Berk Saltuk Yılmaz - 21903419 - Sec03

Homework 1: Algorithm Efficiency and Sorting

Question-1

(a) Show that $5n^3 + 4n^2 + 10$ is $O\left(n^4\right)$ by specifying appropriate c and n_0 values in Big-O definition:

We need to find two positive c and n_0 values that hold for:

$$0 \leq 5n^3 + 4n^2 + 10 \leq c \cdot n^4 \text{ for all } n \geq n_0$$

and

$$\frac{5}{n} + \frac{4}{n^2} + \frac{10}{n^4} \leq c \text{ for all } n \geq n_0$$

If we choose, c = 5 and n_0 = 2 we get:

$$2.5 + 1 + 0.625 = 4.125 \leq 5$$

Thus,

$$5n^3 + 4n^2 + 10 \leq 5n^4 \text{ for all } n \geq 2$$

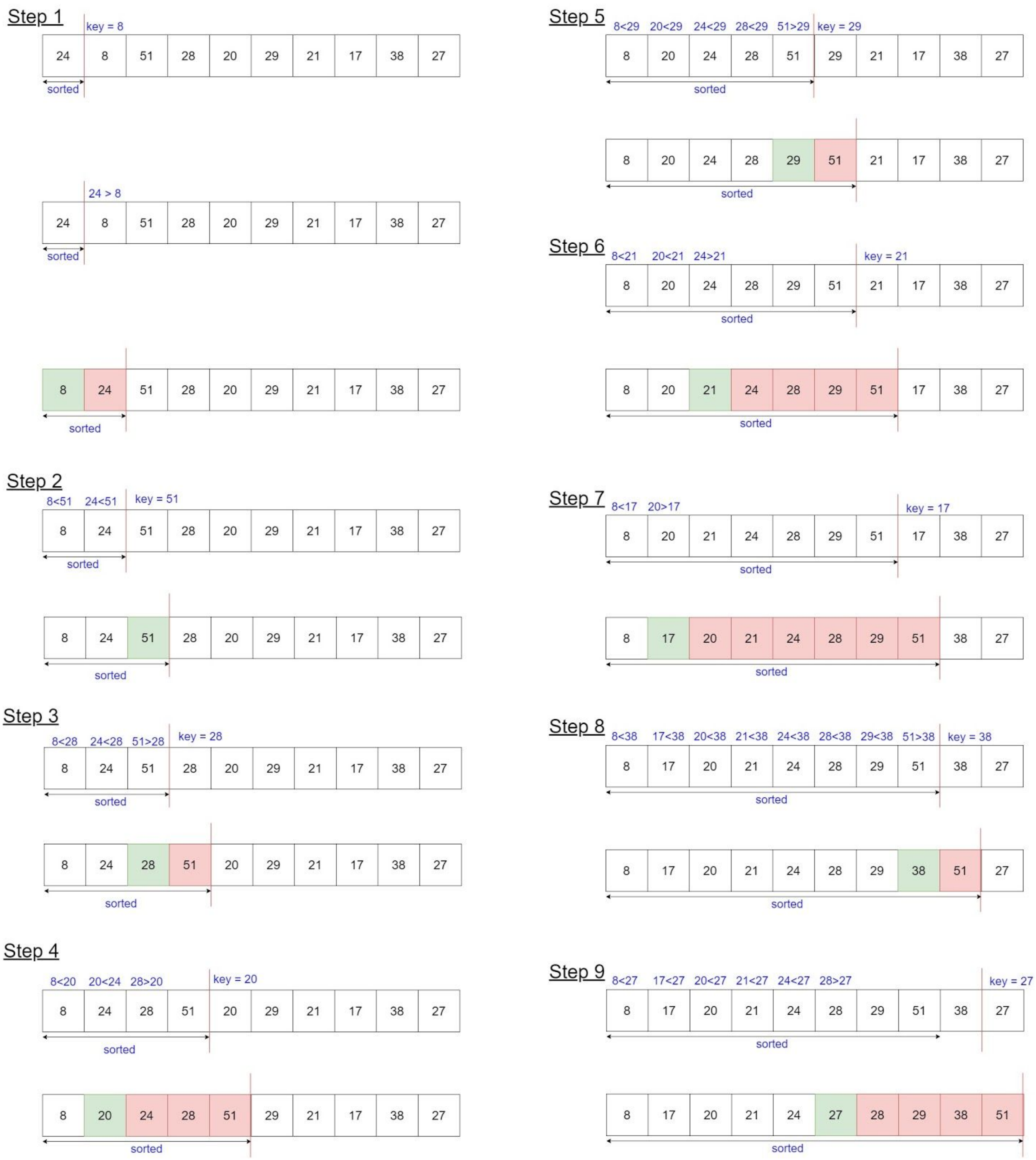
It can be said that $5n^3 + 4n^2 + 10$ is $O\left(n^4\right)$

(b) Trace the following sorting algorithms to sort the array

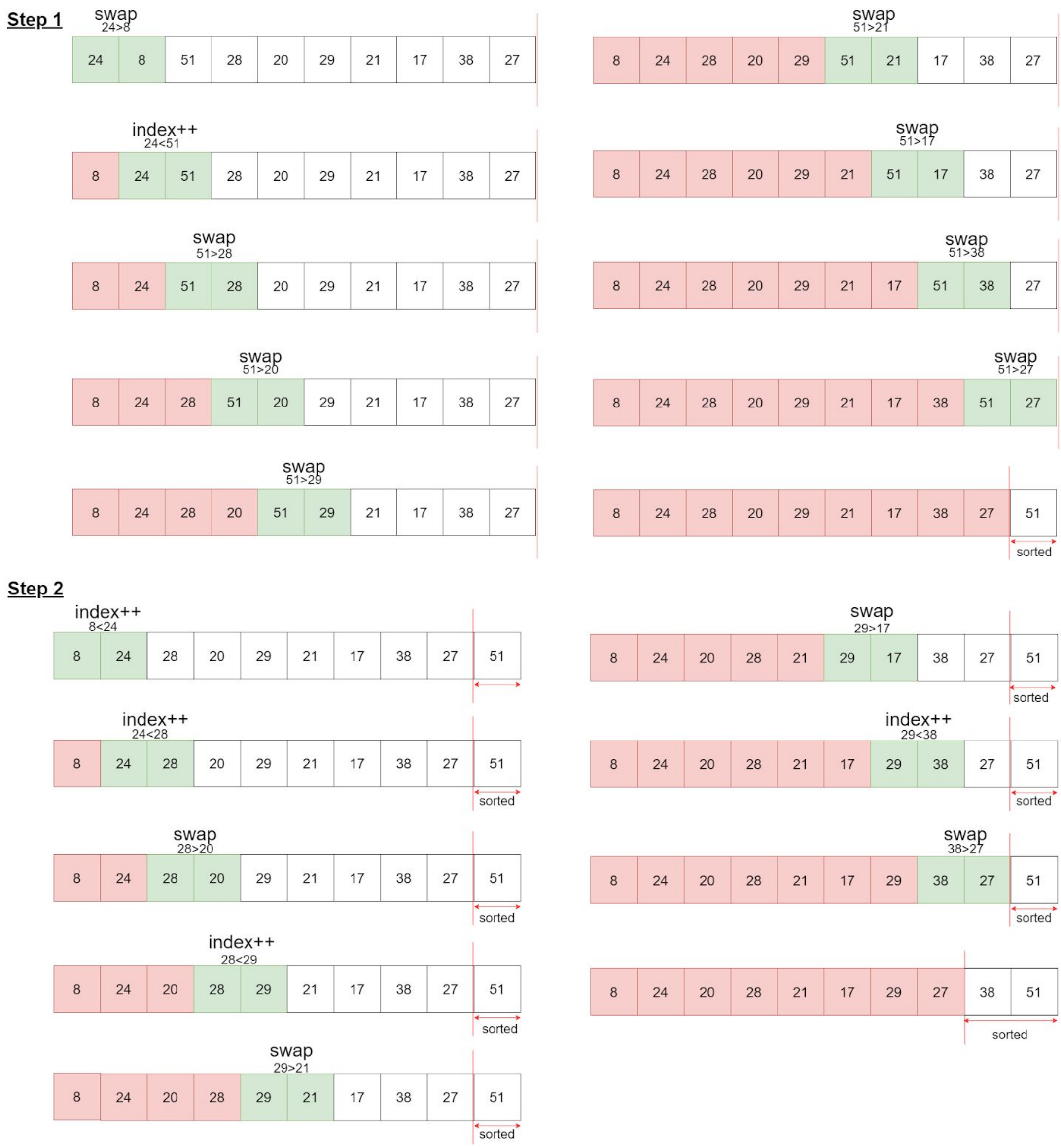
[24, 8, 51, 28, 20, 29, 21, 17, 38, 27]

in ascending order. Use the array implementation of the algorithms as described in the textbook and show all major steps.

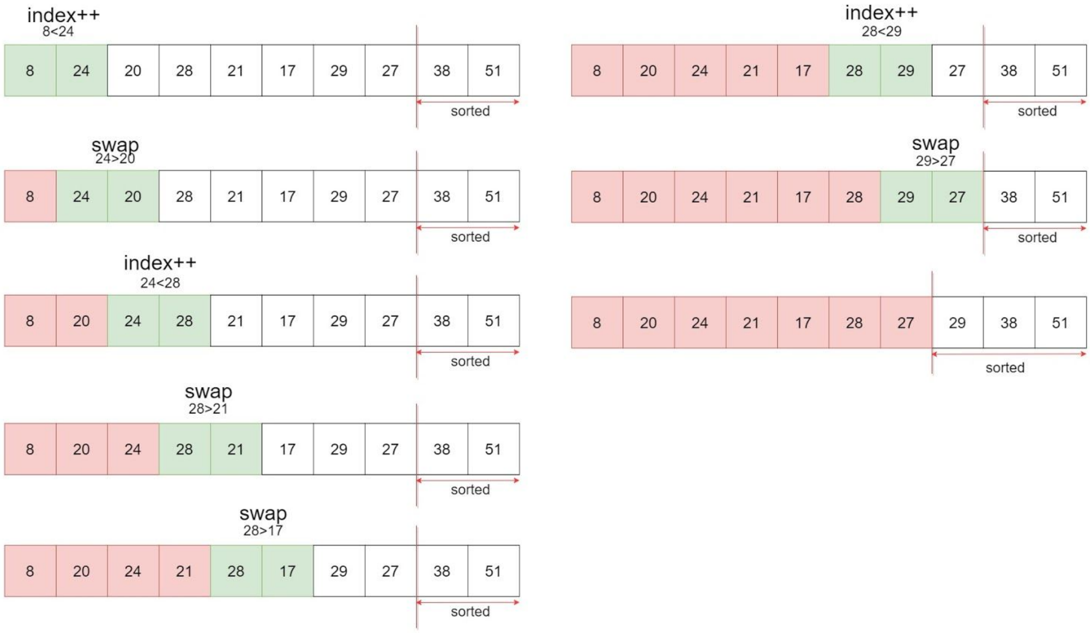
- Insertion sort:



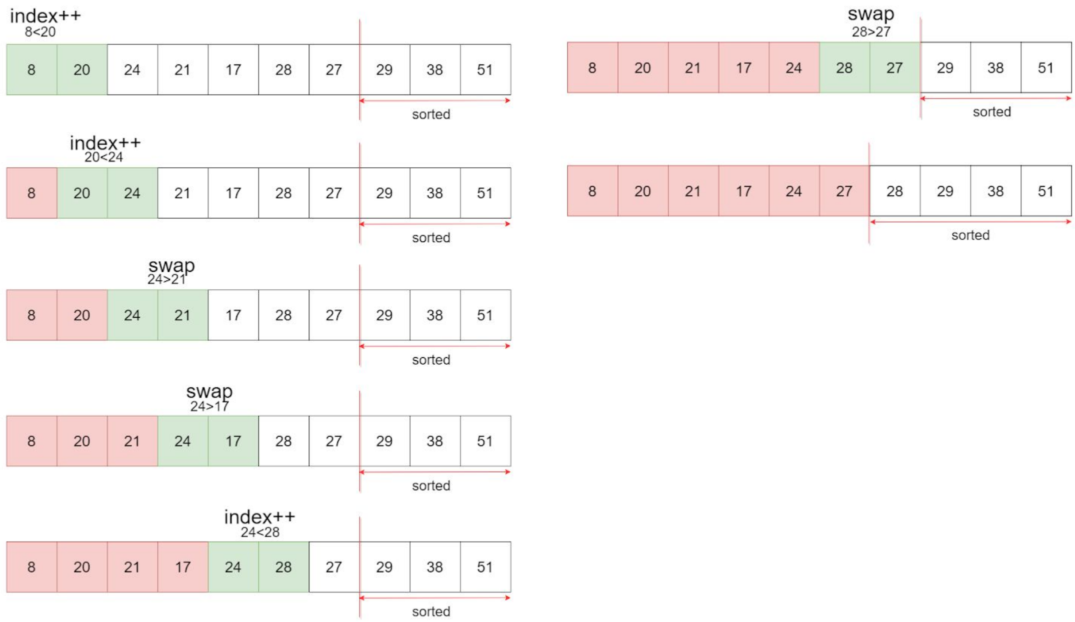
- Bubble sort:


Step 1
swap
24>8
24 8 51 28 20 29 21 17 38 27
index++
24<51
8 24 51 28 20 29 21 17 38 27
swap
51>28
8 24 51 28 20 29 21 17 38 27
swap
51>20
8 24 28 51 20 29 21 17 38 27
swap
51>29
8 24 28 20 51 29 21 17 38 27
swap
51>21
8 24 28 20 29 51 21 17 38 27
swap
51>17
8 24 28 20 29 21 51 17 38 27
swap
51>38
8 24 28 20 29 21 17 51 38 27
swap
51>27
8 24 28 20 29 21 17 38 51 27
8 24 28 20 29 21 17 38 27 51
sorted
Step 2
index++
8<24
8 24 28 20 29 21 17 38 27 51
index++
24<28
8 24 28 20 29 21 17 38 27 51
sorted
swap
28>20
8 24 28 20 29 21 17 38 27 51
sorted
index++
28<29
8 24 20 28 29 21 17 38 27 51
sorted
swap
29>21
8 24 20 28 29 21 17 38 27 51
sorted
swap
29>17
8 24 20 28 21 29 17 38 27 51
sorted
index++
29<38
8 24 20 28 21 17 29 38 27 51
sorted
swap
38>27
8 24 20 28 21 17 29 38 27 51
sorted
8 24 20 28 21 17 29 27 38 51
sorted

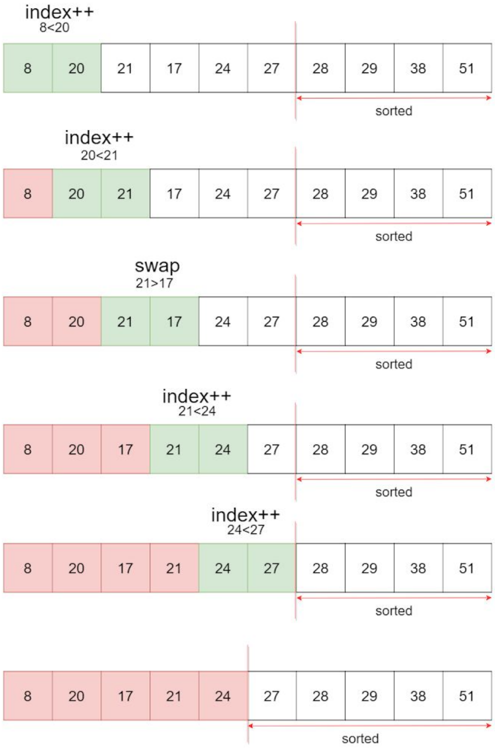
Step 3


index++
8<24
8 24 20 28 21 17 29 27 38 51
sorted
swap
24>20
8 24 20 28 21 17 29 27 38 51
sorted
index++
24<28
8 20 24 28 21 17 29 27 38 51
sorted
swap
28>21
8 20 24 28 21 17 29 27 38 51
sorted
swap
28>17
8 20 24 21 28 17 29 27 38 51
sorted
index++
28<29
8 20 24 21 17 28 29 27 38 51
sorted
swap
29>27
8 20 24 21 17 28 29 27 38 51
sorted
8 20 24 21 17 28 27 29 38 51
sorted

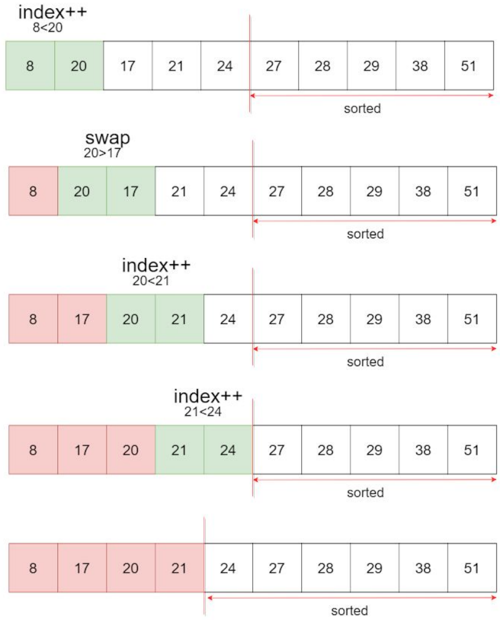
Step 4


index++
8<20
8 20 24 21 17 28 27 29 38 51
sorted
index++
20<24
8 20 24 21 17 28 27 29 38 51
sorted
swap
24>21
8 20 24 21 17 28 27 29 38 51
sorted
swap
24>17
8 20 21 24 17 28 27 29 38 51
sorted
index++
24<28
8 20 21 17 24 28 27 29 38 51
sorted
swap
28>27
8 20 21 17 24 28 27 29 38 51
sorted
8 20 21 17 24 27 28 29 38 51
sorted

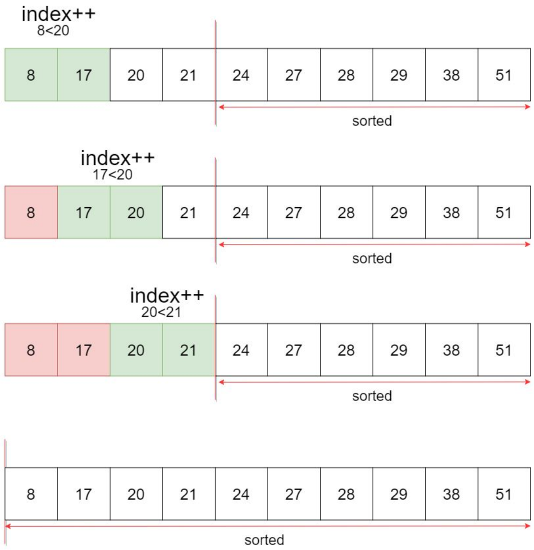
Step 5



Step 6

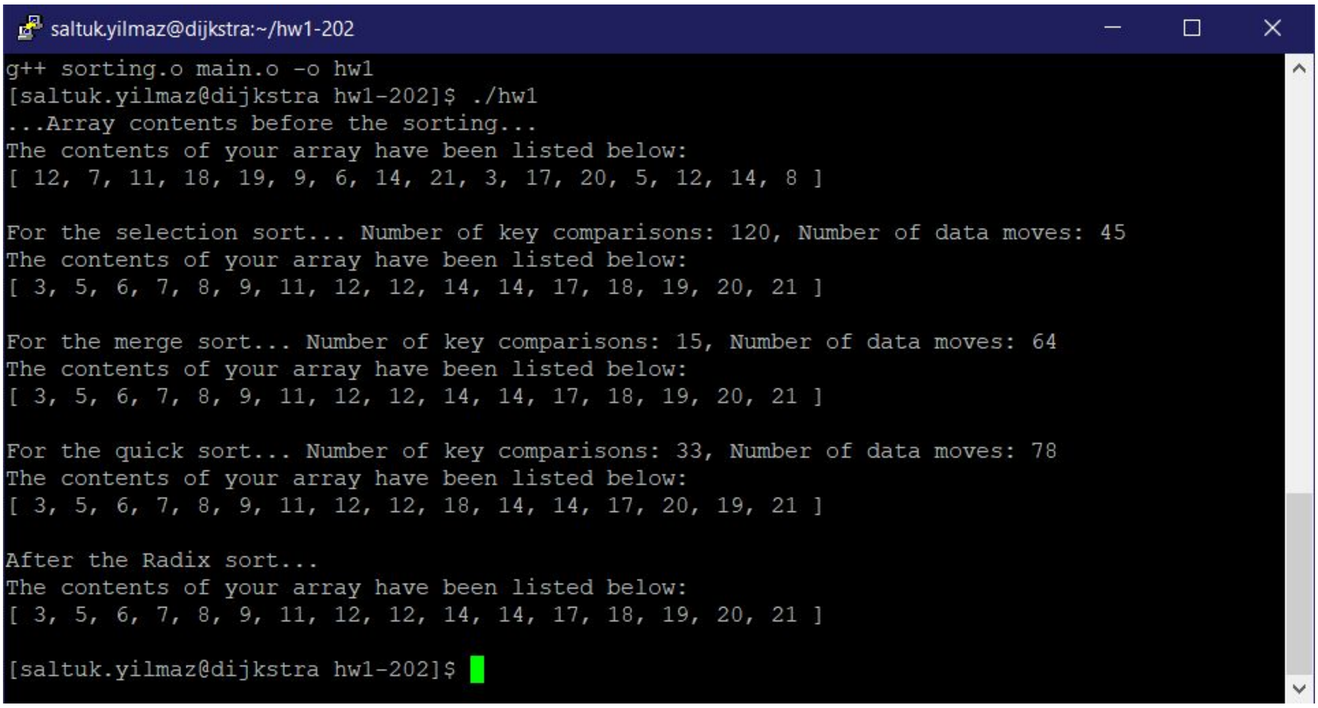


Step 7



Question-2

Console Screenshot for Executable Run:



Output of performanceAnalysis Function:

```
-----------------------------------------------------------------
---------------------- For Random Arrays ----------------------
-----------------------------------------------------------------
Analysis of Selection Sort
Array Size          Elapsed Time              compCount              moveCount
6000                 47 ms                     17997000                 17997
10000               130 ms                     49995000                 29997
14000               245 ms                     97993000                 41997
18000               398 ms                    161991000                 53997
22000               591 ms                    241989000                 65997
26000               842 ms                    337987000                 77997
30000              1104 ms                    449985000                 89997
-----------------------------------------------------------------
Analysis of Merge Sort
Array Size          Elapsed Time              compCount              moveCount
6000                2 ms                           5999                 75808
10000               2 ms                           9999                133616
14000               3 ms                          13999                193616
18000               4 ms                          17999                255232
22000               6 ms                          21999                319232
26000               7 ms                          25999                383232
30000               9 ms                          29999                447232
-----------------------------------------------------------------
Analysis of Quick Sort
Array Size          Elapsed Time              compCount              moveCount
6000                1 ms                          80006                137280
10000               1 ms                         141292                220956
14000               3 ms                         206828                327126
18000               3 ms                         287029                509625
22000               4 ms                         348736                616038
26000               4 ms                         425277                733065
30000               5 ms                         490029                740703
-----------------------------------------------------------------
Analysis of Radix Sort
Array Size          Elapsed Time
6000                2 ms
10000               4 ms
14000               5 ms
18000               6 ms
22000               8 ms
26000               9 ms
30000              11 ms
-----------------------------------------------------------------
```

```
------------------------------------------------------------------
--------------------- For Ascending Arrays ---------------------
------------------------------------------------------------------
Analysis of Selection Sort
Array Size          Elapsed Time           compCount           moveCount
6000                 65 ms                  17997000               17997
10000               152 ms                  49995000               29997
14000               291 ms                  97993000               41997
18000               479 ms                 161991000               53997
22000               718 ms                 241989000               65997
26000               990 ms                 337987000               77997
30000              1329 ms                 449985000               89997
------------------------------------------------------------------
Analysis of Merge Sort
Array Size          Elapsed Time           compCount           moveCount
6000                  0 ms                      5999               75808
10000                 1 ms                      9999              133616
14000                 2 ms                     13999              193616
18000                 4 ms                     17999              255232
22000                 6 ms                     21999              319232
26000                 7 ms                     25999              383232
30000                 9 ms                     29999              447232
------------------------------------------------------------------
Analysis of Quick Sort
Array Size          Elapsed Time           compCount           moveCount
6000                 33 ms                   9000000                9000
10000                74 ms                  25000000               15000
14000               131 ms                  49000000               21000
18000               215 ms                  81000000               27000
22000               322 ms                 121000000               33000
26000               461 ms                 169000000               39000
30000               591 ms                 225000000               45000
------------------------------------------------------------------
Analysis of Radix Sort
Array Size          Elapsed Time
6000                  1 ms
10000                 2 ms
14000                 4 ms
18000                 5 ms
22000                 6 ms
26000                 8 ms
30000                10 ms
------------------------------------------------------------------
```

-------------------- For Descending Arrays --------------------
----

Analysis of Selection Sort

Array Size	Elapsed Time	compCount	moveCount
6000	57 ms	17997000	17997
10000	137 ms	49995000	29997
14000	259 ms	97993000	41997
18000	421 ms	161991000	53997
22000	631 ms	241989000	65997
26000	890 ms	337987000	77997
30000	1207 ms	449985000	89997

Analysis of Merge Sort

Array Size	Elapsed Time	compCount	moveCount
6000	1 ms	5999	75808
10000	1 ms	9999	133616
14000	2 ms	13999	193616
18000	4 ms	17999	255232
22000	4 ms	21999	319232
26000	6 ms	25999	383232
30000	7 ms	29999	447232

Analysis of Quick Sort

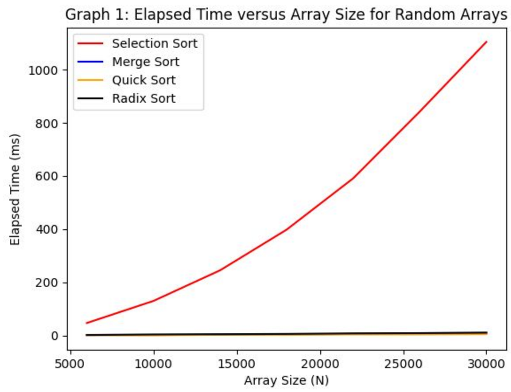
Array Size	Elapsed Time	compCount	moveCount
6000	69 ms	12000000	18015000
10000	172 ms	33333333	50025000
14000	330 ms	65333333	98035002
18000	543 ms	108000000	162045000
22000	753 ms	161333333	242055000
26000	1157 ms	225333333	338065002
30000	1540 ms	300000000	450075000

Analysis of Radix Sort

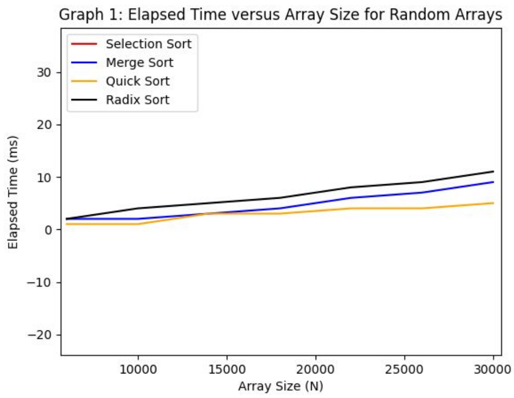
Array Size	Elapsed Time
6000	1 ms
10000	4 ms
14000	4 ms
18000	5 ms
22000	7 ms
26000	8 ms
30000	10 ms

Process returned 0 (0x0)   execution time : 18.017 s
Press any key to continue.

Question-3



(Graph 1: Elapsed Time versus Array Size for Random Arrays)



(Close up version of Graph 1 since Merge, Quick and Radix Sort Algorithms cannot be differentiated from each other in bigger scale)

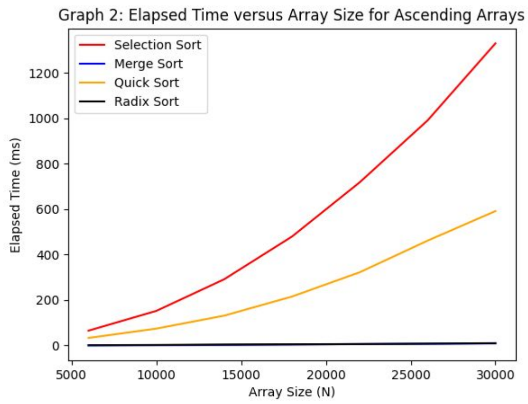
In theory:

- Selection Sort Algorithm has $O\left(n^2\right)$ time complexity for the worst, average and best cases.
- Merge Sort Algorithm has $O\left(n \log n\right)$ time complexity for the worst, average and best cases.
- Quick Sort Algorithm has $O\left(n^2\right)$ time complexity for the worst case and has $O\left(n \log n\right)$ for the best and average cases.
- Finally, Radix Sort Algorithm has $O\left(n\right)$ time complexity for the worst, average and best cases.

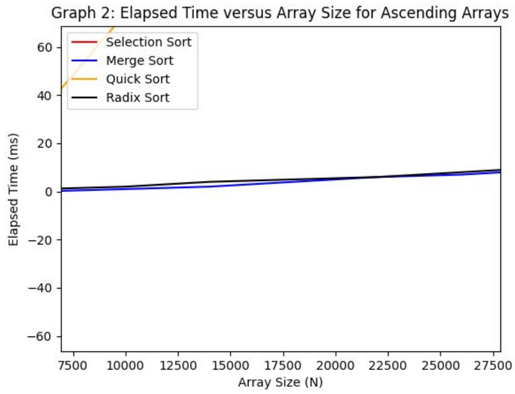
The run with the random arrays corresponds to the average case scenario for these four algorithms since arrays are randomly generated and it is not clear how many key comparisons and data moves are needed before the run.

If we examine the graph we can see that Selection Sort line is behaving as a parabola which is n^2. The empirical result is coherent with the theoretical result for the selection sort for the first case. For the other three algorithms we can say that Quick and Merge Sort algorithms behave similarly and their lines are intersecting at some points. However, in theory the elapsed time for Radix Sort should have been lesser than the Quick and Merge Sort algorithms. This difference might be about my computer's hardware and my operating system. It might also be about the array sizes I have used, for greater array sizes the results will obey the theory since for some inputs nlogn is lesser than n.

If I plot the Graph 1 again for ascending and descending arrays I obtain:



(Graph 2: Elapsed Time versus Array Size for Ascending Arrays)

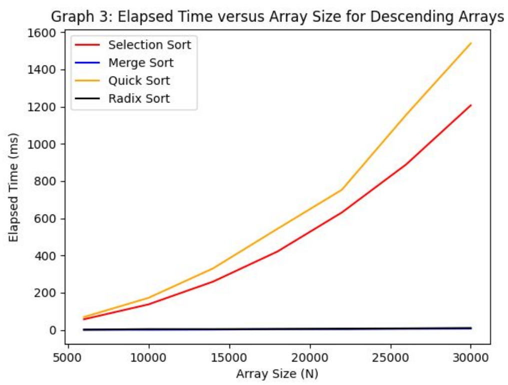


(Close up version of Graph 2 since Merge and Radix Sort Algorithms cannot be differentiated from each other in bigger scale)

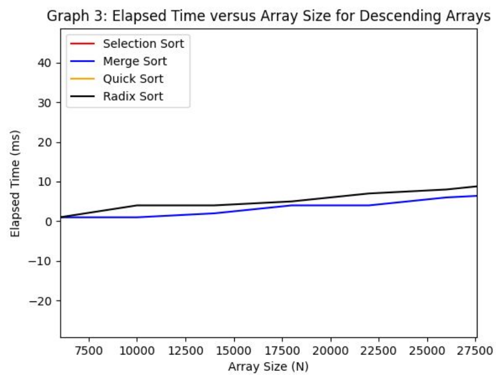
Selection sort behaves similarly to the first run with random arrays and fits the $O(n^2)$. However, this time quick sort is more similar to $O(n^2)$. Already sorted arrays and choosing the first element as pivot mean the worst case for the quick sort algorithm since the key comparisons take $\frac{n^2}{2} - \frac{n}{2}$ in other words $O(n^2)$ and data moves take $\frac{n^2}{2} + \frac{n}{2} - 1$ (also $O(n^2)$). Also Radix and Merge Sort algorithms are similar to the first run.

If we examine Graph 2, Selection sort is nearly the same with Graph 1. Quick sort is similar to n^2 but its growth rate is lesser than the growth rate of Selection sort. For greater array sizes, it may approximate the selection sort more. Moreover, it can be said that Radix sort and Merge sort intersect at more than one point and at some points Merge sort is greater than Radix sort as the theory suggested for greater array sizes it will exceed the Radix sort clearly.

For descending Arrays:



(Graph 3: Elapsed Time versus Array Size for Descending Arrays)



(Close up version of Graph 3 since Merge and Radix Sort Algorithms cannot be differentiated from each other in bigger scale)

In this run, all algorithms except Quick sort behaves similarly to other outputs. For descending arrays the Quick sort definitely behaves with $O\left(n^2\right)$ time complexity and it has a greater growth rate than the Selection sort. It can be said that this is another worst case for the Quick sort.