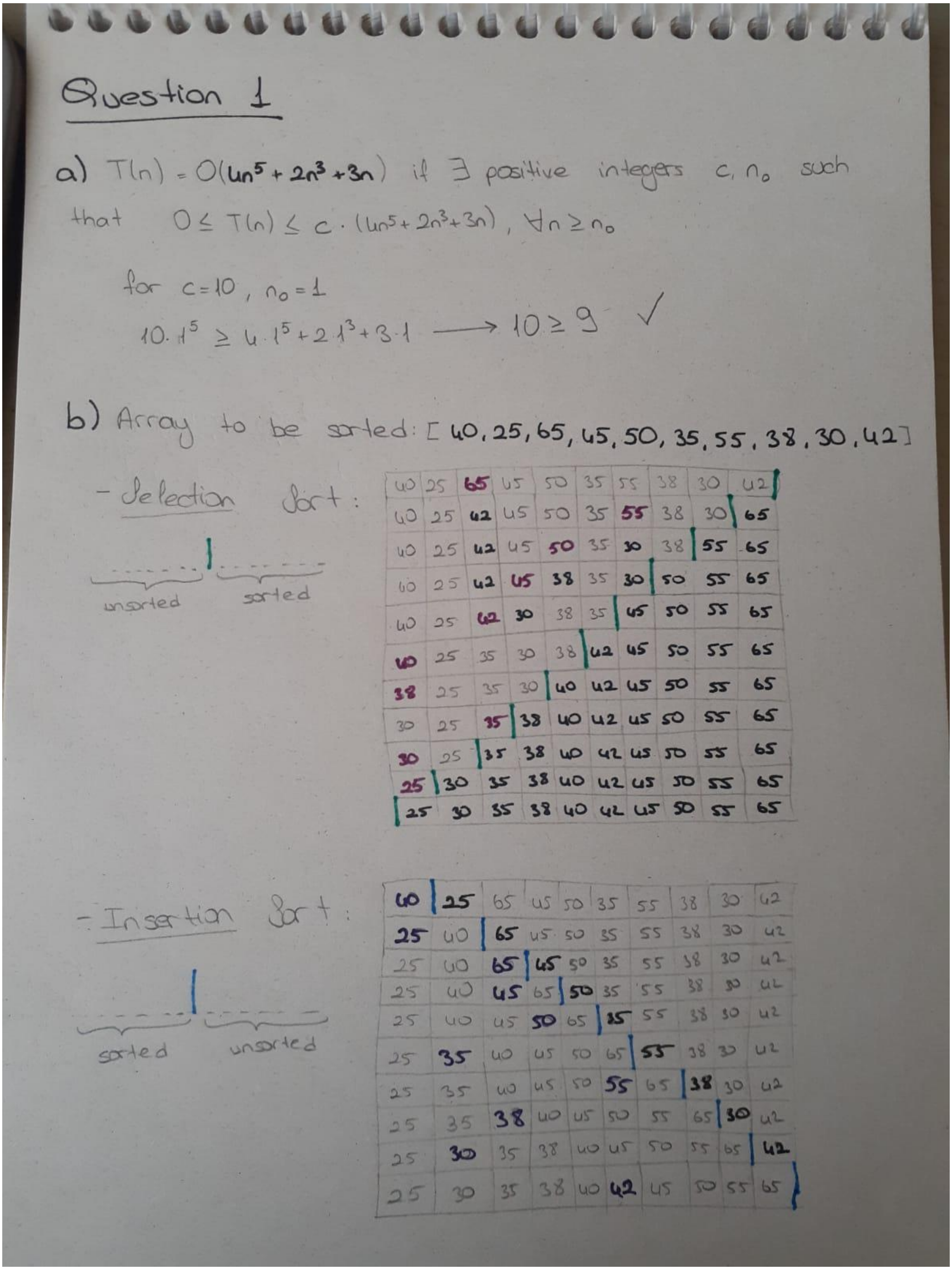
Sümeyye ACAR

ID: 22103640

CS202 / 001

HW1

Q1



Question 1

a) $T(n) = O(4n^5 + 2n^3 + 3n)$ if $\exists$ positive integers c, n_0 such that $0 \le T(n) \le c \cdot (4n^5 + 2n^3 + 3n)$, $\forall n \ge n_0$

for $c = 10$, $n_0 = 1$

$10 \cdot 1^5 \ge 4 \cdot 1^5 + 2 \cdot 1^3 + 3 \cdot 1 \longrightarrow 10 \ge 9$ ✓

b) Array to be sorted: [40, 25, 65, 45, 50, 35, 55, 38, 30, 42]

- Selection Sort:

$\underbrace{\text{- - - - - - -}}_{\text{unsorted}} \,|\, \underbrace{\text{- - - - - - -}}_{\text{sorted}}$

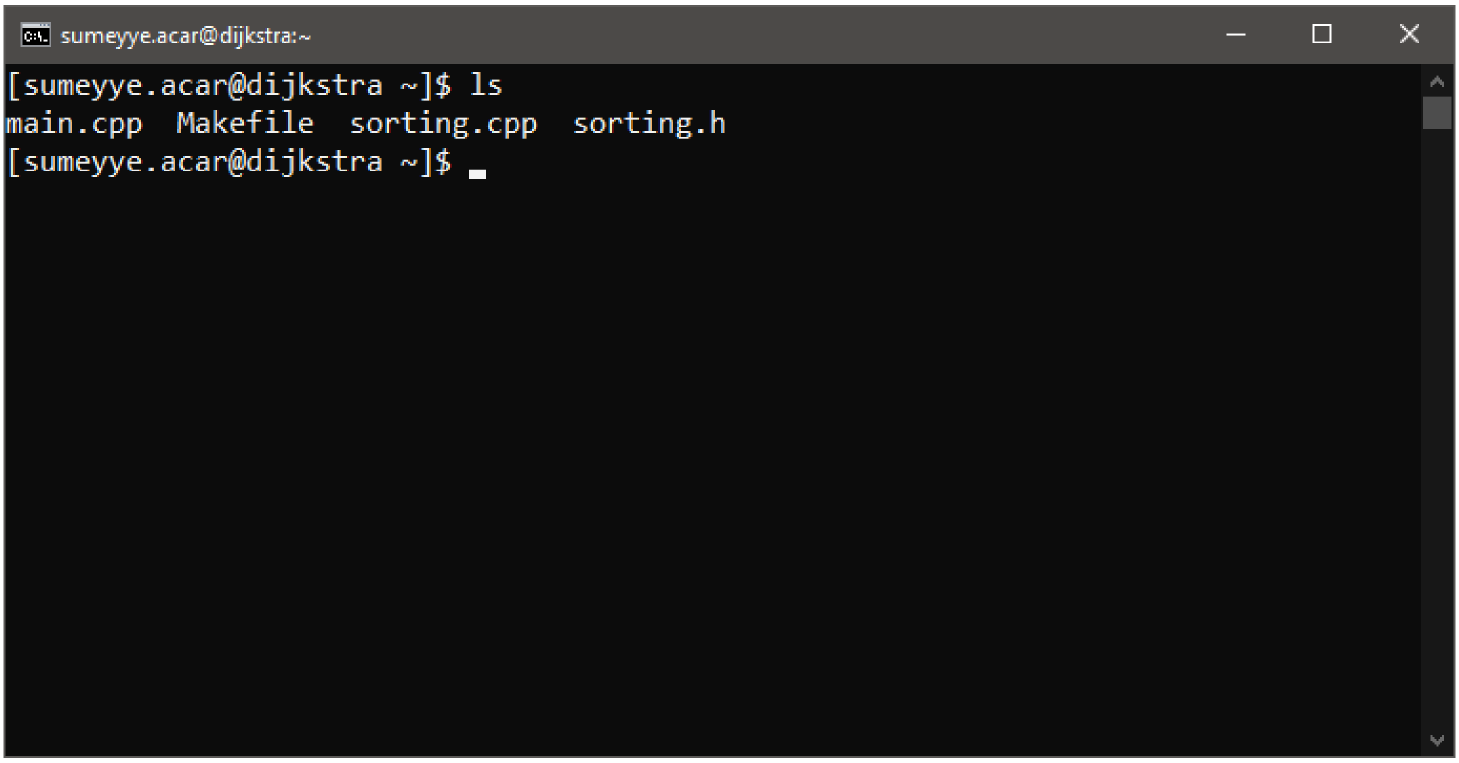
40	25	65	45	50	35	55	38	30	42
40	25	42	45	50	35	55	38	30	65
40	25	42	45	50	35	30	38	55	65
40	25	42	45	38	35	30	50	55	65
40	25	42	30	38	35	45	50	55	65
40	25	35	30	38	42	45	50	55	65
38	25	35	30	40	42	45	50	55	65
30	25	35	38	40	42	45	50	55	65
30	25	35	38	40	42	45	50	55	65
25	30	35	38	40	42	45	50	55	65
25	30	35	38	40	42	45	50	55	65

- Insertion Sort:

$\underbrace{\text{- - - - - - -}}_{\text{sorted}} \,|\, \underbrace{\text{- - - - - - -}}_{\text{unsorted}}$

40	25	65	45	50	35	55	38	30	42
25	40	65	45	50	35	55	38	30	42
25	40	65	45	50	35	55	38	30	42
25	40	45	65	50	35	55	38	30	42
25	40	45	50	65	35	55	38	30	42
25	35	40	45	50	65	55	38	30	42
25	35	40	45	50	55	65	38	30	42
25	35	38	40	45	50	55	65	30	42
25	30	35	38	40	45	50	55	65	42
25	30	35	38	40	42	45	50	55	65

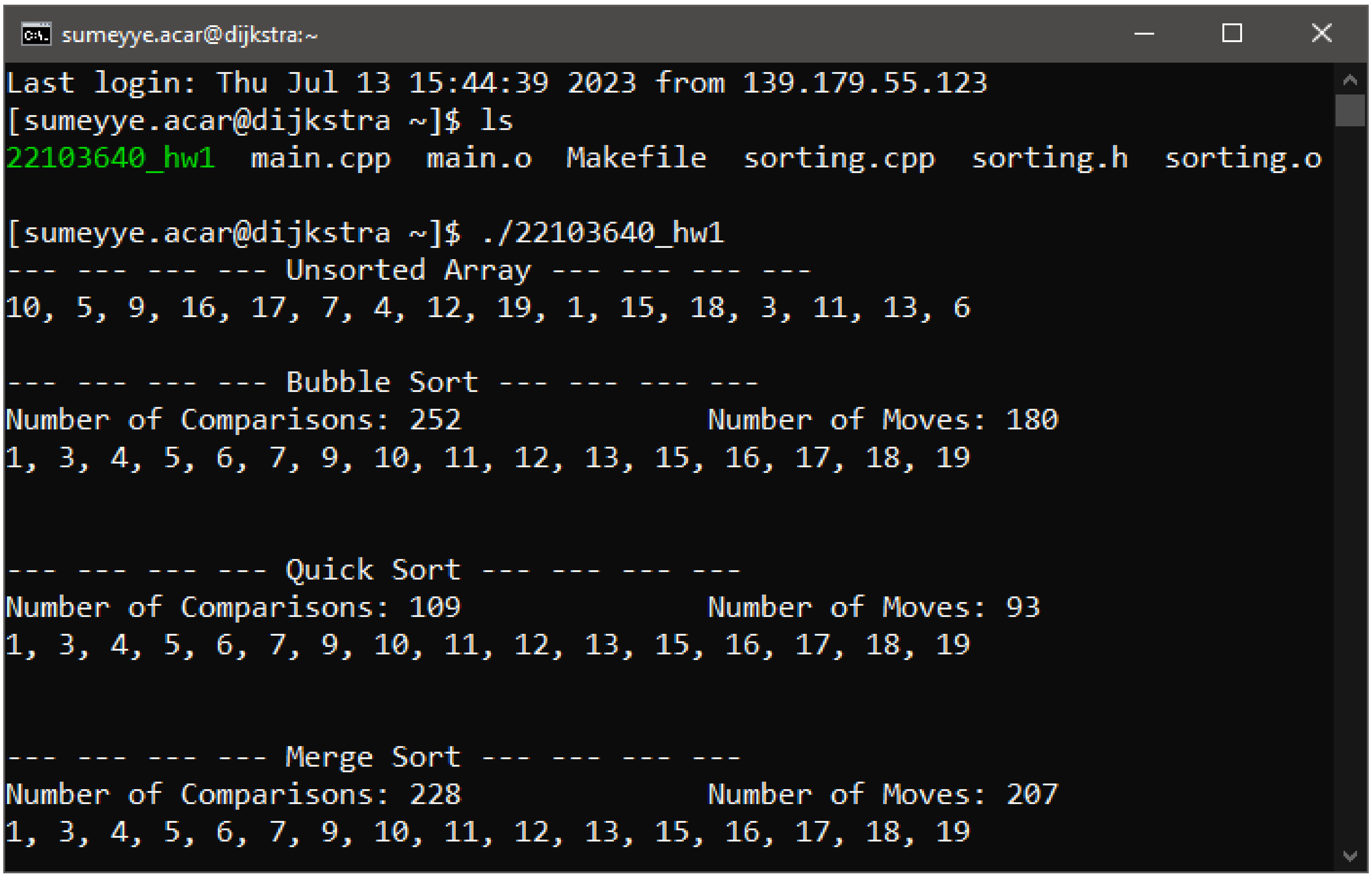
Q2



sumeyye.acar@dijkstra:~

```
[sumeyye.acar@dijkstra ~]$ ls
main.cpp  Makefile  sorting.cpp  sorting.h
[sumeyye.acar@dijkstra ~]$ _
```

After “make”:



sumeyye.acar@dijkstra:~

```
Last login: Thu Jul 13 15:44:39 2023 from 139.179.55.123
[sumeyye.acar@dijkstra ~]$ ls
22103640_hw1  main.cpp  main.o  Makefile  sorting.cpp  sorting.h  sorting.o

[sumeyye.acar@dijkstra ~]$ ./22103640_hw1
--- --- --- --- Unsorted Array --- --- --- ---
10, 5, 9, 16, 17, 7, 4, 12, 19, 1, 15, 18, 3, 11, 13, 6

--- --- --- --- Bubble Sort --- --- --- ---
Number of Comparisons: 252              Number of Moves: 180
1, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19


--- --- --- --- Quick Sort --- --- --- ---
Number of Comparisons: 109              Number of Moves: 93
1, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19


--- --- --- --- Merge Sort --- --- --- ---
Number of Comparisons: 228              Number of Moves: 207
1, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19
```

Output of Performance Analysis (Array with Random Integers):

```
Array Size: 4000

Elapsed Time compCount moveCount

Bubble Sort:

72              15989958                    11973396

Merge Sort:

0               127983          147711

Quick Sort:

19              16003999                    11997

-----------------------------------------------

Array Size: 8000

Elapsed Time compCount moveCount

Bubble Sort:

289             63987408                    48341895

Merge Sort:

1               275967          319423

Quick Sort:

82              64007999                    23997

-----------------------------------------------

Array Size: 12000

Elapsed Time compCount moveCount

Bubble Sort:

653             144010878                   108452085

Merge Sort:

2               435535          502847

Quick Sort:

200             144011999                   35997

-----------------------------------------------

Array Size: 16000

Elapsed Time compCount moveCount

Bubble Sort:

1249            256015790                   192499593

Merge Sort:

2               591935          686847

Quick Sort:

326             256015999                   47997

-----------------------------------------------

Array Size: 20000

Elapsed Time compCount moveCount

Bubble Sort:

2053            399999120                   298059414

Merge Sort:

4               762479          881695

Quick Sort:

660             400019999                   59997

-----------------------------------------------

Array Size: 24000

Elapsed Time compCount moveCount

Bubble Sort:

3102            576011568                   431855091

Merge Sort:

4               931071          1077695

Quick Sort:

868             576023999                   71997
```

Output of Performance Analysis (Array with Ascending Integers):

```
-----------------------------------------------
Array Size: 4000
Elapsed Time     compCount        moveCount
Bubble Sort:
0                8000             0
Merge Sort:
0                127983           147711
Quick Sort:
20               16003999                 11997
-----------------------------------------------
Array Size: 8000
Elapsed Time     compCount        moveCount
Bubble Sort:
0                16000            0
Merge Sort:
1                275967           319423
Quick Sort:
94               64007999                 23997
-----------------------------------------------
Array Size: 12000
Elapsed Time     compCount        moveCount
Bubble Sort:
0                24000            0
Merge Sort:
2                435535           502847
Quick Sort:
204              144011999                35997
-----------------------------------------------
Array Size: 16000
Elapsed Time     compCount        moveCount
Bubble Sort:
0                32000            0
Merge Sort:
2                591935           686847
Quick Sort:
356              256015999                47997
```

```
Array Size: 20000
Elapsed Time     compCount        moveCount
Bubble Sort:
0                40000            0
Merge Sort:
5                762479           881695
Quick Sort:
537              400019999                59997
-----------------------------------------------
Array Size: 24000
Elapsed Time     compCount        moveCount
Bubble Sort:
0                48000            0
Merge Sort:
4                931071           1077695
Quick Sort:
837              576023999                71997
```

Output of Performance Analysis (Array with Descending Integers):

```
-----------------------------------------------
Array Size: 4000
Elapsed Time        compCount         moveCount
Bubble Sort:
0                   63944             83913
Merge Sort:
0                   127983            147711
Quick Sort:
19                  16003999                  11997
-----------------------------------------------
Array Size: 8000
Elapsed Time        compCount         moveCount
Bubble Sort:
0                   127944            167916
Merge Sort:
1                   275967            319423
Quick Sort:
77                  64007999                  23997
-----------------------------------------------
Array Size: 12000
Elapsed Time        compCount         moveCount
Bubble Sort:
1                   311844            431763
Merge Sort:
2                   435535            502847
Quick Sort:
183                 144011999                 35997
-----------------------------------------------
Array Size: 16000
Elapsed Time        compCount         moveCount
Bubble Sort:
1                   447818            623727
Merge Sort:
2                   591935            686847
Quick Sort:
353                 256015999                 47997
```

```
Array Size: 20000
Elapsed Time        compCount         moveCount
Bubble Sort:
1                   319944            419913
Merge Sort:
4                   762479            881695
Quick Sort:
562                 400019999                 59997
-----------------------------------------------
Array Size: 24000
Elapsed Time        compCount         moveCount
Bubble Sort:
2                   575868            791799
Merge Sort:
4                   931071            1077695
Quick Sort:
724                 576023999                 71997
```

Q3

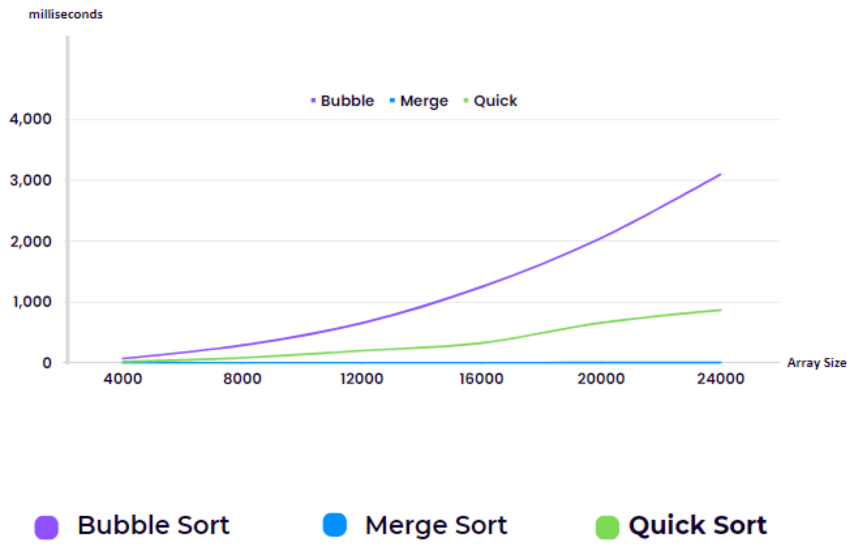


Figure 1.: Array with Random Generated Integers

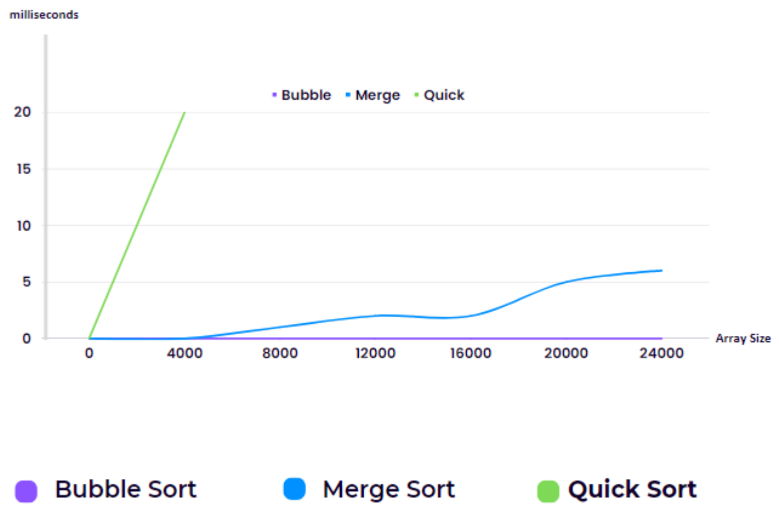


Figure 2.: Array with Ascending Integers

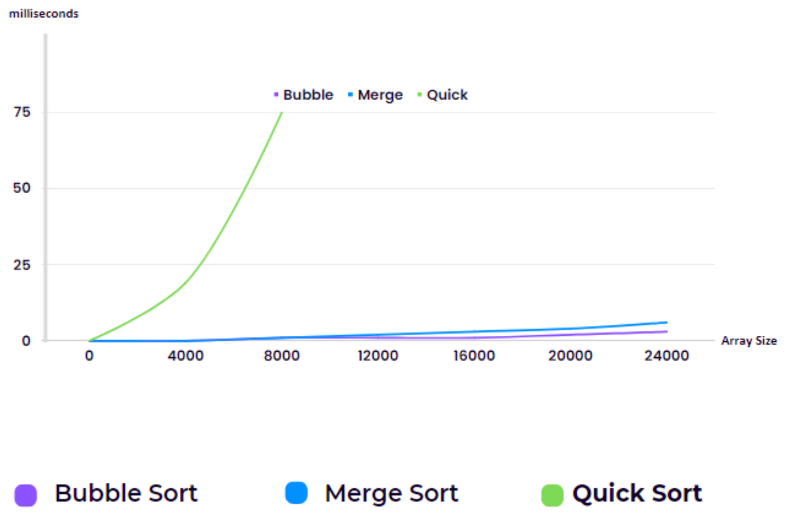


Figure 3.: Array with Descending Integers

Observations:

According to the results shown above (Figure 1, Figure 2, and Figure 3);

First graph represents the case in which the content of the array is unknown. All three algorithms' running times increases as the array size increases as expected. Bubble Sort has the highest increase rate and Merge Sort has the lowest. The graph concludes, the most time efficient sorting algorithm regarding a random generated array is Merge Sort algorithm.

Second graph proves that the Bubble Sort algorithm does the comparison first and performs a move only if the numbers are not already sorted. On the other hand, both Merge Sort and Quick Sort do go through with the algorithms even if the array is already sorted as expected. Here again, one can see that merge sort algorithm is way faster than quick sort algorithm.

The array used to obtain results as in the third graph was sorted to, however, in a descending order. The behavior of the quick sort algorithm does not change remarkably as expected. Bubble sort and merge sort close again but bubble sort is faster than merge sort due to its repetends'. If the data is sorted, bubble sort is faster than merge sort or quick sort regardless of ascending - descending.

The difference between the obtained results and the hypothetical results occurs due to model and capacity of the computer at the very moment the code is run.

All in all, if the content of an array is unknow, it is safer to use Merge Sort; if the array is sorted (ascending or descending), bubble sort or quick sort would make a good choice.