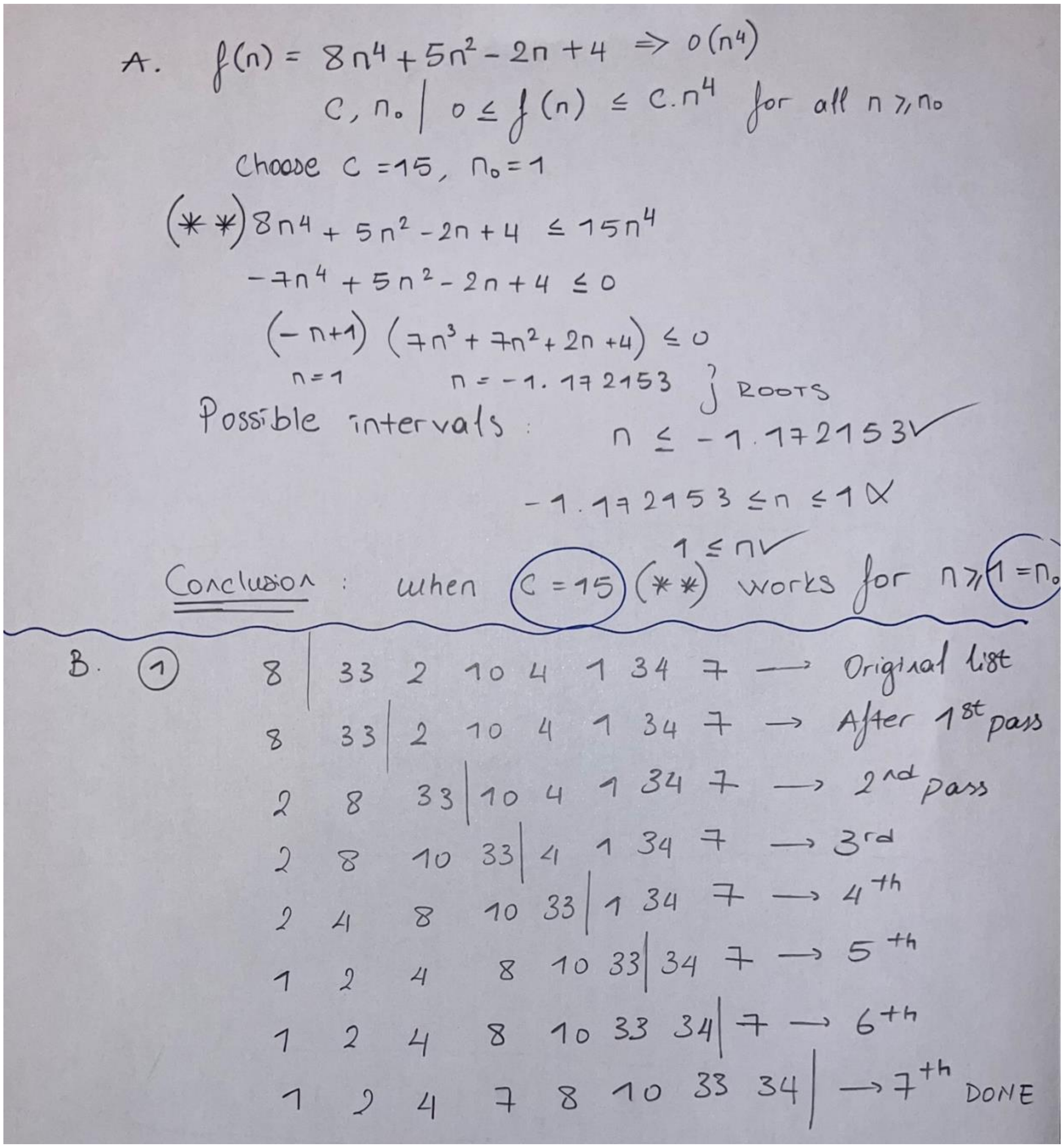
/**
* Title: Algorithm analysis & Sorting
* Author : Yasemin Akın
* ID: 22101782
* Section : 001
* Homework : 1
* Description : Answers for Question 1 and Question 3*/

Question 1



A. $f(n) = 8n^4 + 5n^2 - 2n + 4 \Rightarrow O(n^4)$

$c, n_0 \mid 0 \le f(n) \le c.n^4$ for all $n \geq n_0$

Choose $c = 15$, $n_0 = 1$

$(**)\ 8n^4 + 5n^2 - 2n + 4 \le 15n^4$

$-7n^4 + 5n^2 - 2n + 4 \le 0$

$(-n+1)(7n^3 + 7n^2 + 2n + 4) \le 0$

$n = 1$ $\quad n = -1.172153$ } ROOTS

Possible intervals : $n \le -1.172153$ ✓

$-1.172153 \le n \le 1$ ✗

$1 \le n$ ✓

Conclusion : when $\boxed{C = 15}$ $(**)$ works for $n \geq 1 = n_0$

B. (1)

8 | 33 2 10 4 1 34 7 $\longrightarrow$ Original list

8 33 | 2 10 4 1 34 7 $\longrightarrow$ After 1st pass

2 8 33 | 10 4 1 34 7 $\longrightarrow$ 2nd pass

2 8 10 33 | 4 1 34 7 $\longrightarrow$ 3rd

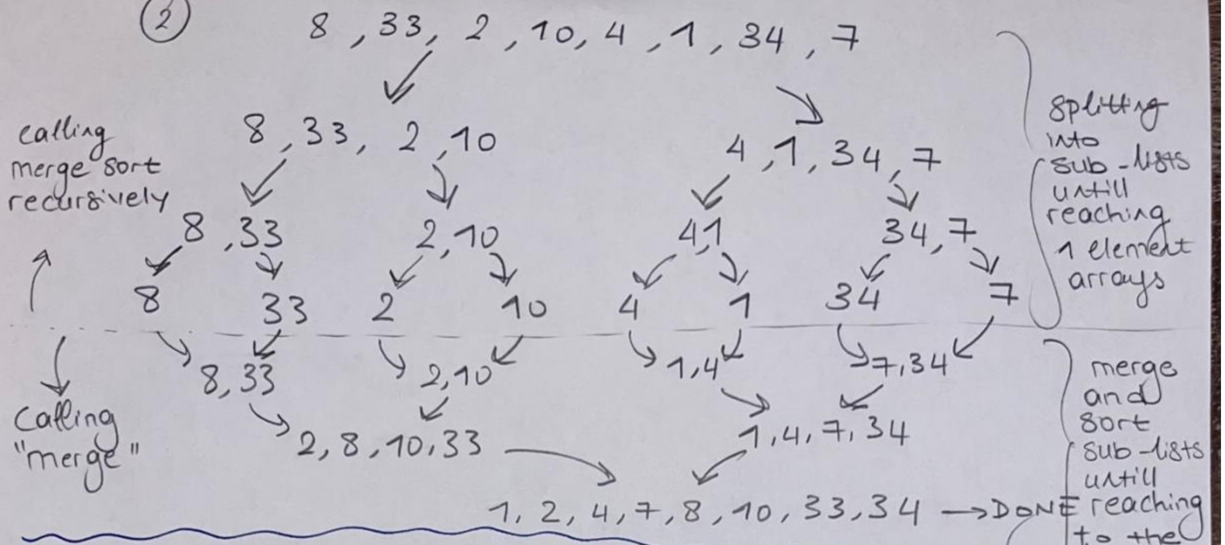
2 4 8 10 33 | 1 34 7 $\longrightarrow$ 4th

1 2 4 8 10 33 | 34 7 $\longrightarrow$ 5th

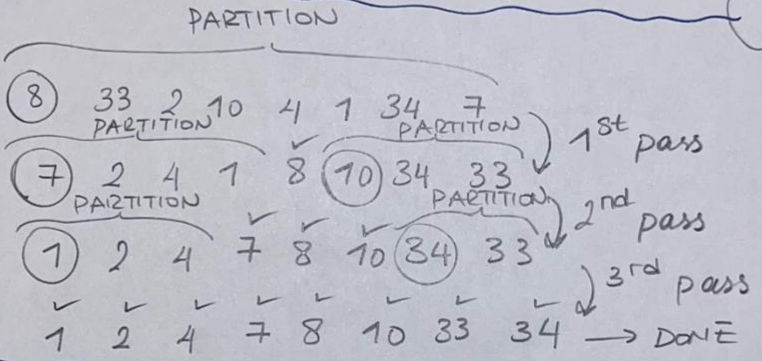
1 2 4 8 10 33 34 | 7 $\longrightarrow$ 6th

1 2 4 7 8 10 33 34 | $\longrightarrow$ 7th DONE

(2)


8 , 33, 2 , 10, 4 , 1 , 34 , 7
8 , 33, 2 , 10
4 , 1 , 34 , 7
8 , 33
2 , 10
4 1
34 , 7
8
33
2
10
4
1
34
7
8, 33
2, 10
1, 4
7, 34
2, 8, 10, 33
1, 4, 7, 34
1, 2, 4, 7, 8, 10, 33, 34 → DONE
calling merge sort recursively
Calling "merge"
splitting into sub-lists untill reaching 1 element arrays
merge and sort sub-lists untill reaching to the final sorted array

(3)



C. $T(n) = T\left(\frac{n}{2}\right) + n^2$, $T(1) = 1$

$$= T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2 + n^2$$

$$= T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2 + \left(\frac{n}{2}\right)^2 + n^2$$

$$= T\left(\frac{n}{2^k}\right) + \left(\frac{n}{2^{k-1}}\right)^2 + \left(\frac{n}{2^{k-2}}\right)^2 + \text{---} + n^2$$

$T(1) = 1$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

substitute $k = \log_2 n$

$$= T(1) + 4 + 16 + \ldots + n^2 = 1 + 4 + 16 + \ldots + n^2$$

geometric series $= 1 + 4^1 + 4^2 + 4^3 + \ldots + n^2 = \frac{1-4^n}{-3}$

$$S = \frac{a(1-r^n)}{1-r} = \frac{1-4^n}{1-4} = \frac{1-4^n}{-3}$$

S: sum of the series

a: first term $= 1$

r: common ratio $= 4$

n: number of terms $= n$

Asymptotic run times in big O notation $= O(4^n)$.

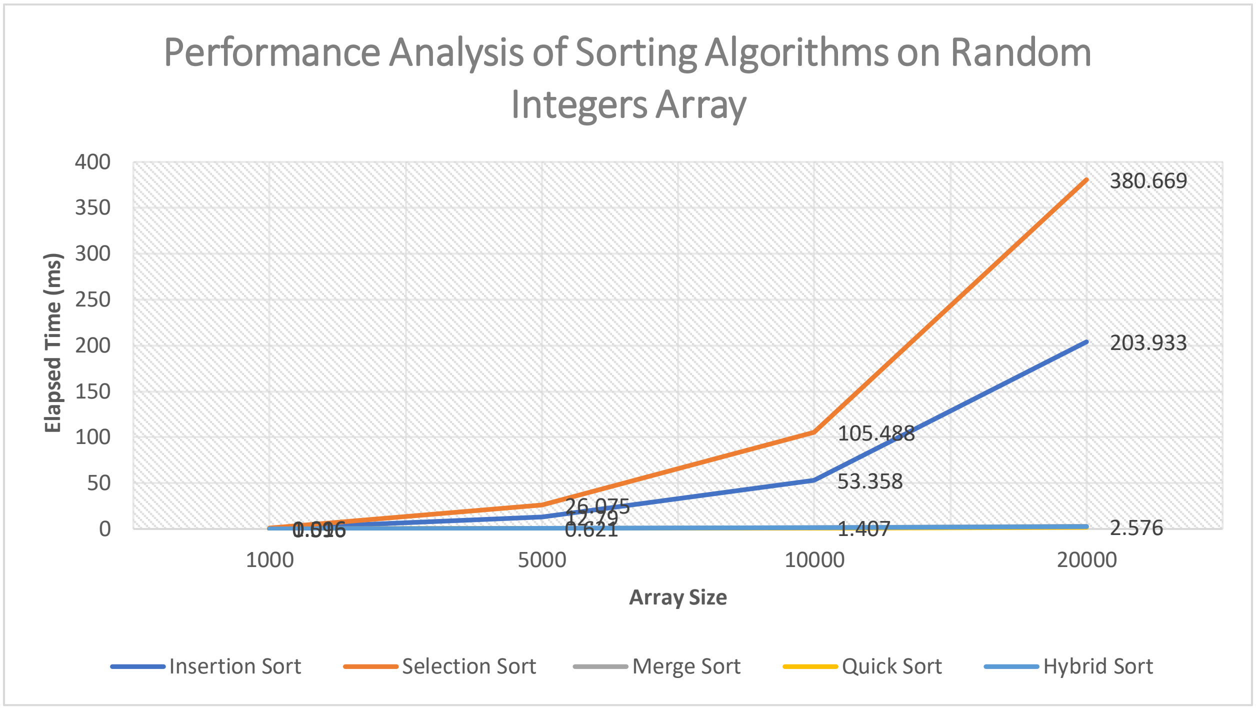
Question 3

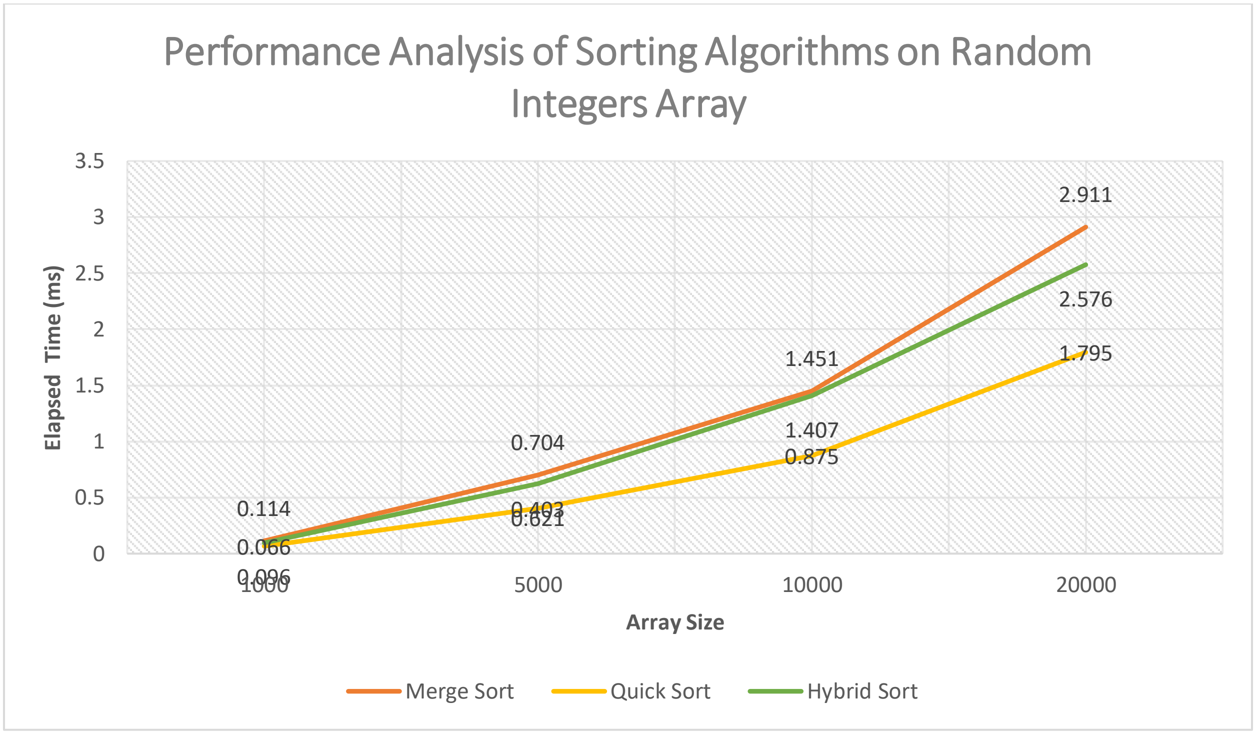
a) Data Table

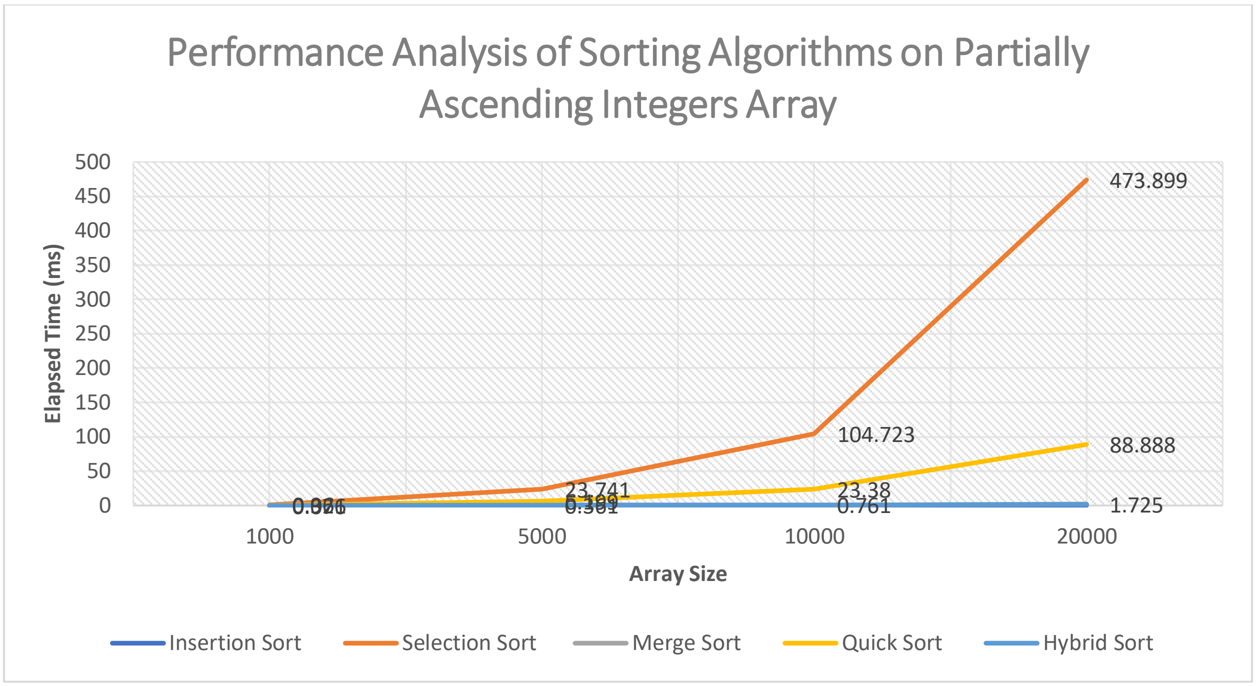
Array	Elapsed Time (ms)					Number of comparisons					Number of Data Moves				
	Insertion Sort	**Selection Sort**	**Merge Sort**	**Quick Sort**	**Hybrid Sort**	**Insertion Sort**	**Selection Sort**	**Merge Sort**	**Quick Sort**	**Hybrid Sort**	**Insertion Sort**	**Selection Sort**	**Merge Sort**	**Quick Sort**	**Hybrid Sort**
R1K	0.516	1.016	0.114	0.066	0.096	247966	499500	8738	10118	12805	248965	2997	19952	16552	22575
R5K	12.79	26.075	0.704	0.403	0.621	62799617	12497500	55209	69906	83461	6284616	14997	123616	105132	148412
R10K	53.358	105.488	1.451	0.875	1.407	24847258	49995000	120483	165540	177243	24857257	29997	267232	244943	318849
R20K	203.933	380.669	2.911	1.795	2.576	99983301	199990000	260790	349389	374713	100003300	59997	574464	484034	680011
A1K	0.018	0.96	0.116	0.321	0.076	3133	499500	5780	145131	8286	4132	2997	19952	6450	16965
A5K	0.092	23.741	0.477	6.199	0.361	19011	12497500	35698	3208560	54321	24010	14997	123616	34395	112166
A10K	0.179	104.723	0.953	23.38	0.761	40520	49995000	76811	12175600	115865	50519	29997	267232	70746	249054
A20K	0.338	473.899	2.163	88.888	1.725	86151	199990000	164165	46424052	246846	106150	59997	574464	143353	546622
D1K	1.011	0.995	0.128	0.39	0.111	498237	499500	5722	124970	10437	499236	2997	19952	215625	28908
D5K	24.343	23.932	0.479	7.315	0.445	12488032	12497500	35089	2561167	66979	12493031	14997	123616	4555146	185402
D10K	96.519	97.185	0.993	28.315	0.878	49974382	49995000	75512	10095507	138988	49984381	29997	267232	17585835	388566
D20K	384.89	485.328	2.107	104.924	1.868	199943848	199990000	161863	37142079	287873	199963847	59997	574464	64359229	808750

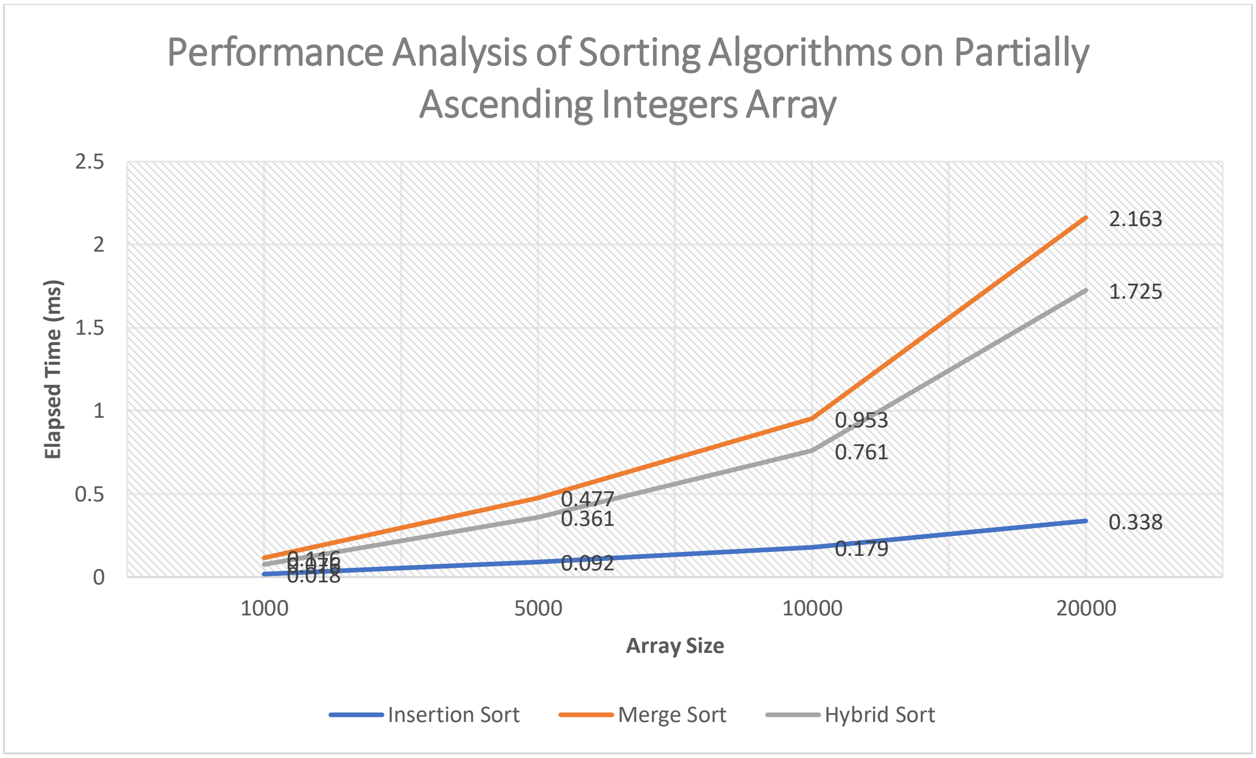
b) Graphs

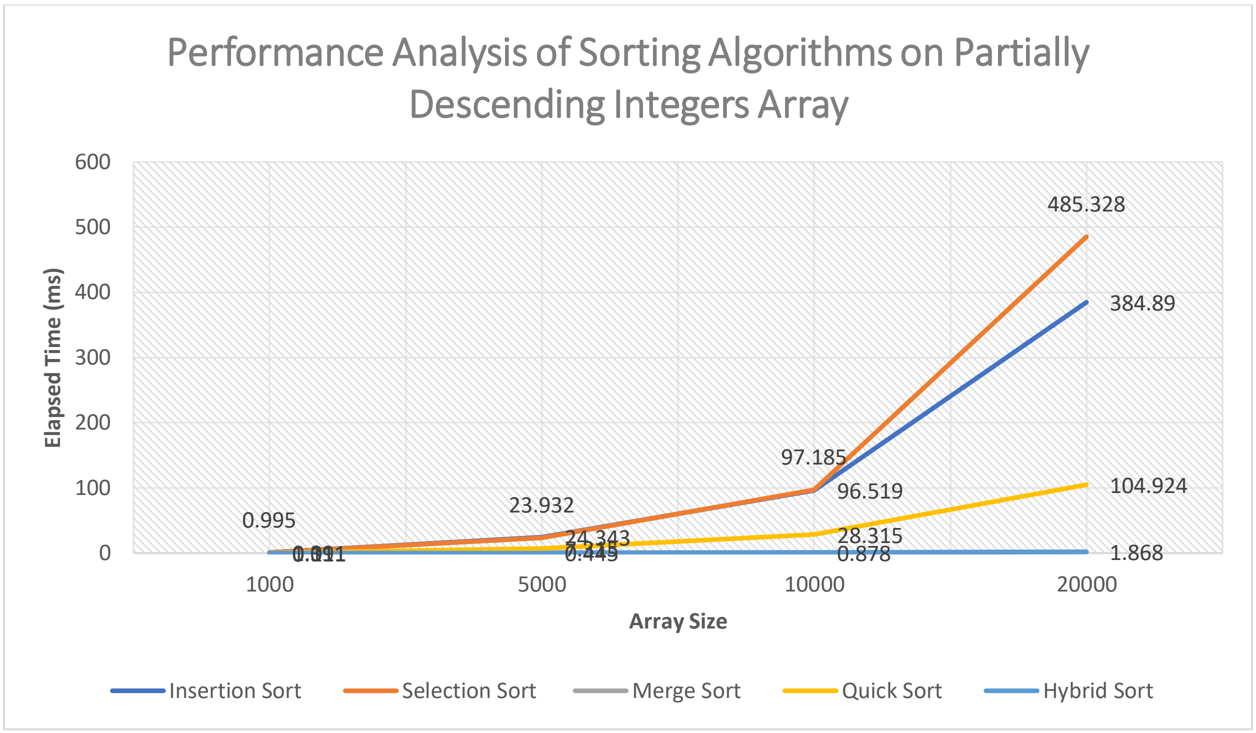
Here, I proceeded by drawing two graphs per array type. This is because showing all the algorithms in a single graph became meaningless due to the significant execution time differences between different algorithms. When those that take much time and those that take very little time are shown in the same graph, the lines of the fast algorithms disappear, as you can see below. For this reason, I showed the ones with shorter run times in separate charts and showed them again to clarify it.

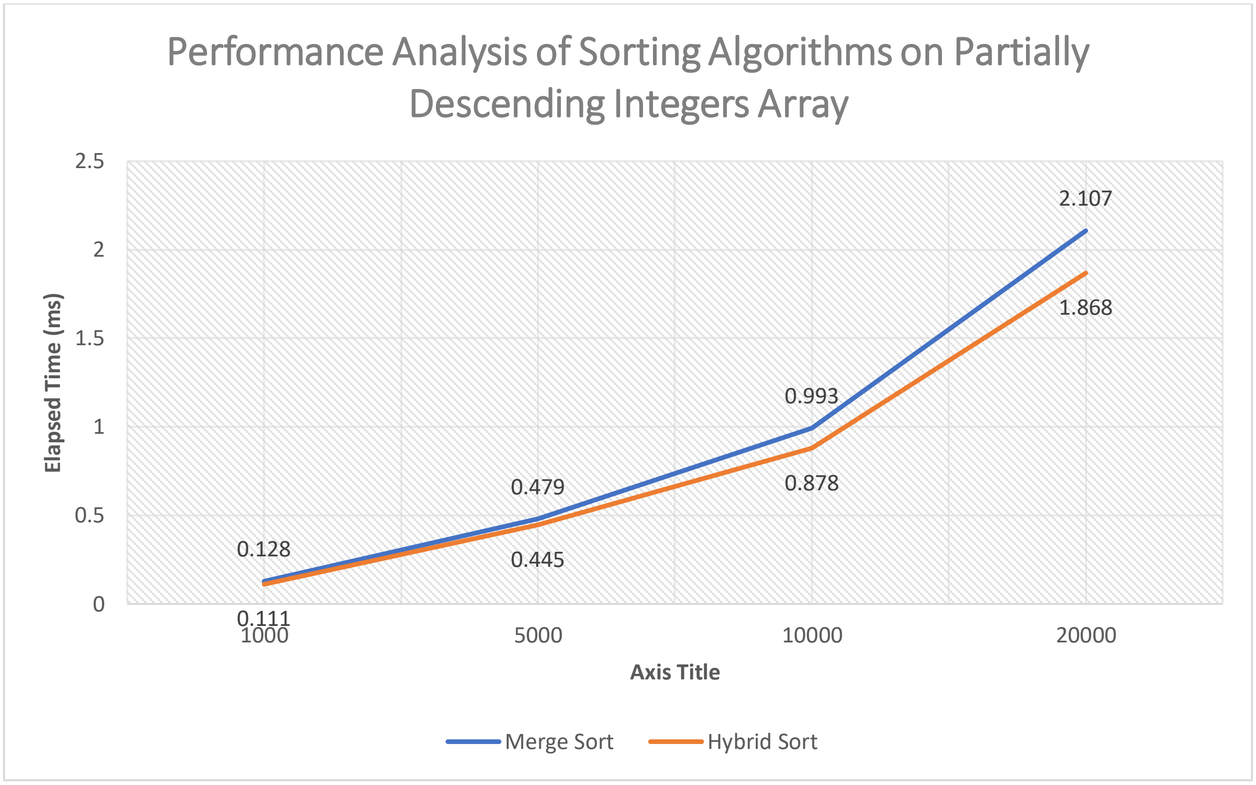

Performance Analysis of Sorting Algorithms on Random Integers Array
Elapsed Time (ms)
400
350
300
250
200
150
100
50
0
1000
5000
10000
20000
Array Size
380.669
203.933
105.488
53.358
26.075
12.79
0.621
1.407
2.576
0.096
Insertion Sort
Selection Sort
Merge Sort
Quick Sort
Hybrid Sort


Performance Analysis of Sorting Algorithms on Random Integers Array
Elapsed Time (ms)
3.5
3
2.5
2
1.5
1
0.5
0
1000
5000
10000
20000
Array Size
2.911
2.576
1.795
1.451
1.407
0.875
0.704
0.403
0.621
0.114
0.066
0.096
Merge Sort
Quick Sort
Hybrid Sort


Performance Analysis of Sorting Algorithms on Partially Ascending Integers Array
Elapsed Time (ms)
500
450
400
350
300
250
200
150
100
50
0
1000
5000
10000
20000
Array Size
473.899
104.723
88.888
23.741
23.38
0.361
0.761
1.725
Insertion Sort
Selection Sort
Merge Sort
Quick Sort
Hybrid Sort


Performance Analysis of Sorting Algorithms on Partially Ascending Integers Array
Elapsed Time (ms)
2.5
2
1.5
1
0.5
0
1000
5000
10000
20000
Array Size
2.163
1.725
0.953
0.761
0.477
0.361
0.338
0.179
0.092
0.116
0.018
Insertion Sort
Merge Sort
Hybrid Sort


Performance Analysis of Sorting Algorithms on Partially Descending Integers Array
600
500
400
300
200
100
0
Elapsed Time (ms)
485.328
384.89
97.185
96.519
104.924
0.995
23.932
24.343
28.315
0.878
1.868
0.445
1000
5000
10000
20000
Array Size
Insertion Sort
Selection Sort
Merge Sort
Quick Sort
Hybrid Sort


Performance Analysis of Sorting Algorithms on Partially Descending Integers Array
2.5
2
1.5
1
0.5
0
Elapsed Time (ms)
2.107
1.868
0.993
0.878
0.479
0.445
0.128
0.111
1000
5000
10000
20000
Axis Title
Merge Sort
Hybrid Sort

c) Comments

As the size of the arrays increases, elapsed time, move count, and comparison count increase in all algorithms, the insertion sort algorithm exhibits the behavior closest to the results expected by people who do not know the subject. The comparison and move count decreases as the array type gets closer to the sorted final version, from partially descending to partially ascending. Therefore, we observe a decrease in elapsed time. The most exciting results are those obtained from the data coming from the selection sort algorithm. Array type does not affect the selection sort algorithm's number of moves and comparisons.
For this reason, the elapsed time for arrays with exact sizes is almost the same in all array types. In the merge and quick sort algorithms, the fact that the variety was close to the correct order did not benefit the algorithm. The merge sort algorithm worked the slowest in the randomly generated array type. Also, surprisingly, the quick sort algorithm worked inefficiently in the partially ascending array type, making an utterly opposite corner. The hybrid sort algorithm gave results like insertion sort. It was at its most efficient when sorting partially climbing array types and was at its most inefficient when sorting partially descending array types. Another comment I can make in general is that the move and comparison count numbers are directly proportional to the time taken by the algorithm.

Quick sort was the most efficient algorithm when sorting randomly generated array types, insertion sort when sorting partially accelerating generated array types, and hybrid sort when sorting partially descending array types. If we compare the famous quick and merge sort duo with their speed. Although quick sort partially outperforms merge sort in randomly generated arrays, merge sort beats quick sort in partially ascending arrays. Quick sort loses all its efficiency in partly descending array types, and merge sort provides a great advantage. Selection sort always seems to be the most inefficient algorithm. Although insertion sort is faster than hybrid sort in partially ascending arrays, it loses to hybrid sort in randomly and partially descending collections. When we look at the results, no algorithm is the fastest and most efficient. The efficiency of algorithms varies depending on the situations in which they are used. However, if I were to make a final public comment, I would always show the selection sort as the least efficient algorithm among these algorithms.

We can also comment on the efficiency of sorting algorithms with different array types and sizes from the graphs. Nearly all 6 of the line graphs show us that while there is not much difference in the time the algorithms take to complete the sorting tasks when the array sizes are small, the time difference widens as the array sizes grow. We can understand this from the vertical divergence of the y coordinates of the straight lines on the right side of the graphs. Contrary to this interpretation, the lines of the hybrid and merge sort algorithms did not move too far away from each other according to the y-axis and showed similar progress. In most places, hybrid sort has served as a more efficient algorithm than merge sort.

In the first graphs that I gave for each array type, which include all sorting algorithms in one, algorithms with short run times did not appear clearly on the charts. Therefore, I made other second graphs for each array type where I show the sorting algorithms with short run times. The algorithms I show again in the second graphs are efficient algorithms that can and should be chosen for that array type. Merge and hybrid sort have always remained in the efficient section; quick sort has been added to this duo when sorting random arrays, and insertion sort is added when sorting partially ascending array types. Selection sort remained inefficient by always being in the upper parts of the first (upper) graphs.

d) There is no memory leak. It was tested with Valgrind.