CS202-HW1(Utku Kurtulmus-21903025)

Question 1

a) Take c = 20

Then ==> 8x^4 + 5x^3 + 7 <= 20x^5

==> -20x^5 + 8x^4 +5x^3 + 7 <= 0

==> (x-1)(-20x^4 - 12x^3 -7x^2 –7x –7) <= 0

==>(x-1)(20x^4 + 12x^3 + 7x^2 + 7x + 7) >= 0

Notice that for x > 0 the right multiplier of the left hand is always positive, and for x > 1 the left side is always positive. For c = 20 and n0 = 1, we showed that n^5 is a upper bound of f(n). Hence, F(n) = O(n^5).

b) the array [22, 8, 49, 25, 18, 30, 20, 15, 35, 27]

Selection Sort

Note: [unsorted | sorted]

//find maximum element in unsorted region, swap it with the end of the element at
//unsorted region. Update the size of unsorted value.

==> [**22**, 8, 49, 25, 18, 30, 20, 15, 35, 27 |]

Max Element = 22,

==> [22, **8**, 49, 25, 18, 30, 20, 15, 35, 27 |]

Max Element = 22,

==> [22, 8, **49**, 25, 18, 30, 20, 15, 35, 27 |]

Max Element = 49,

*

*

*

==> [22, 8, 49, 25, 18, 30, 20, 15, 35, **27** |]

Max Element = 49,

Swap, Update

==> [**22**, 8, 27, 25, 18, 30, 20, 15, 35, | 49]

Max Element = 22,

*

*

*

==> [22, 8, 27, 25, 18, 30, 20, 15, **35**, | 49]

Max Element = 35

Swap, Update

==> [**22**, 8, 27, 25, 18, 30, 20, 15, | 35, 49]

*

*

*

//From now on I'll show only last swap/update steps since the logic is the same and It //will take too much space

==> [22, 8, 27, 25, 18, **15**, 20, | **30**, 35, 49]

==> [22, 8, **20**, 25, 18, 15, | **27**, 30, 35, 49]

==> [22, 8, 20, **15**, 18, | **25,** 27, 30, 35, 49]

==> [**18**, 8, 20, 15, | **22,** 25, 27, 30, 35, 49]

==> [18, 8, **15**, | **20,** 22, 25, 27, 30, 35, 49]

==> [**15**, 8, | **18**, 20**,** 22, 25, 27, 30, 35, 49]

==> [**8**, |**15**, 18, 20**,** 22, 25, 27, 30, 35, 49]

==> [| 8, 15, 18, 20, 22, 25, 27, 30, 35, 49]

BubbleSort

Note: [unsorted | sorted]

//The idea is similar to selection sort, we will try to put the biggest element to the //end but this time we will **compare a pair of data** and **swap them if the left hand side** //**is bigger than the right side**.

==> [**22, 8**, 49, 25, 18, 30, 20, 15, 35, 27 |]

==> [8, **22, 49**, 25, 18, 30, 20, 15, 35, 27 |]

==> [8, 22, **49, 25**, 18, 30, 20, 15, 35, 27 |]

==> [8, 22, 25, **49, 18**, 30, 20, 15, 35, 27 |]

==> [8, 22, 25, 18, **49, 30**, 20, 15, 35, 27 |]

==> [8, 22, 25, 18, 30, **49, 20**, 15, 35, 27 |]

==> [8, 22, 25, 18, 30, 20, **49, 15**, 35, 27 |]

==> [8, 22, 25, 18, 30, 20, 15, **49, 35**, 27 |]

==> [8, 22, 25, 18, 30, 20, 15, 35, **49, 27** |]

==> [**8, 22**, 25, 18, 30, 20, 15, 35, 27, | 49]

==> [8, **22, 25**, 18, 30, 20, 15, 35, 27, | 49]

==> [8, 22, **25, 18**, 30, 20, 15, 35, 27, | 49]

==> [8, 22, 18, **25, 30**, 20, 15, 35, 27, | 49]

==> [8, 22, 18, 25, **30, 20**, 15, 35, 27, | 49]

==> [8, 22, 18, 25, 20, **30, 15**, 35, 27, | 49]

==> [8, 22, 18, 25, 20, 15, **30, 35**, 27, | 49]

==> [8, 22, 18, 25, 20, 15, 30, **35, 27**, | 49]

==> [**8, 22**, 18, 25, 20, 15, 30, 27, | 35, 49]

==> [8, **22, 18**, 25, 20, 15, 30, 27, | 35, 49]

==> [8, 18, **22, 25**, 20, 15, 30, 27, | 35, 49]

==> [8, 18, 22, **25, 20**, 15, 30, 27, | 35, 49]

==> [8, 18, 22, 20, **25, 15**, 30, 27, | 35, 49]

==> [8, 18, 22, 20, 15, **25, 30**, 27, | 35, 49]

==> [8, 18, 22, 20, 15, 25, **30, 27**, | 35, 49]

==> [**8, 18**, 22, 20, 15, 25, 27, | 30, 35, 49]

==> [8, **18, 22**, 20, 15, 25, 27, | 30, 35, 49]

==> [8, 18, **22, 20**, 15, 25, 27, | 30, 35, 49]

==> [8, 18, 20, **22, 15**, 25, 27, | 30, 35, 49]

==> [8, 18, 20, 15, **22, 25**, 27, | 30, 35, 49]

==> [8, 18, 20, 15, 22, **25, 27**, | 30, 35, 49]

==> [**8, 18**, 20, 15, 22, 25, |27, 30, 35, 49]

==> [8, **18, 20**, 15, 22, 25, |27, 30, 35, 49]

==> [8, 18, **20, 15**, 22, 25, |27, 30, 35, 49]

==> [8, 18, 15, **20, 22**, 25, |27, 30, 35, 49]

==> [8, 18, 15, 20, **22, 25**, |27, 30, 35, 49]

==> [**8, 18**, 15, 20, 22, | 25, 27, 30, 35, 49]

==> [8, **18, 15**, 20, 22, | 25, 27, 30, 35, 49]

==> [8, 15, **18, 20**, 22, | 25, 27, 30, 35, 49]

==> [8, 15, 18, **20, 22**, | 25, 27, 30, 35, 49]

==> [**8, 15**, 18, 20, | 22, 25, 27, 30, 35, 49]

==> [8, **15, 18,** 20, | 22, 25, 27, 30, 35, 49]

==> [8, 15, **18, 20,** | 22, 25, 27, 30, 35, 49]

==> [8, 15, 18, 20, 22, 25, 27, 30, 35, 49] //Since we didn’t do any swap, we’re done.

Question 2

<u>Main.cpp output, dijkstra</u>.

```
[utku.kurtulmus@dijkstra HW1_dijkstra]$ make
g++ sorting.cpp main.cpp -o hw1
[utku.kurtulmus@dijkstra HW1_dijkstra]$ ./hw1
=================================
Calling the insertion sort......

Number of key comparisons: 73
Number of data moves: 88
contents of the array: 1 2 4 5 6 7 8 9 11 12 13 16 16 17 18 20
=================================
Calling the bubble sort......

Number of key comparisons: 110
Number of data moves: 174
contents of the array: 1 2 4 5 6 7 8 9 11 12 13 16 16 17 18 20
=================================
Calling the merge sort......

Number of key comparisons: 47
Number of data moves: 128
contents of the array: 1 2 4 5 6 7 8 9 11 12 13 16 16 17 18 20
=================================
Calling the quick sort......

Number of key comparisons: 50
Number of data moves: 125
contents of the array: 1 2 4 5 6 7 8 9 11 12 13 16 16 17 18 20
[utku.kurtulmus@dijkstra HW1_dijkstra]$
```

Algorithm Analysis Output, (MyComputer)

Random Array

C:\Users\utkuk\OneDrive\Masa³st³\Ders\cs202\HW1\HW1_Sorting\bin\Debug\HW1_Sorting.exe

```
=========================
CASE: Random Array

-----------------------------------------------------
Analysis of Insertion Sort
ArraySize       ElapsedTime(ms)       compCount       moveCount
5000            22                    6397319         6402318
10000           67                    25087950        25097949
15000           145                   56094023        56109022
20000           256                   100008508       100028507
25000           383                   155648483       155673482
30000           601                   225552422       225582421
35000           750                   304313998       304348997
40000           991                   398672715       398712714

-----------------------------------------------------
Analysis of Bubble Sort
ArraySize       ElapsedTime(ms)       compCount       moveCount
5000            75                    12486175        19176960
10000           284                   49966080        75233853
15000           634                   112489944       168237072
20000           1126                  199948959       299965527
25000           1737                  312473639       466870452
30000           2483                  449935230       676567269
35000           3348                  612480847       912836997
40000           4388                  799950597       1195898148

-----------------------------------------------------
Analysis of Merge Sort
ArraySize       ElapsedTime(ms)       compCount       moveCount
5000            0                     55176           123616
10000           2                     120490          267232
15000           3                     189262          417232
20000           4                     260957          574464
25000           5                     333960          734464
30000           6                     408582          894464
35000           8                     484377          1058928
40000           10                    561800          1228928

-----------------------------------------------------
Analysis of Quick Sort
ArraySize       ElapsedTime(ms)       compCount       moveCount
5000            1                     75892           116775
10000           1                     154289          237460
15000           1                     235751          345057
20000           2                     326930          518112
25000           3                     422066          701546
30000           3                     567272          939524
35000           4                     630932          968971
40000           4                     718896          1064699
```

Almost Sorted Array

```
CASE: Almost Sorted Array

--------------------------------------------------
Analysis of Insertion Sort
ArraySize       ElapsedTime(ms)      compCount       moveCount
5000             2                    771965          776964
10000            8                    3288505         3298504
15000           18                    7152715         7167714
20000           27                   11179787        11199786
25000           48                   19544643        19569642
30000           72                   29242049        29272048
35000           88                   35670985        35705984
40000           93                   38707461        38747460

--------------------------------------------------
Analysis of Bubble Sort
ArraySize       ElapsedTime(ms)      compCount       moveCount
5000            30                   12488589         2300898
10000           124                  49970910         9835518
15000           272                  112377540       21413148
20000           479                  199960839       33479364
25000           752                  312039569       58558932
30000          1097                  449737544       87636150
35000          1450                  608339619       106907958
40000          1800                  765936374       116002386

--------------------------------------------------
Analysis of Merge Sort
ArraySize       ElapsedTime(ms)      compCount       moveCount
5000             1                    51125           123616
10000            1                    111083          267232
15000            2                    173694          417232
20000            2                    237876          574464
25000            4                    308858          734464
30000            5                    379022          894464
35000            7                    435514         1058928
40000            8                    483003         1228928

--------------------------------------------------
Analysis of Quick Sort
ArraySize       ElapsedTime(ms)      compCount       moveCount
5000             1                    349821          185452
10000            1                    645895          467113
15000            2                    853154          745382
20000            4                   1198840         1277869
25000            4                   1514562         1523612
30000            5                   1729288         2299925
35000           12                   5428614         1795492
40000           53                  27702093         1630652
```

Almost Unsorted Array

CASE: Almost Unsorted Array

Analysis of Insertion Sort

ArraySize	ElapsedTime(ms)	compCount	moveCount
5000	31	11779841	11784840
10000	117	46912673	46922672
15000	252	105301027	105316026
20000	456	188897581	188917580
25000	712	293409761	293434760
30000	1022	421377009	421407008
35000	1405	577429883	577464882
40000	1848	760081839	760121838

Analysis of Bubble Sort

ArraySize	ElapsedTime(ms)	compCount	moveCount
5000	53	12497500	35324526
10000	205	49995000	140708022
15000	463	112492500	315858084
20000	832	199989999	566632746
25000	1307	312487500	880154286
30000	1883	449985000	1264041030
35000	2557	612482500	1732184652
40000	3326	799980000	-2014841776

Analysis of Merge Sort

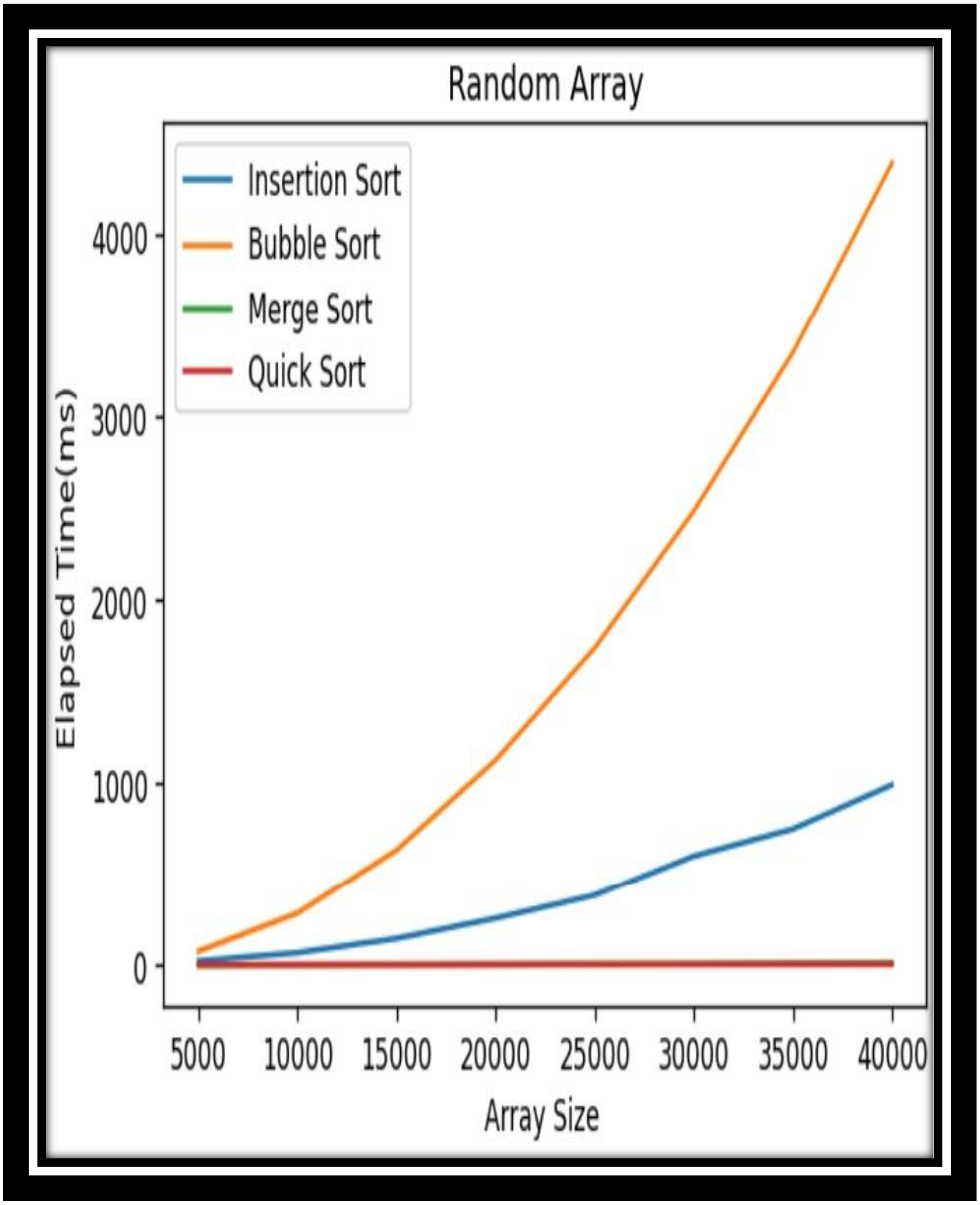
ArraySize	ElapsedTime(ms)	compCount	moveCount
5000	1	49547	123616
10000	1	108691	267232
15000	2	172531	417232
20000	2	237617	574464
25000	3	306583	734464
30000	5	376312	894464
35000	6	435721	1058928
40000	8	484490	1228928

Analysis of Quick Sort

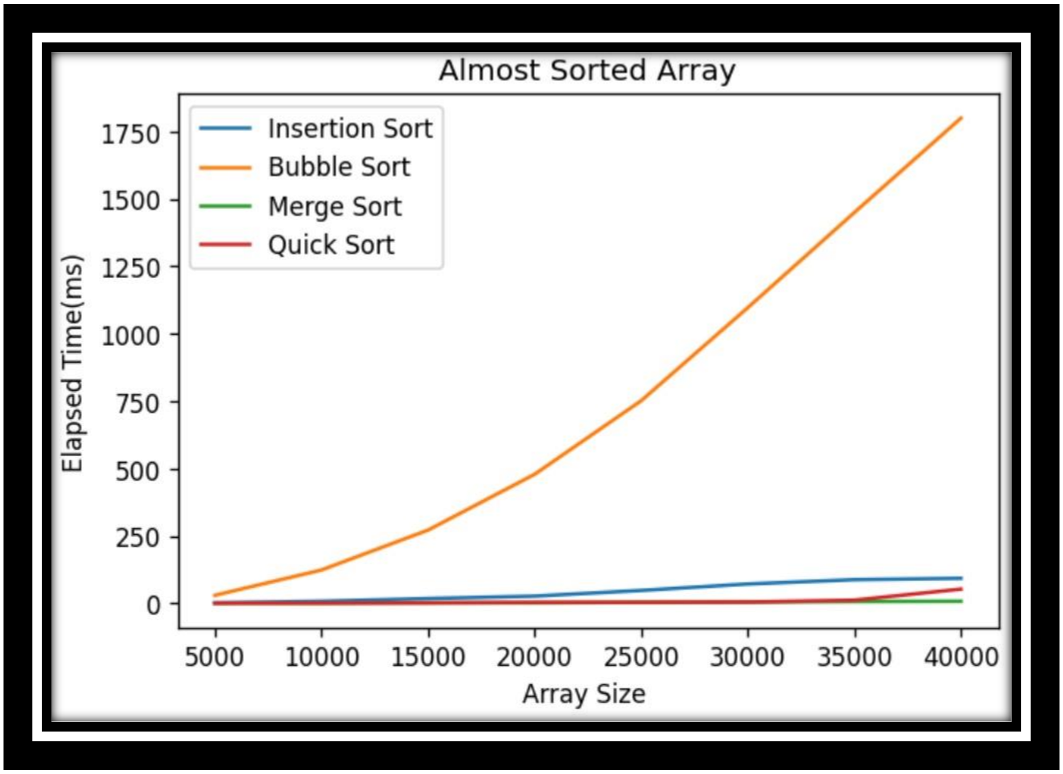
ArraySize	ElapsedTime(ms)	compCount	moveCount
5000	0	121483	191358
10000	1	367813	585540
15000	1	571278	892085
20000	2	568573	914815
25000	4	1055059	1727286
30000	3	873315	1406946
35000	14	4944236	7538032
40000	73	27405350	41251057

Question 3

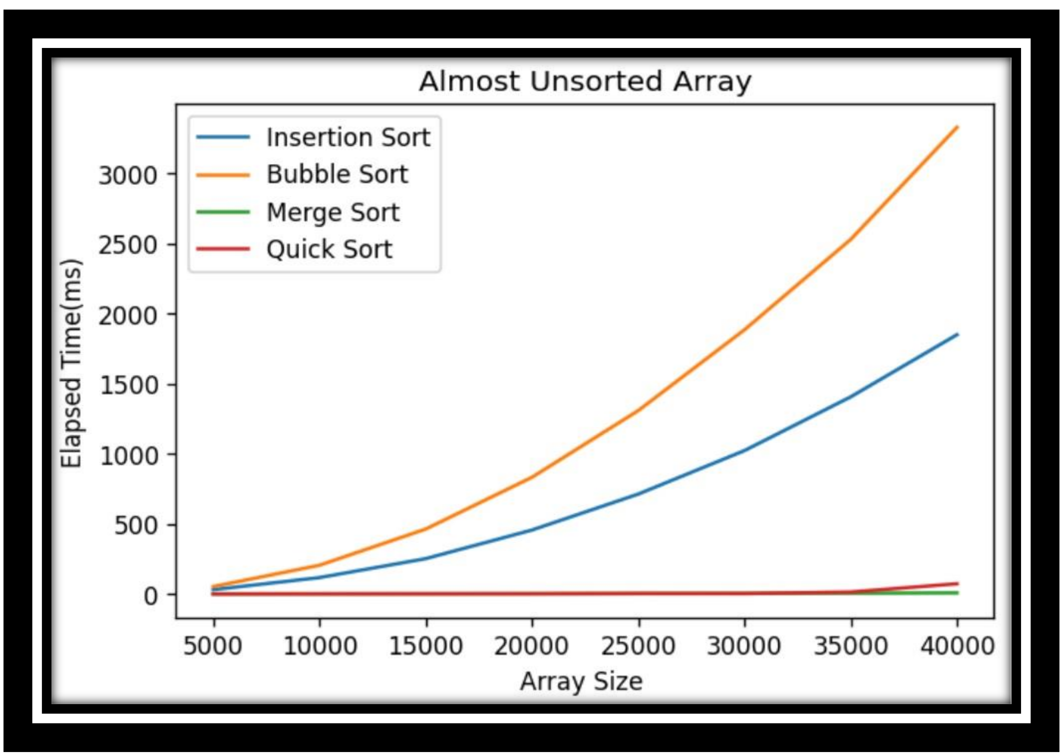
PLOTS



//Merge and Quick Sort almost have the same behavior, so we can't clearly see Merge Sort //here.



//Now you can see, the green line (Merge Sort).



In terms of insertion sort, We had great experimental results to observe it's behavior. In the best case of the (already sorted array) insertion sort, we know that time complexity is o(N) since we won't move any data and just do one traversal. We can see similar behavior on the graph of almost sorted array. It's even seems like a constant time complexity due to the fact that bubble sort increasing dramatically, it seems like constant compared to bubble sort. But if we could get closer to the plot we would see it's increasing linearly. You can also look the empirical data on question 2 to confirm it. Now in the average and worst case the complexity is o(N^2) but of course in the worst case (unsorted array) it takes more time to sort the data compared to average (random array) case.

In terms of bubble sort, We haven't able to observe the best case perfectly (sorted array) since the array wasn't fully sorted; we had to do swap operations, and even one swap operation will cause another traversal. We expected that in the best case time complexity would be o(N) since we would do one traversal and we are done. In terms of the average and worst case we are satisfied with the results. We can clearly see that the time complexity is O(n^2). To see it more clearly you can look to the empirical data on Q2. When the array size doubled, the elapsed time was multiplied by 2^2. This is true for the all cases (Almost sorted-unsorted, random). However, the total time has changed in these 3 cases. For instance, in the almost sorted array we had shorter elapsed time results whereas in the random and unsorted arrays we had longer. One interesting thing that I can't fully understand we had slightly higher elapsed time in random array compared to almost unsorted array, however we did more data moves in almost unsorted array. We even had integer overflow for the array size 40000. And comp counts are almost identical. So maybe my computer was working faster in the almost unsorted array case, I am not sure.

In terms of quick and merge sort, I didn't find this experiment enough to observe their behaviors due to small array sizes. But still, the data on experiment 2 gives good results in terms of number of key comparisons and data moves. Merge sort is O(n*logn) in all cases, and we can see similar behavior on question 2. On the plot both merge and quick sort seems like constant time O(1) because of the bubble sort took too much time. I also run this code on Dijkstra machine and elapsed time results was just a joke (I.e 0,10,0,0,0,10....) but the key comparisons and data moves were similar. Hence, I believe observing the key comparisons and data moves is better than the elapsed time. For quicksort, we know it is O(n^2) in the worst case (already sorted array) when we choose the first element as pivot. And for average and best case it is O(n*logn) similar to merge sort. In the experiment we found that Quicksort was good for random array case. And for almost unsorted and almost sorted array cases it had similar behavior and they were both worse than random array case. We expected this result for the almost sorted array. For the almost unsorted array, the number of key comparisons and data moves wasn't consistent with the array size, for instance when we increased the array size from 25000 to 30000 elapsed time, key comp, and data move results decreased. So, It is important how the array formed after we call the 'createAlmostUnsortedArray' function. Furthermore, for the almost sorted and unsorted arrays, elapsed time started to increase dramatically after a certain array size (>35000). I believe to observe the behavior of these algoritms, we should have bigger array sizes, so that small changes in elapsed time due to other reasons (heating etc.) wouldn't have a considerable effect on it and we would have clearer results.