CS 202

Homework Assignment 1

Section 2

Instructor: Çiğdem Gündüz Demir

Ahmet Işık

a.isik@ug.bilkent.edu.tr

21702226

Q1:

a-) By the Big-Oh definition, T(n) is $O(n^4)$ if $T(n) \leq c \cdot n^4$ for some $n \geq n_0$. So, let us check the following condition: if $20n^4 + 20n^2 + 5 \leq c \cdot n^5$ then $\frac{20}{n} + \frac{20}{n^3} + \frac{5}{n^5} \leq c$. Therefore, the Big-Oh condition holds for $n \geq n_0 = 1$ and $c \geq 45$ (= 20 + 20 + 5). Larger values of n_0 creates smaller factors c but in any case, the above statement is valid.

b-)

For Selection Sort:

blue colour means sorted,

red colour means selected.

Initial array:

18	4	47	24	15	24	17	11	31	23

After 1st swap:

18	4	24	15	24	17	11	31	23	47

After 2nd swap:

18	4	24	15	24	17	11	23	31	47

After 3rd swap:

18	4	15	24	17	11	23	24	31	47

After 4th swap:

18	4	15	17	11	23	24	24	31	47

After 5th swap:

18	4	15	17	11	23	24	24	31	47

After 6th swap:

4	15	17	11	18	23	24	24	31	47

After 7th swap:

4	15	11	17	18	23	24	24	31	47

After 8th swap:

4	11	15	17	18	23	24	24	31	47

After 9th swap:

4	11	15	17	18	23	24	24	31	47

After 10th swap:

4	11	15	17	18	23	24	24	31	47

For Bubble Sort:

Red colour is comparison,

Thick numbers are sorted.

Pass (1)

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

Pass (2)

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

Pass (3)

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

Pass (4)

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

Q2:

a-) Codes are added to file.

b-) Screenshot of the console to the solution of Question 2-b:

```
-bash-4.2$ g++ main.cpp sorting.cpp auxArrayFunctions.cpp -o hw1
^[[A-bash-4./hw1
Insertion Sort
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Number of key comparisons: 74
Number of data moves: 89
Merge Sort
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Number of key comparisons: 46
Number of data moves: 91
Quick Sort
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Number of key comparisons: 47
Number of data moves: 105
-bash-4.2$
```

c-) Screenshot of the console to the solution of Question 2-c:

Performance Analysis for Randomly Ordered Array:

```
C:\Users\Ahmet\Desktop\CS201\Cs202_hw1\bin\Debug\Cs202_hw1.exe
Part c - Time Analysis of Sorting Algorithms with an 5000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   21 ms           6199416       6204415
Merge Sort       1 ms             55197          70194
Quick Sort       1 ms             78546          124263
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 10000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   89 ms           24719862        24729861
Merge Sort       2 ms            120292          150289
Quick Sort       2 ms            156345          234474
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 15000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   192 ms           56424038       56439037
Merge Sort       3 ms            189326          234323
Quick Sort       3 ms            280501          450519
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 20000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   334 ms           99836601       99856600
Merge Sort       5 ms            260843          320840
Quick Sort       3 ms            346292          640365
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 25000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   548 ms           155738881       155763880
Merge Sort       6 ms             334177         409174
Quick Sort       4 ms             458884         728082
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 30000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   725 ms           224296533       224326532
Merge Sort       8 ms             408650         498647
Quick Sort       5 ms             545609         854259
```

Performance Analysis for Already Ordered Array:

```
C:\Users\Ahmet\Desktop\CS201\Cs202_hw1\bin\Debug\Cs202_hw1.exe
Part c - Time Analysis of Sorting Algorithms with an 5000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   0 ms             4999           9998
Merge Sort       0 ms             32004          47001
Quick Sort       28 ms           12497500       14997
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 10000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   0 ms             9999           19998
Merge Sort       1 ms             69008          99005
Quick Sort       105 ms           49995000       29997
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 15000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   0 ms             14999           29998
Merge Sort       2 ms             106364          151361
Quick Sort       235 ms           112492500       44997
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 20000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   0 ms             19999           39998
Merge Sort       4 ms             148016          208013
Quick Sort       450 ms           199990000       59997
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 25000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   0 ms             24999           49998
Merge Sort       5 ms             188476          263473
Quick Sort       673 ms           312487500       74997
-----------------------------------------------------
Part c - Time Analysis of Sorting Algorithms with an 30000 Size Array
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   0 ms             29999           59998
Merge Sort       6 ms             227728          317725
Quick Sort       936 ms           449985000       89997
```

d-)

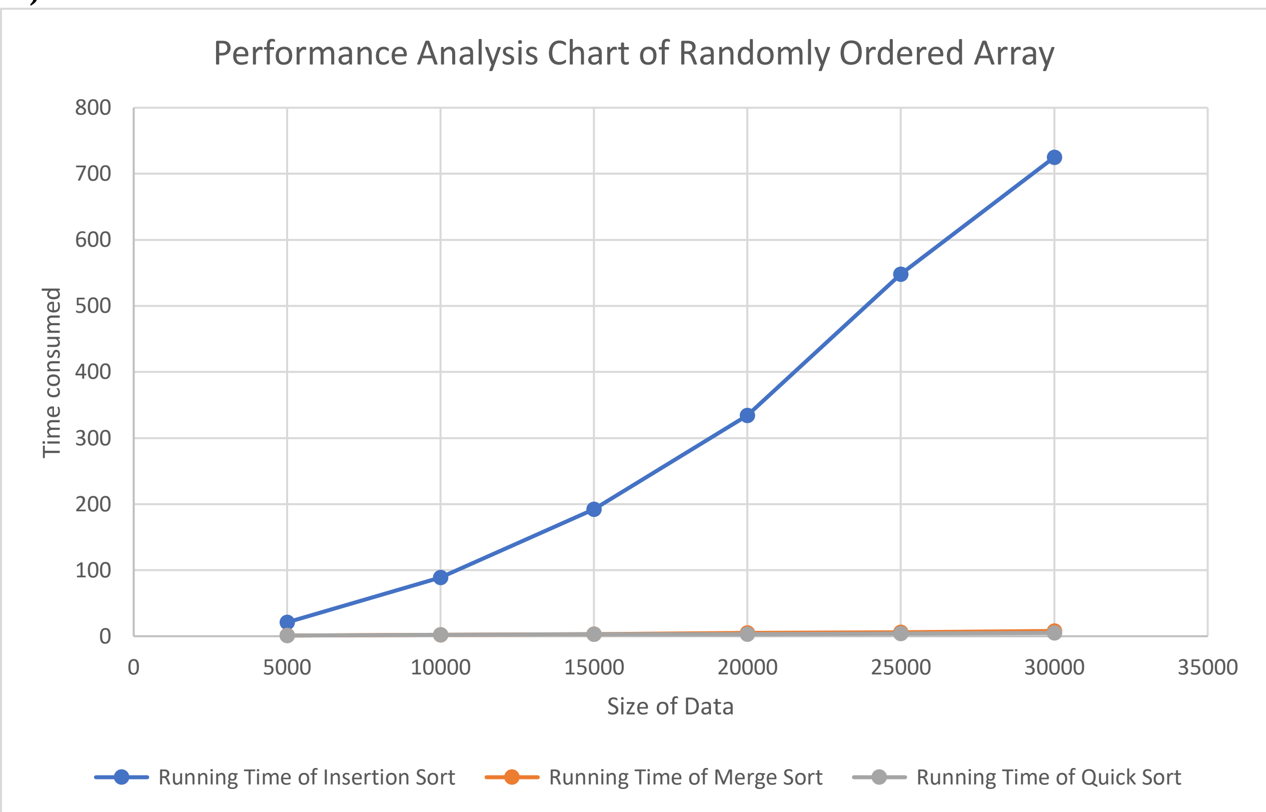

Chart (1)

After implementing three different sorting algorithms which are insertion sort, merge sort and quick sort respectively on both randomly ordered and ascending ordered arrays, timing process of sorting is analysed on a chart with respect to the size of arrays and will be compared with theoretical results of these sorting algorithms. In theoretical results, Insertion Sort has $O(n^2)$ running time complexity for both average and worst cases. Merge Sort has O(n*log n) running time complexity for both average and worst cases. Meanwhile, Quick Sort has $O(n^2)$ running time complexity for worst case while O(n* log n) for average case.

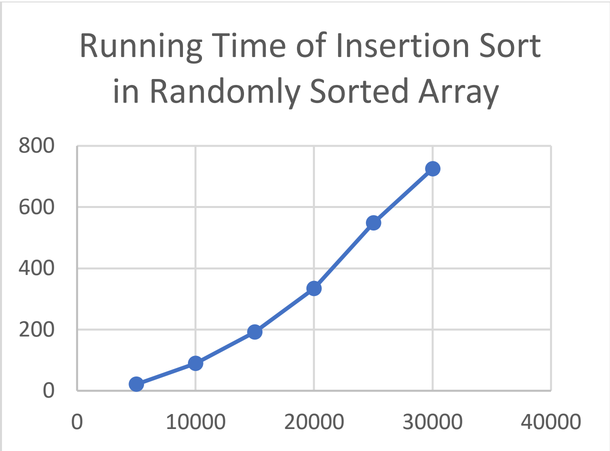

Chart (2)

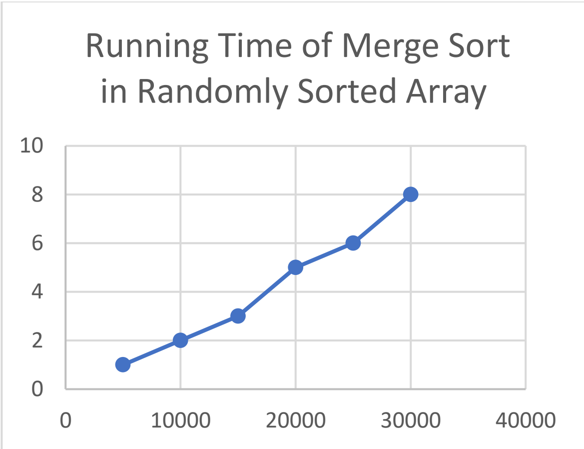

Chart (3)

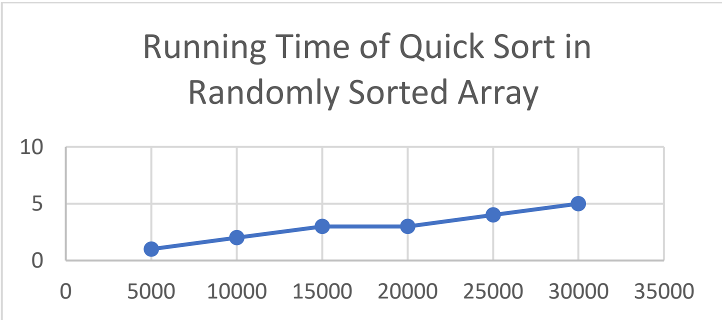

Chart (4)

In randomly ordered array, consumed time of insertion sort increased with the increasing of data size simultaneously. While size of data is increasing twice, consumed time is increasing 4 times. This ratio is almost similar with the theoretical result which is $O(n^2)$. However, if we examine the quick sort and merge sort, we can infer that changing in data size does not affect the consumed time, it takes too short time nearing almost zero. If we examine the ratio of time with respect to the data size from the chart (3) and (4) above, we can easily observe that its time complexity is almost same with theoretical one which is O(n*log n). Additionally, we can infer that by using quick sort algorithm, an array with big data size can be sorted in less time than merge sort. Randomly sorted array creates average case for these three sorting algorithms.

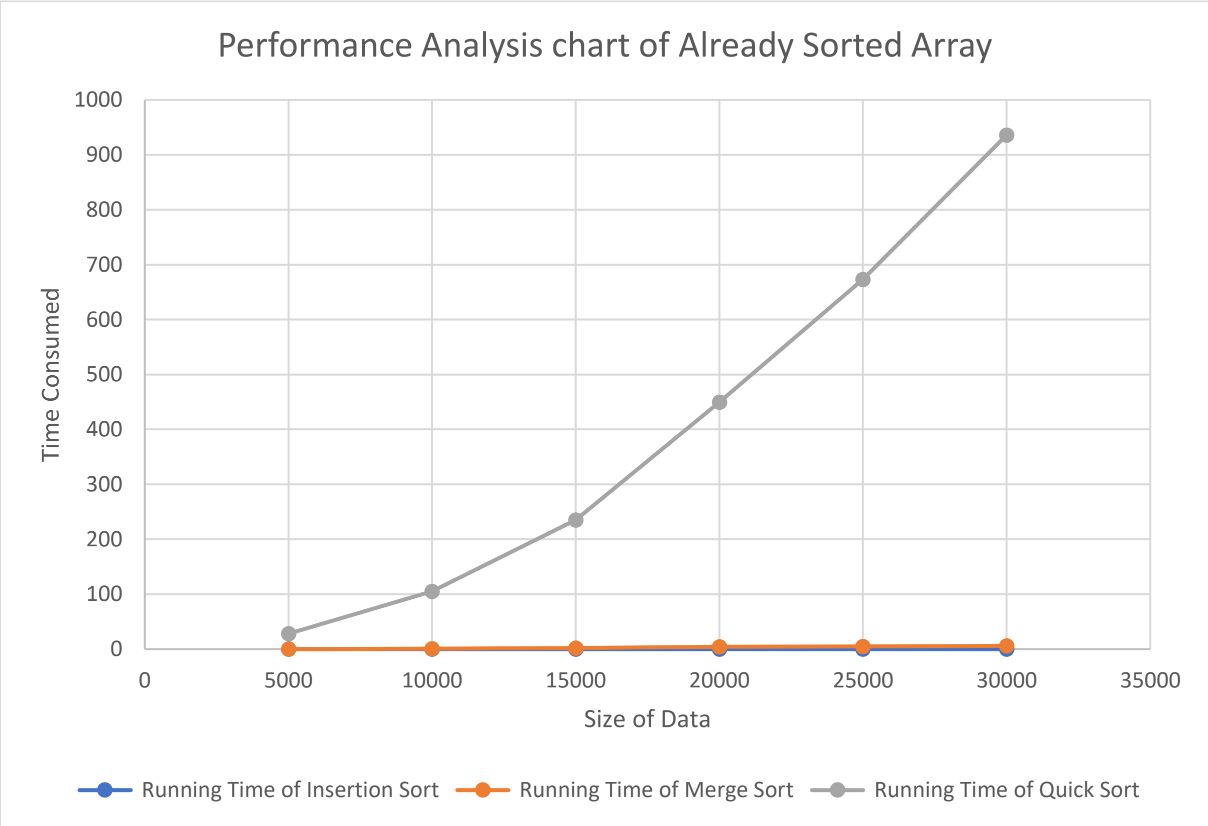

Chart (5)

Examining the time for randomly sorted array, consumed time data while sorting is collected for already sorted arrays with different data sizes and results will be interpreted and compared with theoretical ones. By observing the chart (5), we can easily see that sorting an already sorted array by using quick sort is useless because it takes much more time while data is increasing. Hence, we can infer that an already sorted array constitutes worst case and running time complexity is similar with the theoretical one which is $O(n^2)$ for worst case.

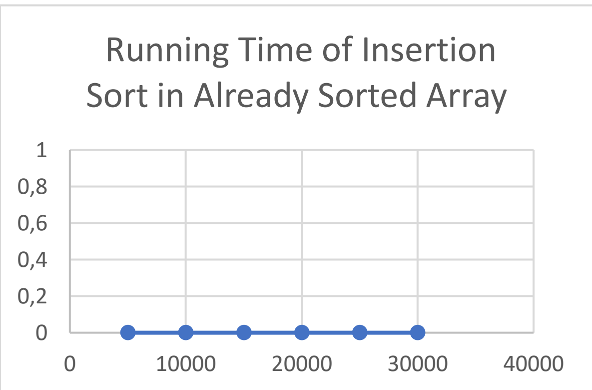

Chart (6)

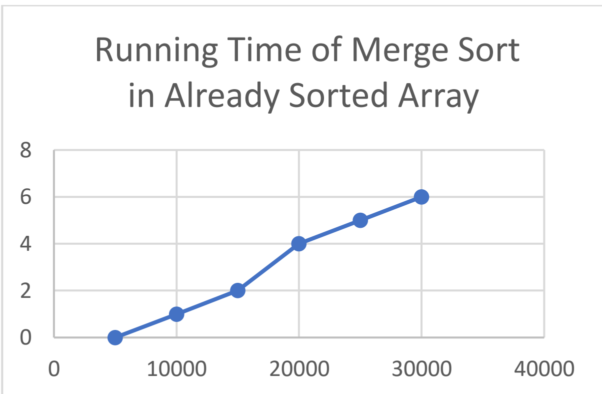

Chart (7)

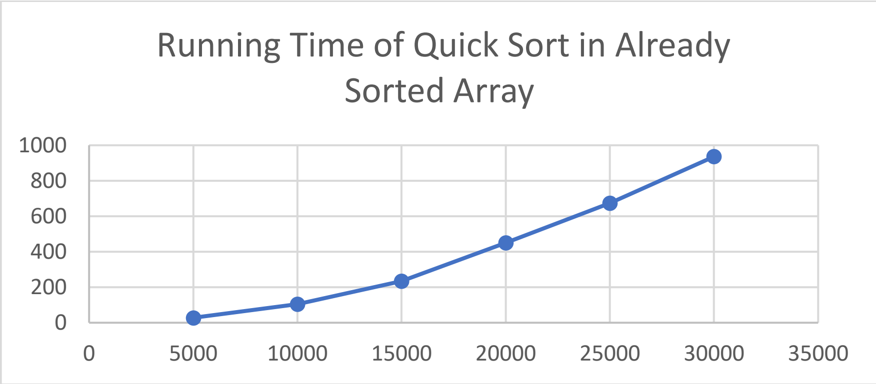

Chart (8)

Although quick sort was the best way of sorting for a randomly sorted array, in that case it works like insertion sort for randomly sorted arrays if we compare the chart (2) and chart (8). On the other hand, insertion sort took zero millisecond for each size of already sorted arrays in chart (6). So, we can say that insertion sort algorithm is the best way for sorting an already sorted array. In merge sort, almost nothing has changed in duration because it might do same operations for both randomly sorted and already sorted arrays. This situation is consistent the running time complexity of merge sort for both worst and average case which are same, O(n*log n).

In summarize, while sorting a randomly sorted array constitutes average case for insertion, merge and quick sort algorithms, in already sorted array it constitutes again average case for merge sort, worst case for quick sort and best case for insertion sort. When we compare the experimental results of increasing rate of time with respect to the data size, we can see that they are almost same with theoretical running time complexity for these sorting algorithms.

Q3:

In this part, insertion, merge and quick sort algorithms are implemented three identical nearly sorted arrays. In nearly sorted arrays, each item in these arrays is at most K away from its target location and that K is arranged from 800 to 28800 and from 5 to 180 in an array with 30000 size and the timing results of sorting algorithms are compared. If the all items in array are at most 800 far from the target location, that means there are much more same item that does not require comparison and sorting than an array that their elements are at most 28800 far from the target location.

In that case, it is expected that insertion sort is going to take more time when the distance between items increases because the nearly sorted arrays becomes near to randomly sorted arrays from already sorted arrays while the distance between items is increasing. Whenever the K decreases, the array becomes more similar with already sorted arrays. Whenever the K increases, the array becomes more similar with randomly sorted arrays. We can prove that idea with the experimental results as follows:

In table (1), items are too close to each other in first condition (max distance is 5) and quick sort algorithm consumes time like it consumed in already sorted arrays which is worst case for quick sort, $O(n^2)$. With the rise of the distance, especially in the table (2), quick sort algorithm takes smaller time to sort the array which is similar with quick sort in randomly sorted arrays in question 2-d. Quick sort algorithm has average case for that situation which it does in randomly sorted arrays, O(n*log n).

On the other hand, insertion sort even does not consume time in the experiments in table (1) like it does in randomly sorted arrays which is best case for insertion sort. Meanwhile, consumed time in insertion sort is increasing with the rise of the distance between items especially in table (2). These results are similar with the insertion sort for randomly sorted arrays which means with the increasing the size of data it becomes the worst case, $O(n^2)$.

Finally, in merge sort algorithm, we can say that almost nothing has changed with the change of distance between items in array as we observed in both already sorted and randomly sorted arrays.

In brief, if K is a too big value (the distance between items is too far), because it will be similar with randomly sorted arrays, it would be more efficient to use quick sort algorithm and merge sort algorithm. If K is a small value (the distance between items is too close), because it will be similar with already sorted arrays, it would be more efficient to use insertion sort algorithm.

```
C:\Users\Ahmet\Desktop\CS201\Cs202_hw1\bin\Debug\Cs202_hw1.exe
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 5 away from target
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   1 ms           77432       107431
Merge Sort       7 ms             246084         336081
Quick Sort       366 ms             158669615         152868
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 20 away from target
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   1 ms           227091       257090
Merge Sort       7 ms             270417         360414
Quick Sort       129 ms             51424775         271236
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 45 away from target
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   4 ms           472878       502877
Merge Sort       15 ms             287312         377309
Quick Sort       124 ms             25274528         359790
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 80 away from target
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   7 ms           825530       855529
Merge Sort       16 ms             299724         389721
Quick Sort       71 ms             14010850         442038
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 125 away from target
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   11 ms           1276768       1306767
Merge Sort       16 ms             309429         399426
Quick Sort       48 ms             9249081         522180
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 180 away from target
Sorting Type     Time Elapsed     compCount      moveCount
Insertion Sort   9 ms           1821998       1851997
Merge Sort       11 ms             317536         407533
Quick Sort       30 ms             5956041         595761
```

Table (1)

C:\Users\Ahmet\Desktop\CS201\Cs202_hw1\bin\Debug\Cs202_hw1.exe

```
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 800 away from target
Sorting Type      Time Elapsed      compCount       moveCount
Insertion Sort    29 ms            7952748        7982747
Merge Sort        9 ms              350536          440533
Quick Sort        8 ms              1943317          947016
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 3200 away from target
Sorting Type      Time Elapsed      compCount       moveCount
Insertion Sort    155 ms            31377859        31407858
Merge Sort        16 ms              379161          469158
Quick Sort        13 ms              954582          1526076
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 7200 away from target
Sorting Type      Time Elapsed      compCount       moveCount
Insertion Sort    261 ms            68818976        68848975
Merge Sort        10 ms              394568          484565
Quick Sort        9 ms              1017332          2325879
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 12800 away from target
Sorting Type      Time Elapsed      compCount       moveCount
Insertion Sort    396 ms            116718878        116748877
Merge Sort        8 ms              403791          493788
Quick Sort        10 ms              1128185          2754264
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 20000 away from target
Sorting Type      Time Elapsed      compCount       moveCount
Insertion Sort    1072 ms            179134300        179164299
Merge Sort        9 ms              407262          497259
Quick Sort        6 ms              715170          1398795
-----------------------------------------------------
Question 3 - Time Analysis of Sorting Algorithms  with 30000 size of Array that items are at most 28800 away from target
Sorting Type      Time Elapsed      compCount       moveCount
Insertion Sort    1057 ms            223296773        223326772
Merge Sort        10 ms              408592          498589
Quick Sort        5 ms              523083          905007
```

Table (2)