Logo, company name

Description automatically generated

CS 202

# Section 1

# Homework 1

# Ömer Oktay Gültekin

# 21901413

# 08/03/2022

Question 1:

1. Show that is by specifying appropriate c and values in Big-O definition

We need to find 2 positive constants c and such that:

Choose c = 20 and = 1:

Therefore

1. Trace the following sorting algorithms to sort the array [ 22, 8, 49, 25, 18, 30, 20, 15, 35, 27 ] in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps.
2. Selection Sort

Initial Array: [ 22, 8, 49, 25, 18, 30, 20, 15, 35, 27 ]

Pass number 1:

Number of key comparisons: 9

Key comparison pairs: (8, 22), (49, 22), (25, 49), (18, 49), (30, 49), (20, 49), (15, 49), (35, 49), (27, 49)

Largest index and value: 2, 49

Last index and value: 9, 27

Swap between indexes 2 and 9: temp = 49, x = 27, y = 49

Total swap and moves so far: 1, 3

After iteration: [ 22, 8, 27, 25, 18, 30, 20, 15, 35, 49 ]

Pass number 2:

Number of key comparisons: 8

Key comparison pairs: (8, 22), (27, 22), (25, 27), (18, 27), (30, 27), (20, 30), (15, 30), (35, 30)

Largest index and value: 8, 35

Last index and value: 8, 35

Swap between indexes 8 and 8: temp = 35, x = 35, y = 35

Total swap and moves so far: 2, 6

After iteration: [ 22, 8, 27, 25, 18, 30, 20, 15, 35, 49 ]

Pass number 3:

Number of key comparisons: 7

Key comparison pairs: (8, 22), (27, 22), (25, 27), (18, 27), (30, 27), (20, 30), (15, 30)

Largest index and value: 5, 30

Last index and value: 7, 15

Swap between indexes 5 and 7: temp = 30, x = 15, y = 30

Total swap and moves so far: 3, 9

After iteration: [ 22, 8, 27, 25, 18, 15, 20, 30, 35, 49 ]

Pass number 4:

Number of key comparisons: 6

Key comparison pairs: (8, 22), (27, 22), (25, 27), (18, 27), (15, 27), (20, 27)

Largest index and value: 2, 27

Last index and value: 6, 20

Swap between indexes 2 and 6: temp = 27, x = 20, y = 27

Total swap and moves so far: 4, 12

After iteration: [ 22, 8, 20, 25, 18, 15, 27, 30, 35, 49 ]

Pass number 5:

Number of key comparisons: 5

Key comparison pairs: (8, 22), (20, 22), (25, 22), (18, 25), (15, 25)

Largest index and value: 3, 25

Last index and value: 5, 15

Swap between indexes 3 and 5: temp = 25, x = 15, y = 25

Total swap and moves so far: 5, 15

After iteration: [ 22, 8, 20, 15, 18, 25, 27, 30, 35, 49 ]

Pass number 6:

Number of key comparisons: 4

Key comparison pairs: (8, 22), (20, 22), (15, 22), (18, 22)

Largest index and value: 0, 22

Last index and value: 4, 18

Swap between indexes 0 and 4: temp = 22, x = 18, y = 22

Total swap and moves so far: 6, 18

After iteration: [ 18, 8, 20, 15, 22, 25, 27, 30, 35, 49 ]

Pass number 7:

Number of key comparisons: 3

Key comparison pairs: (8, 18), (20, 18), (15, 20)

Largest index and value: 2, 20

Last index and value: 3, 15

Swap between indexes 2 and 3: temp = 20, x = 15, y = 20

Total swap and moves so far: 7, 21

After iteration: [ 18, 8, 15, 20, 22, 25, 27, 30, 35, 49 ]

Pass number 8:

Number of key comparisons: 2

Key comparison pairs: (8, 18), (15, 18)

Largest index and value: 0, 18

Last index and value: 2, 15

Swap between indexes 0 and 2: temp = 18, x = 15, y = 18

Total swap and moves so far: 8, 24

After iteration: [ 15, 8, 18, 20, 22, 25, 27, 30, 35, 49 ]

Pass number 9:

Number of key comparisons: 1

Key comparison pairs: (8, 15)

Largest index and value: 0, 15

Last index and value: 1, 8

Swap between indexes 0 and 1: temp = 15, x = 8, y = 15

Total swap and moves so far: 9, 27

After iteration: [ 8, 15, 18, 20, 22, 25, 27, 30, 35, 49 ]

2) Bubble Sort

Initial Array: [ 22, 8, 49, 25, 18, 30, 20, 15, 35, 27 ]

Pass number 1:

Number of key comparisons: 9

Swap between indexes 0 and 1: temp = 22, x = 8, y = 22

Total swap and moves so far: 1, 3

After swap: [ 8, 22, 49, 25, 18, 30, 20, 15, 35, 27 ]

Swap between indexes 2 and 3: temp = 49, x = 25, y = 49

Total swap and moves so far: 2, 6

After swap: [ 8, 22, 25, 49, 18, 30, 20, 15, 35, 27 ]

Swap between indexes 3 and 4: temp = 49, x = 18, y = 49

Total swap and moves so far: 3, 9

After swap: [ 8, 22, 25, 18, 49, 30, 20, 15, 35, 27 ]

Swap between indexes 4 and 5: temp = 49, x = 30, y = 49

Total swap and moves so far: 4, 12

After swap: [ 8, 22, 25, 18, 30, 49, 20, 15, 35, 27 ]

Swap between indexes 5 and 6: temp = 49, x = 20, y = 49

Total swap and moves so far: 5, 15

After swap: [ 8, 22, 25, 18, 30, 20, 49, 15, 35, 27 ]

Swap between indexes 6 and 7: temp = 49, x = 15, y = 49

Total swap and moves so far: 6, 18

After swap: [ 8, 22, 25, 18, 30, 20, 15, 49, 35, 27 ]

Swap between indexes 7 and 8: temp = 49, x = 35, y = 49

Total swap and moves so far: 7, 21

After swap: [ 8, 22, 25, 18, 30, 20, 15, 35, 49, 27 ]

Swap between indexes 8 and 9: temp = 49, x = 27, y = 49

Total swap and moves so far: 8, 24

After swap: [ 8, 22, 25, 18, 30, 20, 15, 35, 27, 49 ]

After pass: [ 8, 22, 25, 18, 30, 20, 15, 35, 27, 49 ]

Pass number 2:

Number of key comparisons: 8

Swap between indexes 2 and 3: temp = 25, x = 18, y = 25

Total swap and moves so far: 9, 27

After swap: [ 8, 22, 18, 25, 30, 20, 15, 35, 27, 49 ]

Swap between indexes 4 and 5: temp = 30, x = 20, y = 30

Total swap and moves so far: 10, 30

After swap: [ 8, 22, 18, 25, 20, 30, 15, 35, 27, 49 ]

Swap between indexes 5 and 6: temp = 30, x = 15, y = 30

Total swap and moves so far: 11, 33

After swap: [ 8, 22, 18, 25, 20, 15, 30, 35, 27, 49 ]

Swap between indexes 7 and 8: temp = 35, x = 27, y = 35

Total swap and moves so far: 12, 36

After swap: [ 8, 22, 18, 25, 20, 15, 30, 27, 35, 49 ]

After pass: [ 8, 22, 18, 25, 20, 15, 30, 27, 35, 49 ]

Pass number 3:

Number of key comparisons: 7

Swap between indexes 1 and 2: temp = 22, x = 18, y = 22

Total swap and moves so far: 13, 39

After swap: [ 8, 18, 22, 25, 20, 15, 30, 27, 35, 49 ]

Swap between indexes 3 and 4: temp = 25, x = 20, y = 25

Total swap and moves so far: 14, 42

After swap: [ 8, 18, 22, 20, 25, 15, 30, 27, 35, 49 ]

Swap between indexes 4 and 5: temp = 25, x = 15, y = 25

Total swap and moves so far: 15, 45

After swap: [ 8, 18, 22, 20, 15, 25, 30, 27, 35, 49 ]

Swap between indexes 6 and 7: temp = 30, x = 27, y = 30

Total swap and moves so far: 16, 48

After swap: [ 8, 18, 22, 20, 15, 25, 27, 30, 35, 49 ]

After pass: [ 8, 18, 22, 20, 15, 25, 27, 30, 35, 49 ]

Pass number 4:

Number of key comparisons: 6

Swap between indexes 2 and 3: temp = 22, x = 20, y = 22

Total swap and moves so far: 17, 51

After swap: [ 8, 18, 20, 22, 15, 25, 27, 30, 35, 49 ]

Swap between indexes 3 and 4: temp = 22, x = 15, y = 22

Total swap and moves so far: 18, 54

After swap: [ 8, 18, 20, 15, 22, 25, 27, 30, 35, 49 ]

After pass: [ 8, 18, 20, 15, 22, 25, 27, 30, 35, 49 ]

Pass number 5:

Number of key comparisons: 5

Swap between indexes 2 and 3: temp = 20, x = 15, y = 20

Total swap and moves so far: 19, 57

After swap: [ 8, 18, 15, 20, 22, 25, 27, 30, 35, 49 ]

After pass: [ 8, 18, 15, 20, 22, 25, 27, 30, 35, 49 ]

Pass number 6:

Number of key comparisons: 4

Swap between indexes 1 and 2: temp = 18, x = 15, y = 18

Total swap and moves so far: 20, 60

After swap: [ 8, 15, 18, 20, 22, 25, 27, 30, 35, 49 ]

After pass: [ 8, 15, 18, 20, 22, 25, 27, 30, 35, 49 ]

Pass number 7:

Number of key comparisons: 3

After pass: [ 8, 15, 18, 20, 22, 25, 27, 30, 35, 49 ]

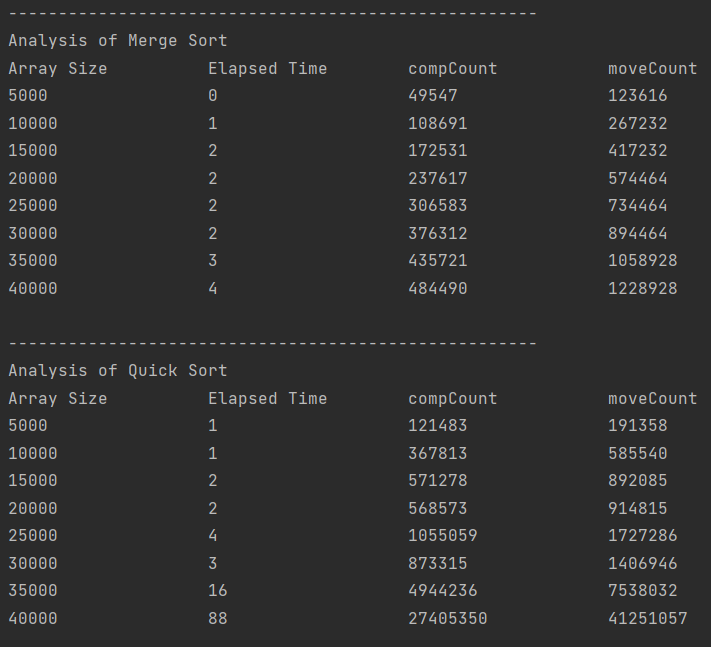
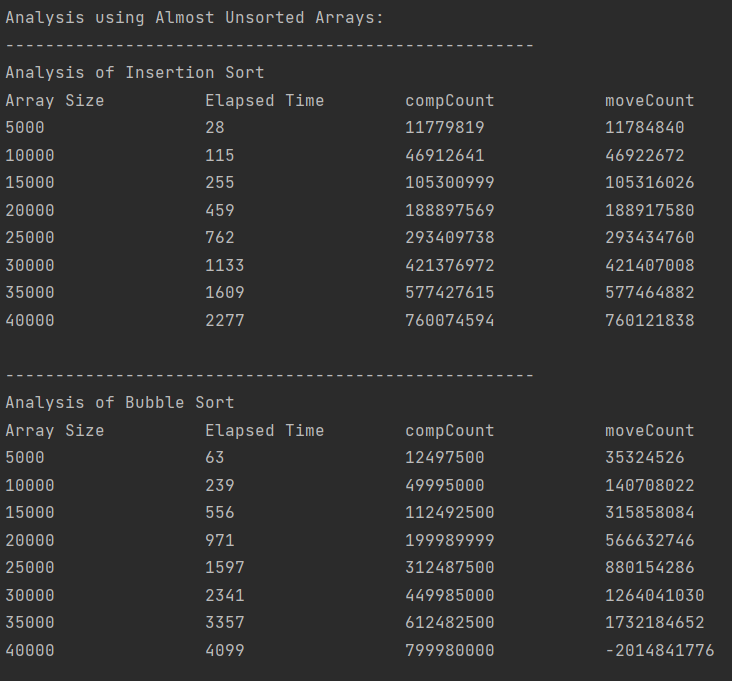
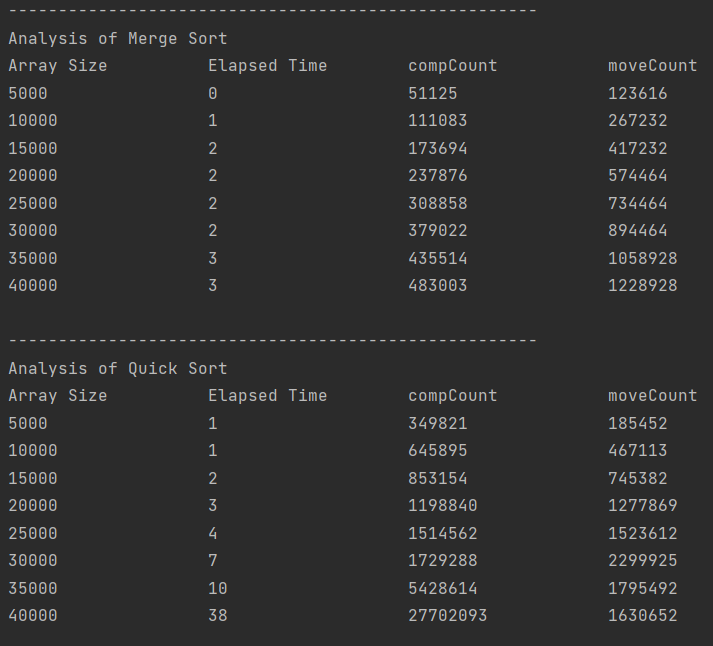
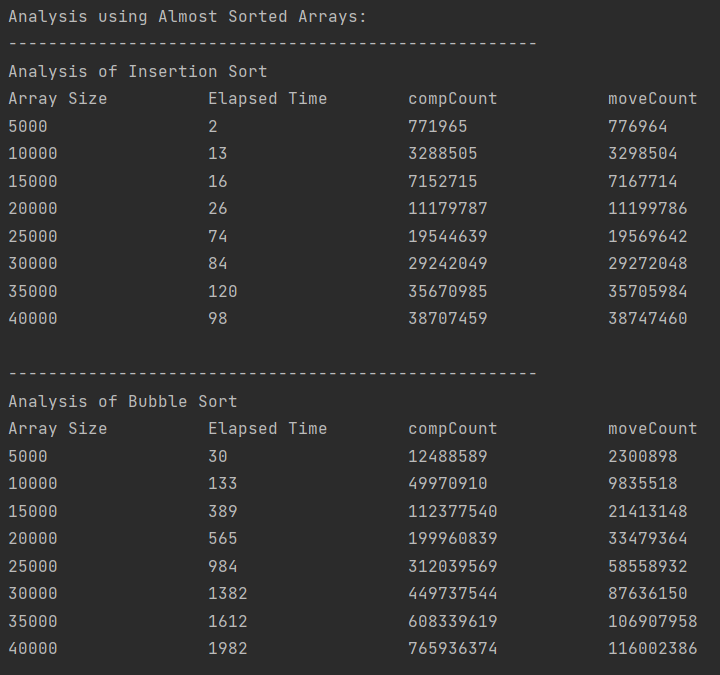
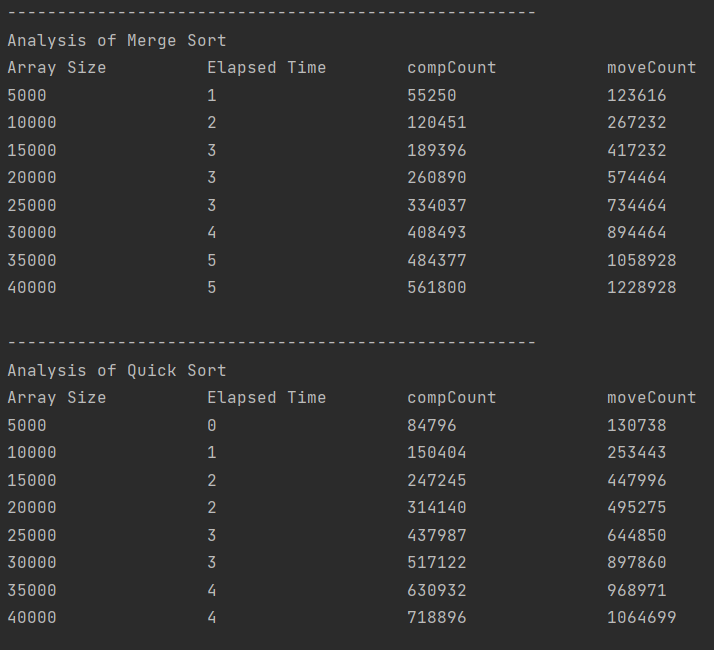
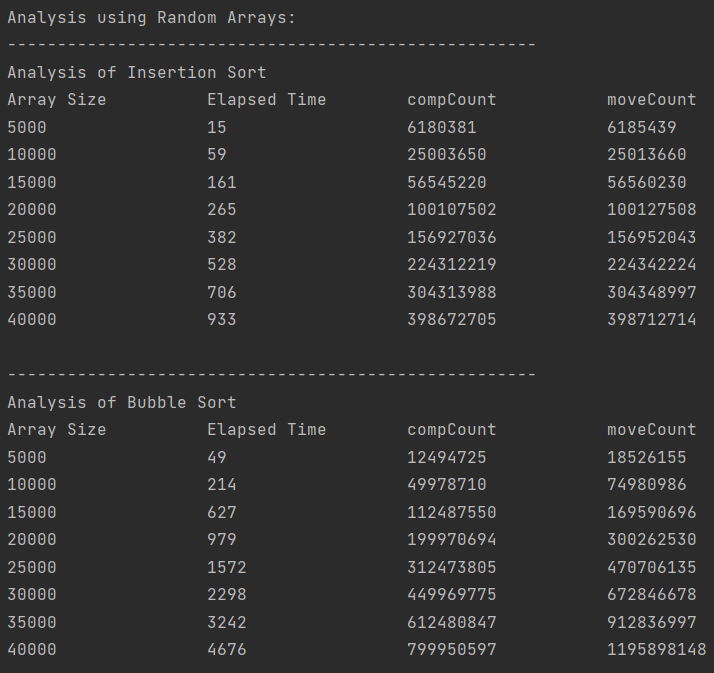
Question 2:

**c)**

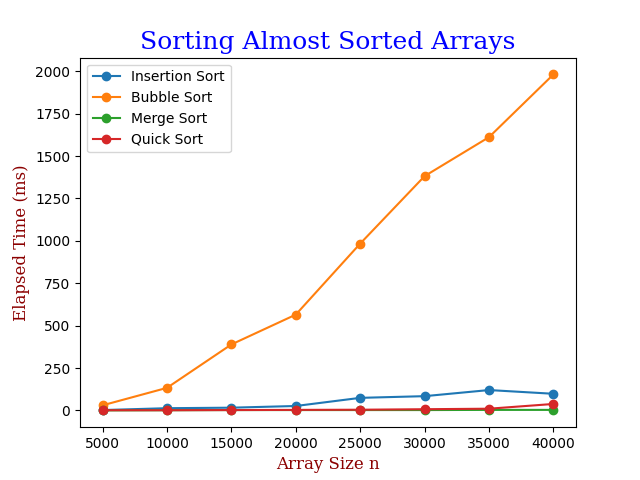
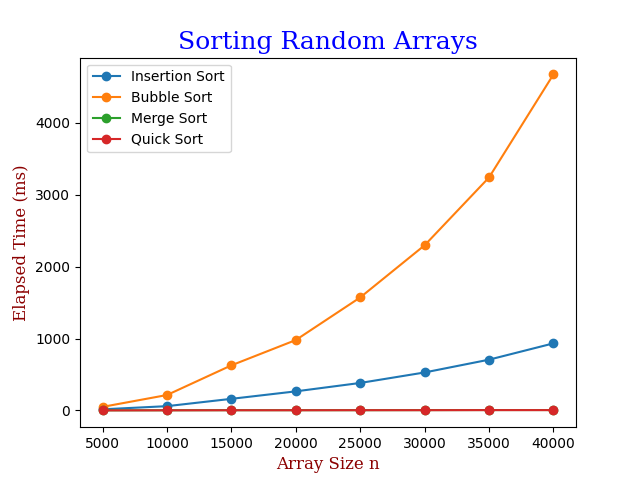
Text

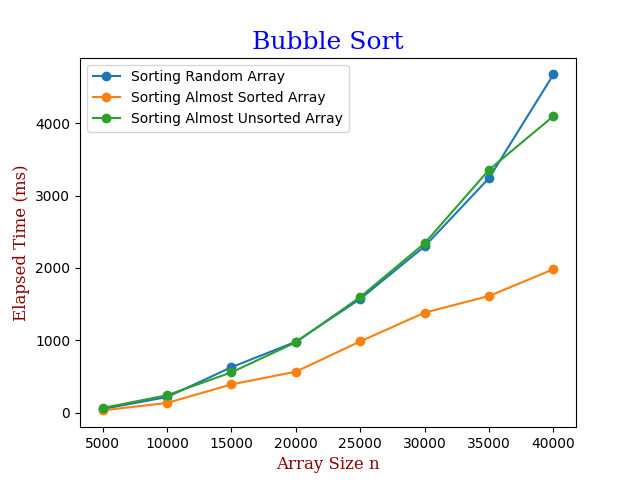
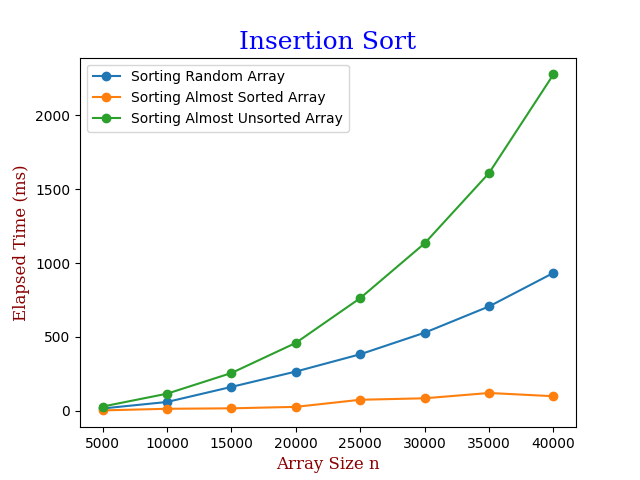
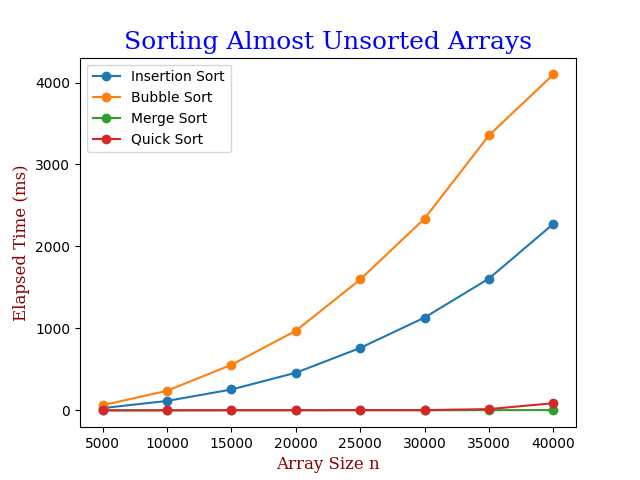
Description automatically generated with medium confidence

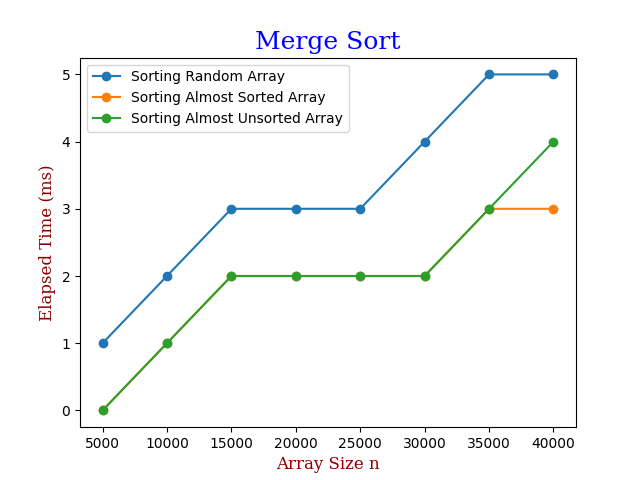
**d)**

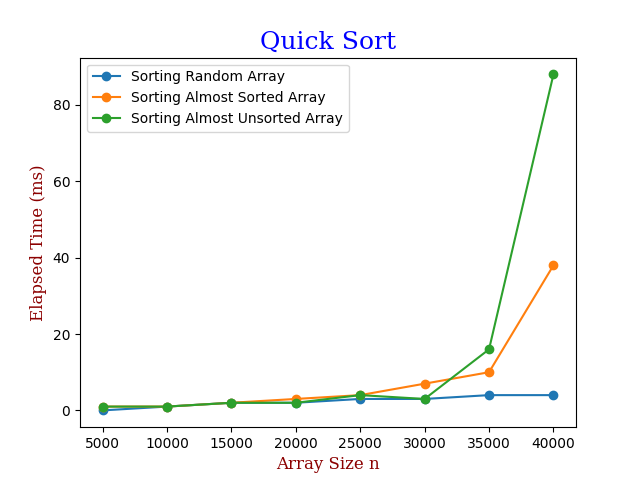


Question 3:









**Report**

Theoretically, we expect that insertion sort is for best case, for average and worst cases. From experimental results, we see that for random array (average case) it looks like parabola, for almost sorted array (best case) it looks like line and for almost unsorted array (worst case) it even more parabola than average case. That is theoratical = experimental values.

Theoretically, we expect that bubble sort is for best case, for average and worst cases. From experimental results, we see that for random array (average case) it looks like parabola, for almost sorted array (best case) it looks like line but it’s slope much larger than insertion sort’s best case slope and for almost unsorted array (worst case) it very much resembles average case line. That is theoratical = experimental values.

Theoretically, merge sort is for all possible cases. From experimental results, ladder-like lines suggest that it is not linear but some multiplication of n and for our case, factor should be . Therefore, merge sort is also experimentally correct.

Theoretically, we expect that quick sort is for best and average case, for worst case. From experimental results, almost sorted array (worst case) shows parabola and we expect that the others should be ladder-like lines. Our claim is true for random array, but not for almost unsorted array which has some outlier value at the end. Ignoring the last value of almost unsorted array we can see that quick sort is experimentally correct.

Between 4 sorting algorithms we see that merge and quick sort is much more efficient that insertion and bubble sorts which we expect from their time complexities. It seems among them the worst algorithm is bubble sort and the best is merge sort. This homework clearly shows that how and much more efficient than since even in very high values the first two ones not change much.