
2
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Merge sort (Size vs time)
35
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Quick Sort (Size vs time)
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Radix Sort(Size vs time)
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Counting Sort (Size vs time)
45
40
35
30
25
20
15
10
5
0
1000
10000
100000
1000000
0-9
0-99
0-999
0-9999
0-99999
0-999999


Comparing all 8 sorting algorithms (Size vs time)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0-9
0-99
0-999
0-9999
0-99999
0-999999
selection
insertion
Bubble
Heap
Merge
Quick
Radix
Counting