BILKENT UNIVERSITY CS202

HOMEWORK 1

İrem Seven

Section : 1

ID: 21704269

Part 1)

a)

We need to find two positive constants c and n0 such that

0 <= 20 n^4 + 20 n^2 + 5 <= n^5

for all n >= n0

dividing both sides with n^5 we can see that to make c constant integer we can put n0 = 1 into n.

Then, with n0 = 1 we get c = 45, for all n >= 1.

b)

Selection Sort:

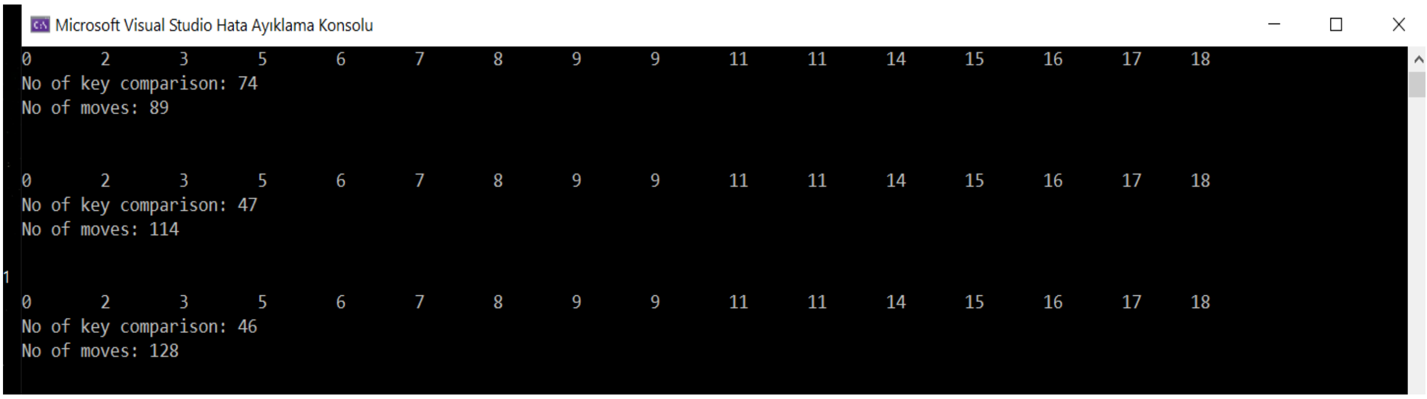
18	4	47	24	15	24	17	11	31	23	initial
18	4	**23**	24	15	24	17	11	31	**47**	
18	4	23	24	15	24	17	11	**31**	47	
18	4	23	**11**	15	24	17	**24**	31	47	
18	4	23	11	15	**17**	**24**	24	31	47	
18	4	**17**	11	15	**23**	24	24	31	47	
15	4	17	11	**18**	23	24	24	31	47	
15	4	**11**	**17**	18	23	24	24	31	47	
11	4	**15**	17	18	23	24	24	31	47	
11	**4**	15	17	18	23	24	24	31	47	
4	11	15	17	18	23	24	24	31	47	

Bubble Sort:

18	**4**	47	24	15	24	17	11	31	23	initial
4	**18**	**47**	24	15	24	17	11	31	23	
4	18	**47**	**24**	15	24	17	11	31	23	
4	18	24	**47**	**15**	24	17	11	31	23	
4	18	24	15	**47**	**24**	17	11	31	23	
4	18	24	15	24	**47**	**17**	11	31	23	
4	18	24	15	24	17	**47**	**11**	31	23	
4	18	24	15	24	17	11	**47**	**31**	23	
4	18	24	15	24	17	11	31	**47**	**23**	
4	**18**	24	15	24	17	11	31	23	47	
4	**18**	**24**	15	24	17	11	31	23	47	
4	18	**24**	**15**	24	17	11	31	23	47	
4	18	15	**24**	**24**	17	11	31	23	47	
4	18	15	24	**24**	**17**	11	31	23	47	
4	18	15	24	17	**24**	**11**	31	23	47	
4	18	15	24	17	11	**24**	**31**	23	47	
4	18	15	24	17	11	24	**31**	**23**	47	
4	**18**	**15**	24	17	11	24	23	31	47	
4	15	**18**	**24**	17	11	24	23	31	47	
4	15	18	**24**	**17**	11	24	23	31	47	
4	15	18	17	**24**	**11**	24	23	31	47	
4	15	18	17	11	**24**	**24**	23	31	47	
4	15	18	17	11	24	**24**	**23**	31	47	
4	**15**	**18**	17	11	24	23	24	31	47	
4	15	**18**	**17**	11	24	23	24	31	47	
4	15	17	**18**	**11**	24	23	24	31	47	
4	15	17	11	**18**	**24**	23	24	31	47	
4	15	17	11	18	**23**	**24**	24	31	47	

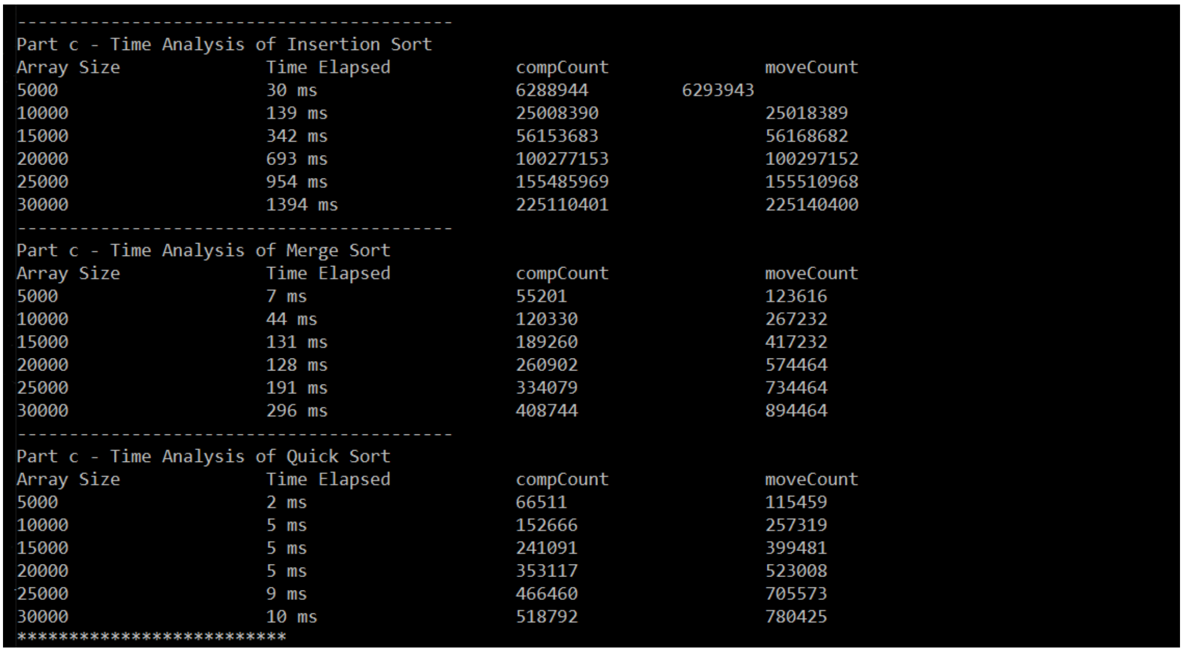
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	17	11	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	15	11	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47
4	11	15	17	18	23	24	24	31	47

Part 2)



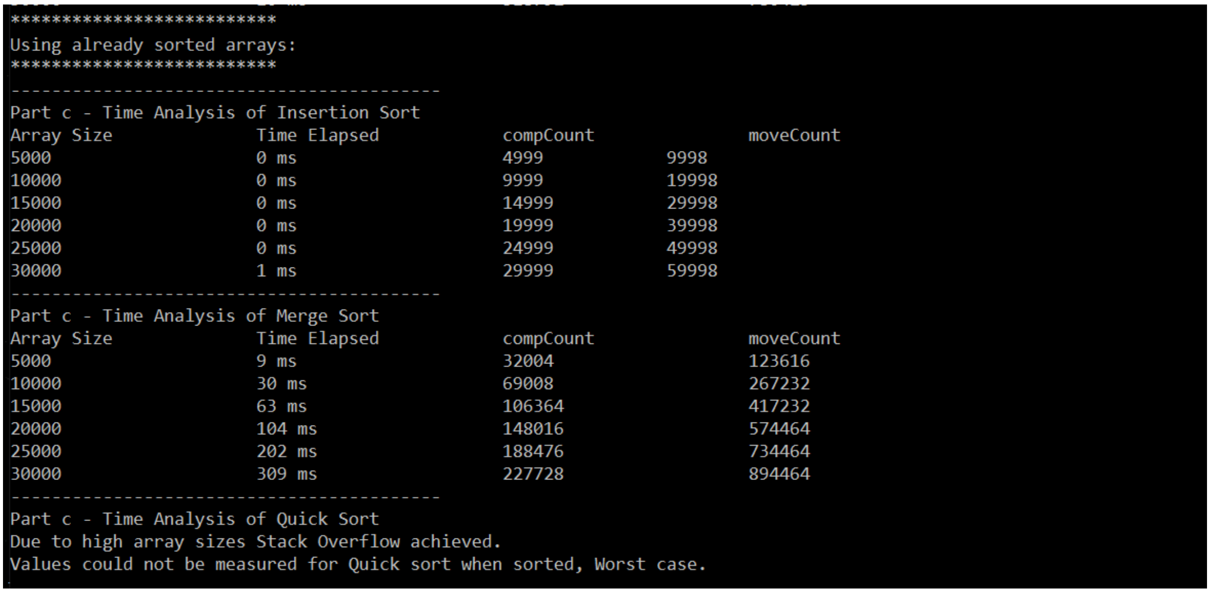
```
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
No of key comparison: 74
No of moves: 89


0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
No of key comparison: 47
No of moves: 114


0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
No of key comparison: 46
No of moves: 128
```

Figure1: Screenshot of Question 2 part a



```
-----------------------------------------
Part c - Time Analysis of Insertion Sort
Array Size              Time Elapsed            compCount               moveCount
5000                    30 ms                   6288944         6293943
10000                   139 ms                  25008390                25018389
15000                   342 ms                  56153683                56168682
20000                   693 ms                  100277153               100297152
25000                   954 ms                  155485969               155510968
30000                   1394 ms                 225110401               225140400
-----------------------------------------
Part c - Time Analysis of Merge Sort
Array Size              Time Elapsed            compCount               moveCount
5000                    7 ms                    55201                   123616
10000                   44 ms                   120330                  267232
15000                   131 ms                  189260                  417232
20000                   128 ms                  260902                  574464
25000                   191 ms                  334079                  734464
30000                   296 ms                  408744                  894464
-----------------------------------------
Part c - Time Analysis of Quick Sort
Array Size              Time Elapsed            compCount               moveCount
5000                    2 ms                    66511                   115459
10000                   5 ms                    152666                  257319
15000                   5 ms                    241091                  399481
20000                   5 ms                    353117                  523008
25000                   9 ms                    466460                  705573
30000                   10 ms                   518792                  780425
**************************
```

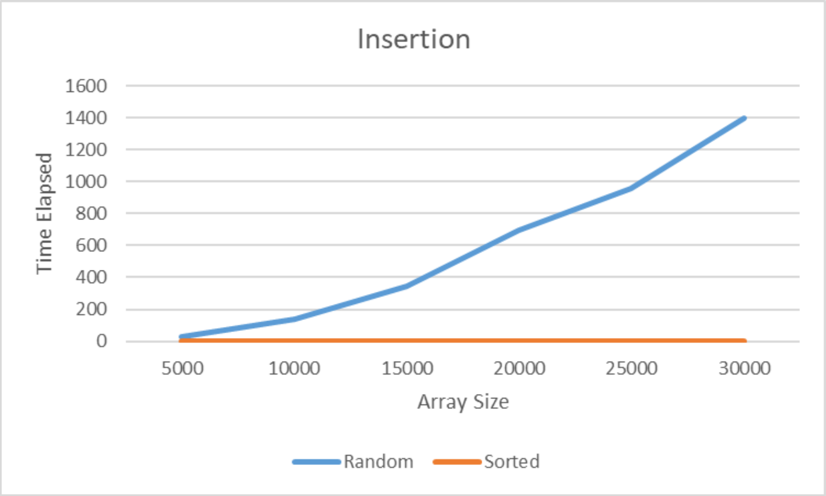
Figure2: Screenshot of Randomly Created Performance Analysis

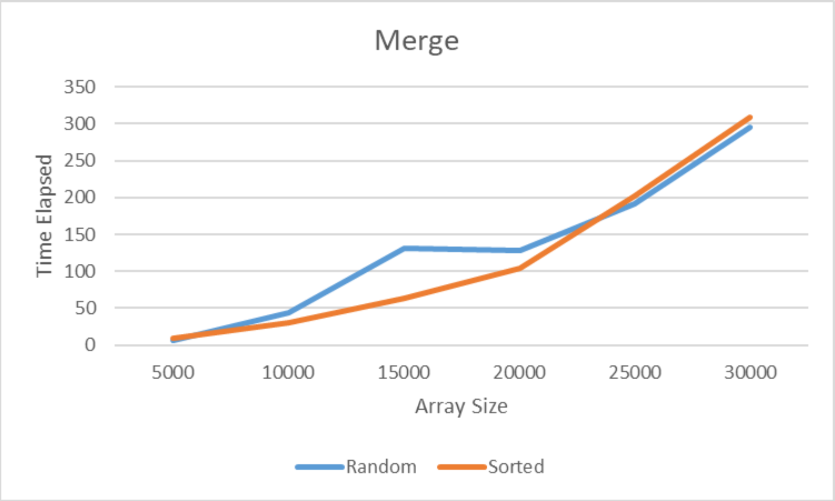


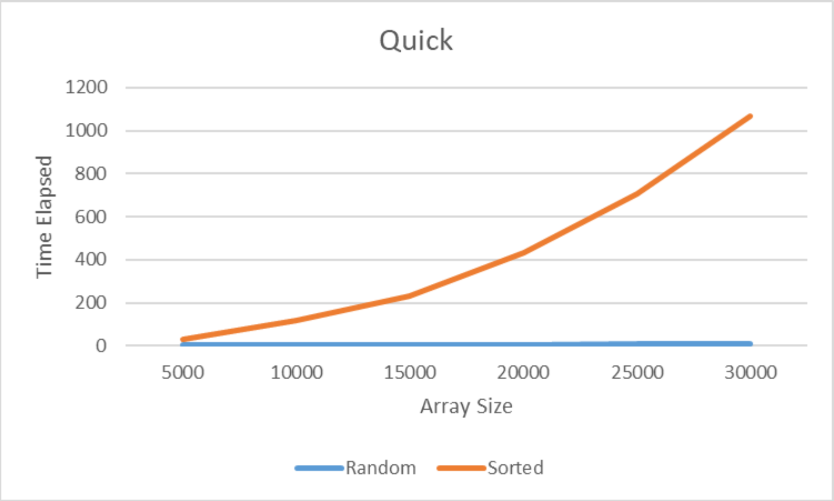
```
**************************
Using already sorted arrays:
**************************
-----------------------------------------
Part c - Time Analysis of Insertion Sort
Array Size              Time Elapsed            compCount               moveCount
5000                    0 ms                    4999            9998
10000                   0 ms                    9999            19998
15000                   0 ms                    14999           29998
20000                   0 ms                    19999           39998
25000                   0 ms                    24999           49998
30000                   1 ms                    29999           59998
-----------------------------------------
Part c - Time Analysis of Merge Sort
Array Size              Time Elapsed            compCount               moveCount
5000                    9 ms                    32004                   123616
10000                   30 ms                   69008                   267232
15000                   63 ms                   106364                  417232
20000                   104 ms                  148016                  574464
25000                   202 ms                  188476                  734464
30000                   309 ms                  227728                  894464
-----------------------------------------
Part c - Time Analysis of Quick Sort
Due to high array sizes Stack Overflow achieved.
Values could not be measured for Quick sort when sorted, Worst case.
```

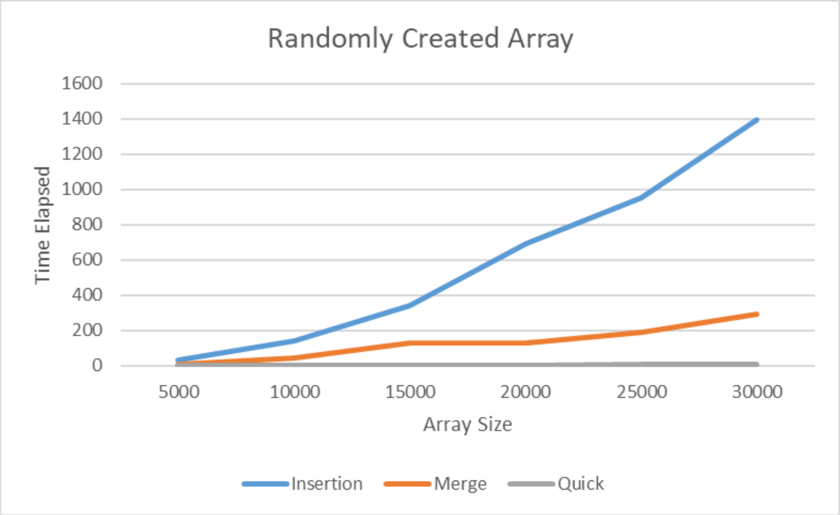
Figure3: Screenshot of Sorted Created Performance Analysis

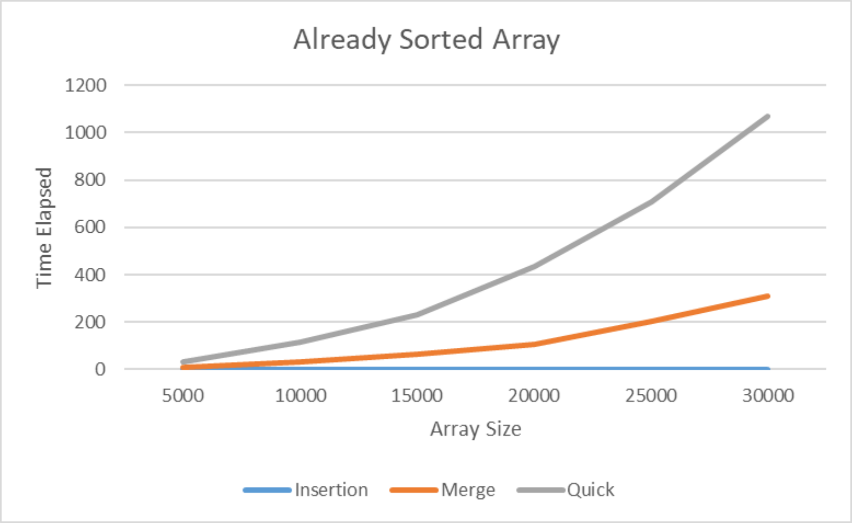
Plots:


Insertion
1600
1400
1200
1000
800
600
400
200
0
Time Elapsed
5000
10000
15000
20000
25000
30000
Array Size
Random
Sorted


Merge
350
300
250
200
150
100
50
0
Time Elapsed
5000
10000
15000
20000
25000
30000
Array Size
Random
Sorted


Quick
1200
1000
800
600
400
200
0
Time Elapsed
5000
10000
15000
20000
25000
30000
Array Size
Random
Sorted





Discussion:

As it can be seen in the plots, when considering randomly created array it can be said that quick sort performs way better than the merge and insertion sort algorithms. Also, merge sort performed way better than the insertion sort algorithm. Theoretically, the results seems logical. Merge sort and Quick sort algorithms have O(nlogn) time complexities which are less than Insertion sort time complexity O(n^2). Although their time complexities are same, quick sort seemed more efficient than the merge sort in the experiment. This can be resulted due to merge sort copying the array inside its merge function.

When already sorted array is used, it can be said that quick sort is the worst. Theoretically, quick sort has its worst case when the array items are already sorted and it gives O(n) time complexity. In experiment, it was not possible to obtain no of comparisons and no of moves since it results in stack over flow. Thus, the values in graphs are obtained separately only for time elapse values apart from the given code. On the other hand, Insertion sort’s efficiency is significantly increased when sorted array is used. This is because Insertion sort has its best case when the array elements are sorted, O(n). Regarding merge sort its worst, average and best case have all the same time complexity O(nlogn). Thus, measured time elapse did not change significantly for merge sort in both experiments.

Part 3)

Since the array is nearly sorted choosing merge sort will give the best efficient solution. If we choose key as n/2 it will result time complexity similar to merge sorts best case. Key should be half of the size when entering each recursive function and the target should be as close as possible to key. Thus, k should be as close to make the target as the merge sorts sublist middle. If k is 0 it will result in best.