CS-202 HOMEWORK 1

Full Name: Arda İynem
ID: 22002717
Section: 1
Assignment: 1

Question 1 - Part (a)

Question 1 - Part (a)

$T(n) = 3n^3 + 4n^2 + 2n$, $O(T(n)) = ?$

$0 \leq \underbrace{3n^3 + 4n^2 + 2n}_{T(n)} \leq c \underbrace{n^3}_{f(n)}$, for all $n \geq n_0$

$3 + \frac{4}{n} + \frac{2}{n^2} \leq c$, for all $n \geq n_0$

Let $c = 9$, $n_0 = 1$ $\Big\}$ $0 \leq 3n^3 + 4n^2 + 2n \leq 9n^3$ for all $n \geq 1$

Therefore $T(n) = O(n^3)$ as $T(n) \leq 9 \underbrace{n^3}_{f(n)}$ for $n \geq 1$

Question 1 - Part (b) $T(n) = T(n-1) + n^2$

Question 1 - Part (b)

$T(n) = T(n-1) + n^2, \quad T(1) = 1$

$= \left(T(n-2) + (n-1)^2\right) + n^2$

$= \left(T(n-3) + (n-2)^2\right) + (n-1)^2 + n^2$

$\vdots$

$= T(n-k) + \sum_{i=n-k+1}^{n} i^2$

$\Downarrow$

$T(n) = \underbrace{T(1)}_{1} + \underbrace{\sum_{i=2}^{n} i^2}_{\frac{n(n+1)(2n+1)}{6} - 1^2} \Rightarrow \boxed{T(n) = \frac{n(n+1)(2n+1)}{6}}$

$T(n) = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$

$c_1(n^3) \le \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \le c_2(n^3)$, for all $n \ge n_0$

$c_1 \le \frac{1}{3} + \frac{1}{2n} + \frac{1}{6n^2} \le c_3$, for all $n \ge n_0$

Let $c_1 = \frac{1}{3}$, $c_2 = 1$, $n_0 = 1$

$\frac{1}{3}\underbrace{n^3}_{f(n)} \le \underbrace{\frac{n(n+1)(2n+1)}{6}}_{T(n)} \le \underbrace{n^3}_{f(n)}$, for all $n \ge 1$

Thus $\Theta(T(n)) = \Theta(n^3)$

Question 1 - Part (b) **T(n) = 2 T(n/2) + n/2**

Question 1 – Part (b)

$T(n) = 2\,T(\frac{n}{2}) + \frac{n}{2}$, $T(1) = 1$

$= 2\,(2\,T(\frac{n}{4}) + \frac{n}{4}) + \frac{n}{2}$

$= 2\,(2\,(2\,T(\frac{n}{8}) + \frac{n}{8}) + \frac{n}{4}) + \frac{n}{2}$

$\vdots$

$= 2^k\,T(\frac{n}{2^k}) + k \cdot \frac{n}{2}$

$\Downarrow$

$T(n) = n \cdot \underbrace{T(1)}_{1} + \log_2 n \cdot \frac{n}{2} \Rightarrow \boxed{T(n) = n + \frac{n \log_2 n}{2}}$

$T(n) = n + \frac{\log_2 n}{2}$ $\quad c_1(n + n\log_2 n) \le n + \frac{n\log_2 n}{2} \le c_2(n + n\log_2 n)$ for all $n \ge n_0$

Let $c_1 = \frac{1}{2}$, $c_2 = 1$, $n_0 = 1$

$\frac{n}{2} + \frac{\log_2 n}{2} \le \overbrace{n + \frac{n\log_2 n}{2}}^{T(n)} \le n + \log_2 n$ for all $n \ge 1$

Therefore $\Theta(T(n)) = \Theta(n + \log_2 n)$

Question 1 - Part (c) **Selection Sort**

Question 1 - Part (c)

Selection Sort

Pass										
Inital	21	9	58	28	36	18	27	19	4	25
1	21	9	25	28	36	18	27	19	4	58
2	21	9	25	28	4	18	27	19	36	58
3	21	9	25	19	4	18	27	28	36	58
4	21	9	25	19	4	18	27	28	36	58
5	21	9	18	19	4	25	27	28	36	58
6	4	9	18	19	21	25	27	28	36	58
7	4	9	18	19	21	25	27	28	36	58
8	4	9	18	19	21	25	27	28	36	58
9	4	9	18	19	21	25	27	28	36	58

= Largest element in unsorted region marker

Unsorted | Sorted = Seperator

Question 1 – Part (c)

Insertion Sort

Pass										
Initial	21	9	58	28	36	18	27	19	4	25
1	21	9	58	28	36	18	27	19	4	25
2	9	21	58	28	36	18	27	19	4	25
3	9	21	58	28	36	18	27	19	4	25
4	9	21	28	58	36	18	27	19	4	25
5	9	21	28	36	58	18	27	19	4	25
6	9	18	21	28	36	58	27	19	4	25
7	9	18	21	27	28	36	58	19	4	25
8	9	18	19	21	27	28	36	58	4	25
9	4	9	18	19	21	27	28	36	58	25
10	4	9	18	19	21	25	27	28	36	58

□ = Key element to be compared and inserted into sorted region

Unsorted | Sorted = Seperator

Question 2 - Part (c)

```
[arda.iynem@dijkstra hw1]$ ./hw1
Analysis of Bubble Sort
12, 23, 24, 25, 26, 27, 29, 31, 32, 33, 35, 37, 38, 40, 56, 79
Data move count: 204
Key comparison count: 114

After merge sort:
12, 23, 24, 25, 26, 27, 29, 31, 32, 33, 35, 37, 38, 40, 56, 79
Data move count: 128
Key comparison count: 46

After quicksort:
12, 23, 24, 25, 26, 27, 29, 31, 32, 33, 35, 37, 38, 40, 56, 79
Data move count: 114
Key comparison count: 48
```

Question 2 - Part (d) Randomly Created Arrays

```
RANDOMLY CREATED ARRAYS:
 ========================
-----------------------------------------------------
Analysis of Bubble Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       78                7995225         12048096
8000       357               31992514        47974338
12000      857               71979635        109502193
16000      1585              127973855       191839524
20000      2515              199980130       301834467
24000      3666              287916747       431389797
28000      5034              391899680       589179015
32000      6613              511848019       769467843
36000      8406              647977344       967671630
40000      10447             799958264       1193508219
44000      12727             967951665       1450571304
48000      15146             1151972345      1720010772
-----------------------------------------------------
Analysis of Merge Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       1                 42875           95808
8000       2                 93584           207616
12000      4                 147579          327232
16000      5                 203384          447232
20000      6                 260846          574464
24000      8                 319408          702464
28000      9                 378659          830464
32000      11                438501          958464
36000      12                500096          1092928
40000      14                561955          1228928
44000      15                624036          1364928
48000      16                686870          1500928
-----------------------------------------------------
Analysis of Quick Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       1                 54070           87684
8000       2                 127365          189581
12000      3                 200702          339847
16000      4                 256978          395503
20000      5                 341802          567767
24000      6                 434622          662939
28000      7                 484386          751208
32000      8                 578225          1049390
36000      9                 632732          1022989
40000      10                696099          1052474
44000      12                820081          1228687
48000      13                936977          1617909
```

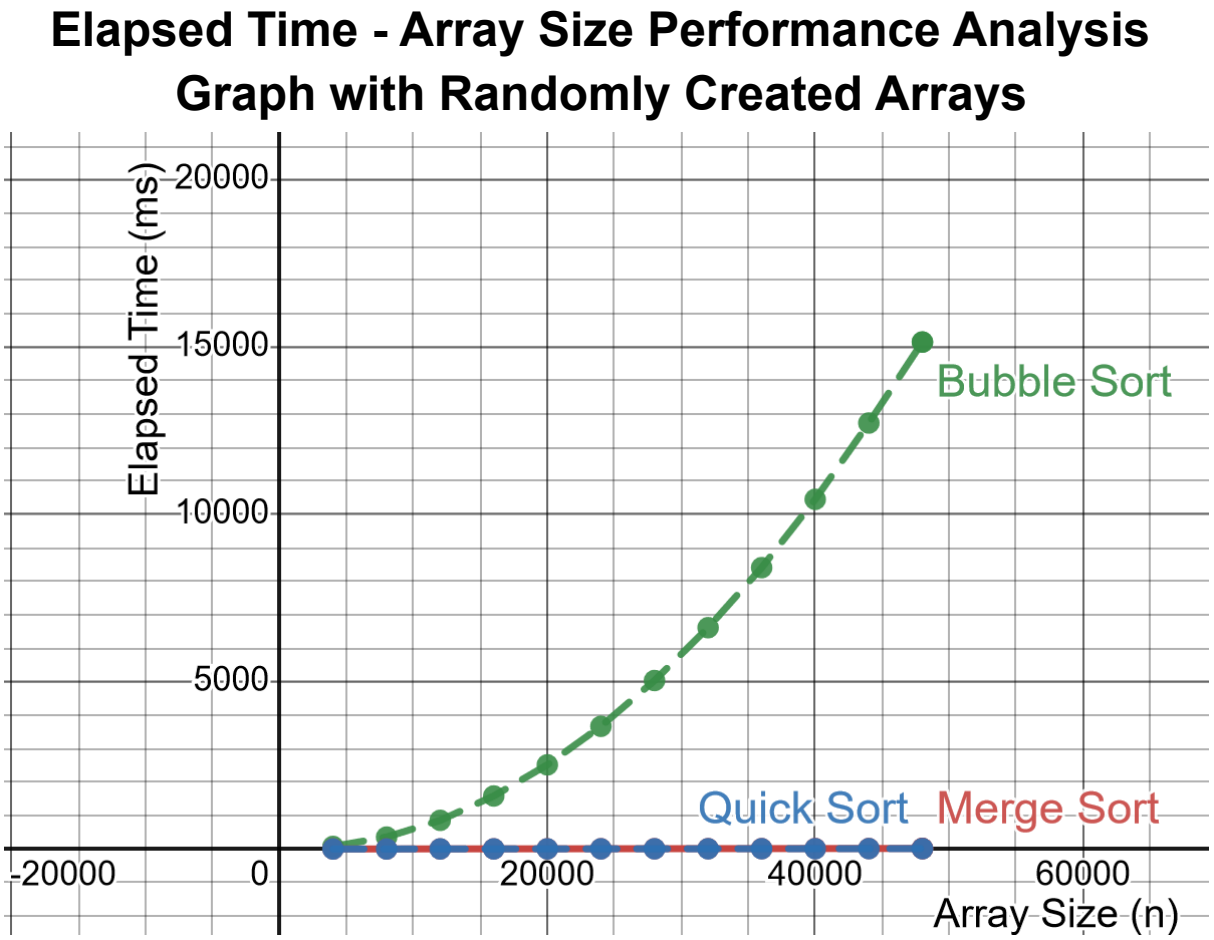
Question 2 - Part (d) Ascending Order Arrays

```
ASCENDING ARRAYS:
 ========================
-----------------------------------------------------
Analysis of Bubble Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       0                 3999            0
8000       0                 7999            0
12000      0                 11999           0
16000      0                 15999           0
20000      0                 19999           0
24000      0                 23999           0
28000      0                 27999           0
32000      0                 31999           0
36000      0                 35999           0
40000      0                 39999           0
44000      0                 43999           0
48000      0                 47999           0
-----------------------------------------------------
Analysis of Merge Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       1                 24352           95808
8000       1                 52735           207616
12000      2                 84702           327232
16000      3                 113472          447232
20000      4                 148864          574464
24000      4                 181404          702464
28000      5                 213961          830464
32000      6                 242998          958464
36000      7                 280953          1092928
40000      8                 317686          1228928
44000      9                 353687          1364928
48000      9                 386867          1500928
-----------------------------------------------------
Analysis of Quick Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       34                7998000         15996
8000       135               31996000        31996
12000      303               71994000        47996
16000      539               127992000       63996
20000      842               199990000       79996
24000      1212              287988000       95996
28000      1650              391986000       111996
32000      2155              511984000       127996
36000      2727              647982000       143996
40000      3367              799980000       159996
44000      4074              967978000       175996
48000      4849              1151976000      191996
```

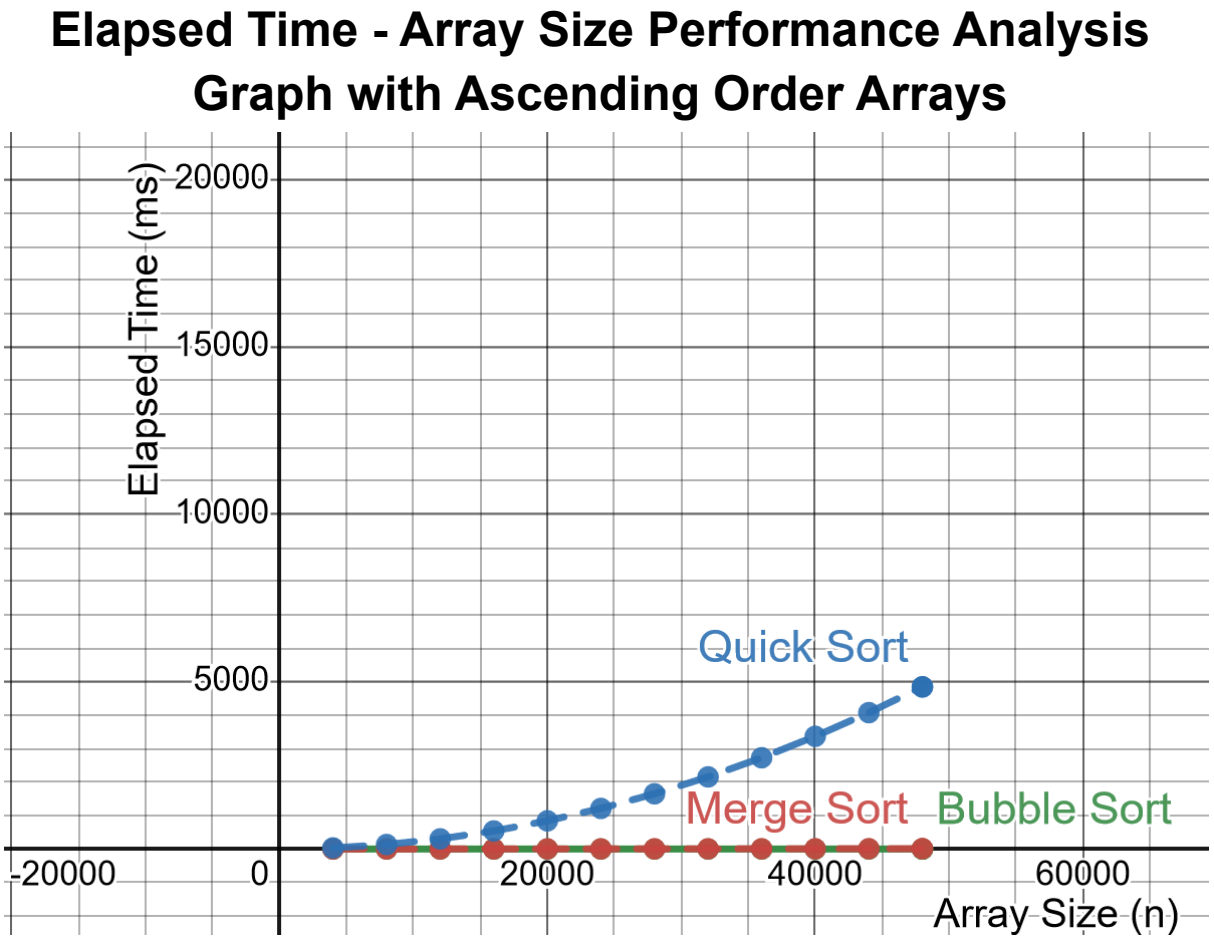
Question 2 - Part (d) Descending Order Arrays

```
DESCENDING ARRAYS:
 ========================
--------------------------------------------------
Analysis of Bubble Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       84                7998000         23992770
8000       335               31996000        95985399
12000      759               71994000        215978157
16000      1347              127992000       383970708
20000      2109              199990000       599963322
24000      3059              287988000       863955651
28000      4145              391986000       1175949027
32000      5409              511984000       1535941089
36000      6827              647982000       1943934087
40000      8427              799980000       2399926518
44000      10196             967978000       2903919333
48000      12126             1151976000      3455911602
--------------------------------------------------
Analysis of Merge Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       1                 23728           95808
8000       1                 51456           207616
12000      2                 79312           327232
16000      3                 110912          447232
20000      4                 139216          574464
24000      4                 170624          702464
28000      5                 202512          830464
32000      6                 237824          958464
36000      7                 267280          1092928
40000      8                 298432          1228928
44000      8                 330416          1364928
48000      9                 365248          1500928
--------------------------------------------------
Analysis of Quick Sort
Array Size Elapsed Time (ms) compCount       moveCount
4000       51                7568042         11369832
8000       202               30266187        45434636
12000      457               68400836        102654569
16000      810               121410256       182186390
20000      1264              189369439       284142911
24000      1818              272416504       408731107
28000      2484              371891122       557961183
32000      3233              483887141       725972334
36000      4105              614434773       921812050
40000      5060              757638899       1136635639
44000      6125              917468270       1376397664
48000      7279              1090510201      1635977891
```

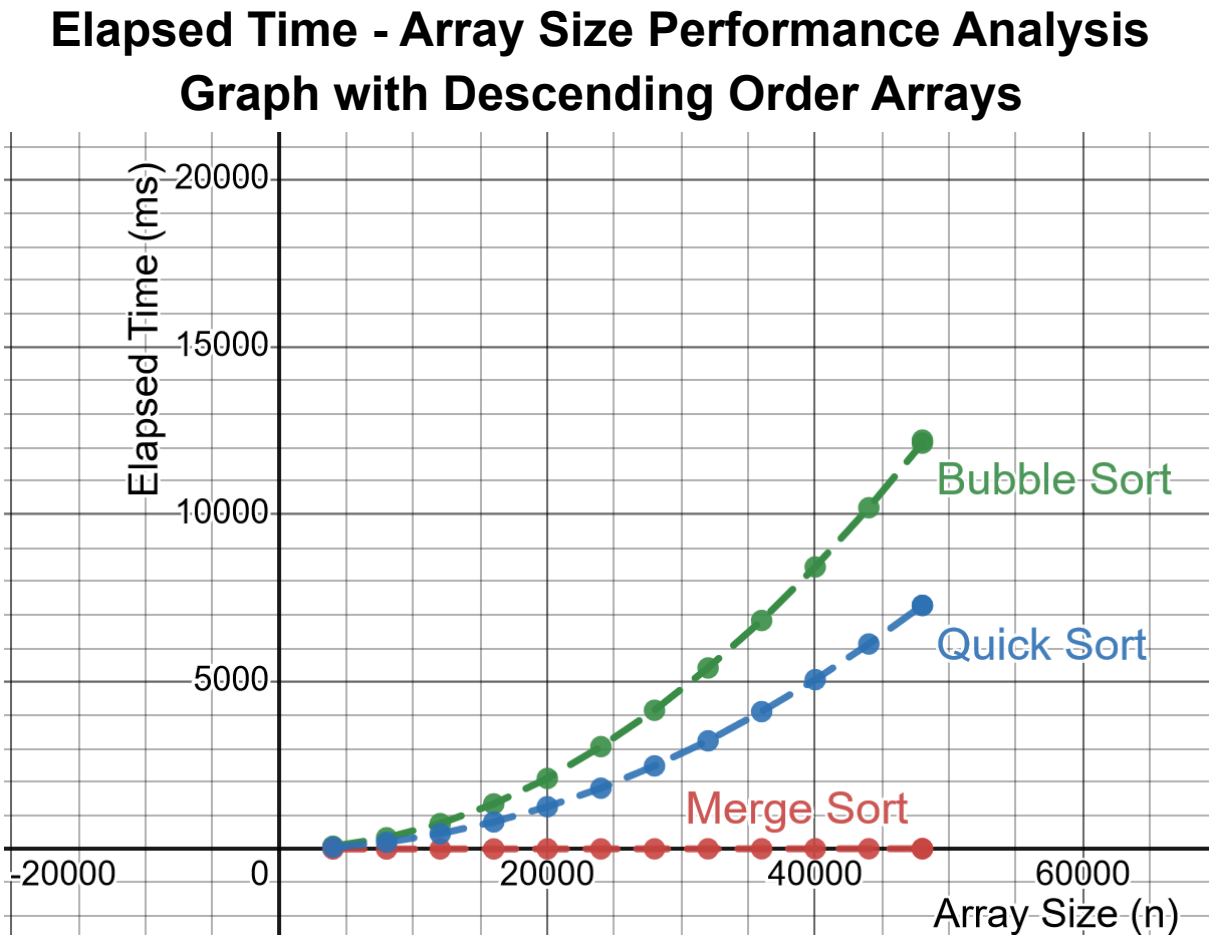
Question 3


Elapsed Time - Array Size Performance Analysis Graph with Randomly Created Arrays
Elapsed Time (ms)
20000
15000
10000
5000
-20000
0
20000
40000
60000
Array Size (n)
Bubble Sort
Quick Sort
Merge Sort

Question 3


Elapsed Time - Array Size Performance Analysis Graph with Ascending Order Arrays
Elapsed Time (ms)
20000
15000
10000
5000
Quick Sort
Merge Sort
Bubble Sort
-20000
0
20000
40000
60000
Array Size (n)

Question 3


Elapsed Time - Array Size Performance Analysis Graph with Descending Order Arrays
Elapsed Time (ms)
20000
15000
10000
5000
-20000
0
20000
40000
60000
Array Size (n)
Bubble Sort
Quick Sort
Merge Sort

Question 3 - Discussion

Randomly Created Arrays

Obtained empirical performance analysis results significantly resemble the theoretical results. Bubble sort was the slowest among all as expected while having a parabolic increase as its graph line is almost identical that of $O(n^2)$. Both Quick sort and Merge sort were remarkably fast and their plot line were almost identical to each other whereas they are similar to that of O(n*logn) (Data result confirms this while graph's proportions make it hard to see). Conclusively empirical results were very similar to theoretical ones.

Ascending Order Arrays

This time Bubble sort is the fastest due to the fact that n-1 key comparison and zero data movements have been done since the array is already sorted. Instead of theoretical O(n), a flat line has been plotted due to low cost of key comparisons with this size of data . Conversely, Quick sort is known to perform poorly when the array is sorted and its impact reflected on the graph since Quick sort was the slowest due to very high amount of key comparison even though its data movement was similar to that of Merge sort. And as expected its plot line resembles that of $O(n^2)$ which is also the theoretical result. Finally merge sort again performed well, it was almost as fast as bubble sort as its worst case theoretical result is O(n*logn) by contrast with quick sort.

Descending Order Arrays

As expected Bubble sort is the slowest because its worst case is a descending sorted array, and its graph corresponds to the theoretical result $O(n^2)$. Not much changed for Quick sort since its array is still sorted and worst case $O(n^2)$ continuous, however even though its key comparisons amount was similar to that of Bubble sort, its data movements were almost half which made Quick sort the second worst after Bubble sort. And as expected its plot line resembles that of $O(n^2)$ which is also the theoretical result. As expected Merge sort again performed best, since its both worst and best case is O(n * logn) nothing much changed through to different ordered arrays for merge sort.It kept its empirical results very similar to theoretical results in each case.

In conclusion

Each algorithm performed as expected for each different case, which has shown that their performance may vary depending on the array to be sorted. However, it is hard to ignore Merge sort's consistent and great performance in each case as it performs with same time complexity in each best and worst case O(n*logn). Quick sort also performs with O(n*logn) time complexity in best case (randomly sorted arrays) however it performs quite poorly compared to other 2 algorithms in its worst case (ascending/descending order arrays) with time complexity $O(n^2)$. Finally Bubble sort performs poorly except in its best case (randomly sorted arrays) with time complexity O(n) while it performs with time complexity $O(n^2)$ in worst and average cases. Merge sort seems to be the most efficient algorithm in general among all.