Afer OLKAY 2305126

EE441 Programming Assignment 3

Time Spent

Part 1 – HASHING: 15 Hours

Part 2 – SORTING: 12 Hours

Report Writing: 5.5 Hours

Total: 32.5 Hours

Part 1-HASHING Performance Evaluation

1. $lf = \frac{i}{n * r}$ where, for our case i=200, n=100 and r=1. This applies to all three hash functions. Therefore,

$$lf_{folding} = lf_{middle\ squaring} = lf_{truncation} = \frac{200}{100*1} = 2$$

2.

Hash Function	Number of Collisions
Folding	115
Middle Squaring	178
Truncation	113

NOTE: In console application, when asked to enter the hashing function, write the corresponding:

Folding: "folding "

Middle Squaring: "middlesquaring"

Truncation: "truncation"

3. Even though their loading factors are same, their number of collisions are different. The lowest is truncation and right after that folding is the second lowest with only 2 more collisions whereas middle squaring has the highest number of collisions by far. As the number of collisions increase, inserting, searching, and deleting functions have to travel to the end of the linked lists, which this journey takes more time and processing power which is undesired. Therefore, we can say that, folding and truncation functions are very close in terms of efficiency and both are more efficient than middle squaring function.

Part 2-SORTING Performance Evaluation

Note: Swap function is calculated as 3 data moves since it requires 3 individual assignment operation.

Table 3. Average Statistics for N=100

Algorithm	#comparisons	#moves	time(msec)
Bubble	4728	7525	0.01632
Selection	4950	297	0.01228
Quick-1	928	496	0.00206
Quick-2	887	547	0.0018
Quick-3	907	564	0.007
Quick-4	810	570	0.0097
My Own Sort	770	679	0.00146

Table 1. Average Statistics for N=1,000

Algorithm	#comparisons	#moves	time(msec)
Bubble	495351	743703	1.628
Selection	499500	2997	1.092
Quick-1	15181	7110	0.053
Quick-2	13065	7699	0.05
Quick-3	13852	7879	0.091
Quick-4	11873	8044	0.119
My Own Sort	12466	9252	0.031

Table 2. Average Statistics for N=5,000

Algorithm	#comparisons	#moves	time(msec)
Bubble	12482245	18737358	41.589
Selection	12497500	14997	25.999
Quick-1	99496	43831	0.423
Quick-2	83547	47437	0.415
Quick-3	82503	47678	0.502
Quick-4	74436	48720	0.646
My Own Sort	80299	54955	0.335

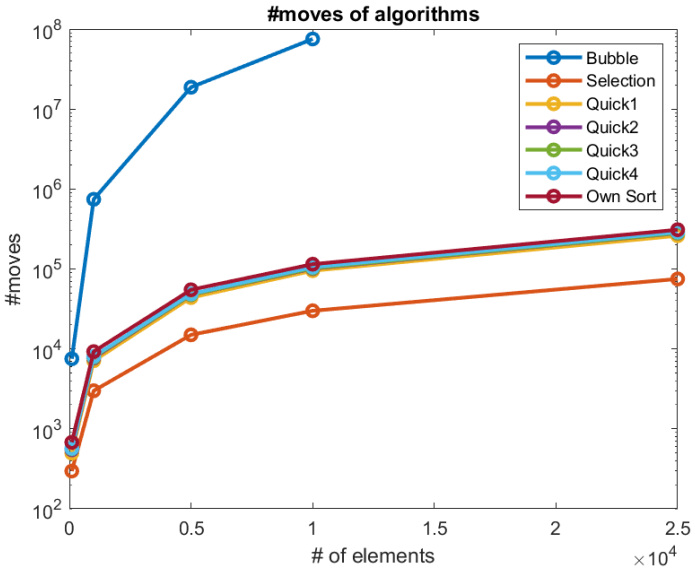
Table 3. Average Statistics for N=10,000

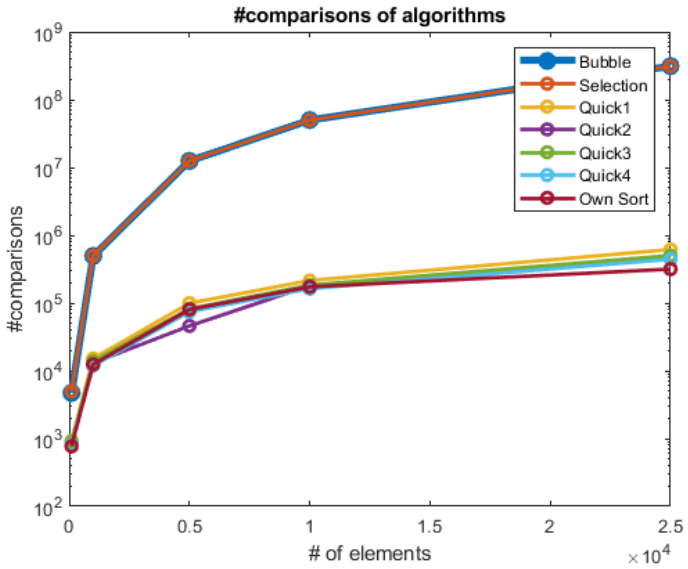
Algorithm	#comparisons	#moves	time(msec)
Bubble	49952985	75245065	182.3
Selection	49995000	29997	103.366
Quick-1	215002	95146	0.9
Quick-2	179595	101031	0.895
Quick-3	184200	102303	1.063
Quick-4	162624	105208	1.351
My Own Sort	173236	114778	0.738

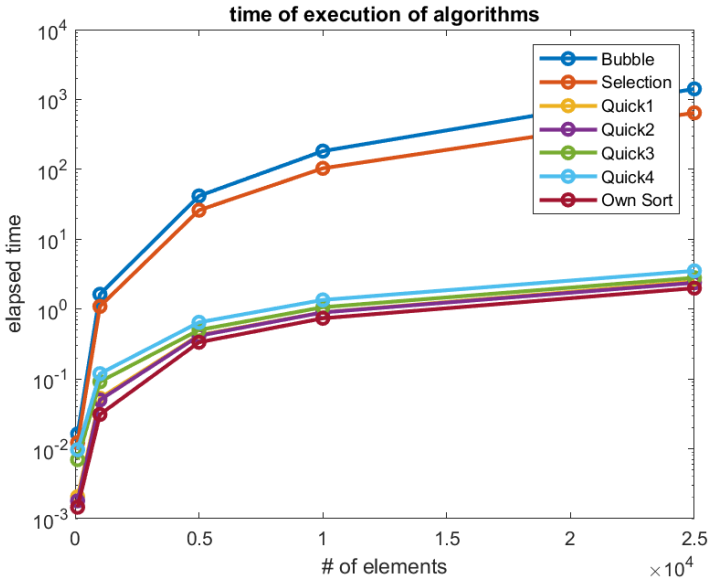
Table 4. Average Statistics for N=25,000

Algorithm	#comparisons	#moves	time(msec)
Bubble	312307796	Out of limit*	1417.25
Selection	312487500	74997	644.288
Quick-1	615656	260388	2.468
Quick-2	482534	280109	2.376
Quick-3	501807	281884	2.804
Quick-4	442667	289057	3.52
My Own Sort	316249	309473	1.992

*it was so large that allocated ram for the variable could not store it







When these graphs are overviewed, it is easy to see that Quick Sort variant algorithms perform similarly but not identical in all areas, namely, number of data comparisons, moves and execution time. Also, the difference between algorithms increase as the number of elements increase.

First, let us compare Quick Sort variants with Selection and Bubble Sort. In terms of execution time, Quick Sort variants are much superior to the Bubble and Selection Sort. Bubble Sort takes the longest to sort for all cases and Selection Sort always lasts approximately half of it. Interestingly, Bubble and Selection Sorts have the same number of comparisons for all cases, but Selection Sort has the lowest number of moves among the others whereas Bubble Sort has the highest. Therefore, it is obvious that Bubble Sort is the worst algorithm since it lacks behind in all categories and the reason why Selection Sort performs better than Bubble Sort is its low number of moves. But despite having the lowest number of data moves, Selection Sort's execution time is not even close to the Quick Sort variants' execution time. In terms of number of moves, Selection Sort is significantly better than the Quick Sort variants but when attention is given to number of data comparisons, Quick Sort variants dominate Selection Sort. By analysing this, we can say that data comparisons take more time and CPU labour therefore more of an important parameter than data moves.

And now let us compare the Quick Sort variants. It is known that for small data arrays, Quick Sort is not superior to other algorithms. My Own Sort performs best since it uses Quick Sort to sort big chunks of data and later uses insertion sort to finish the sorting. After that, middle element-pivot variant Quick Sorting is the second fastest. This is expected since its number of data comparisons is very close to My Own Sort, but it has a larger number of data moves compared to others. This is harmonious to the conclusion we arrived above that data comparisons are more dominant than data moves in terms of performance. First element-pivot variant comes third performing very similar to middle element-pivot, having the largest number of data comparisons but lowest number of data moves. Median of random 3-pivot variant perform worst since it wastes a lot of process power for choosing random 3 elements and then finding the median.