26.04.2180

CS202 HW01

Section 2

Irmak Akyeli

21803690

Question 1

a. To conclude that f(n) = 20n^4+ 20n^2 + 5 is O(n^5), we need to prove that there is c and n0 for which

20n^4+ 20n^2 + 5 < c*n^5 when n > n0. Taking c = 50 and n0 = 1, we can see that 20n^4+ 20n^2 + 5 < 50*n^5 when n > 1. Taking n=2 we obtain

20*16+20*4+5 = 405 <1600 = 50*32 This means that using mathematical induction, it can be concluded that is true for every n > 1. So, as big O notation shows an upper bound and we showed that the function is smaller than O(n^5), the function is O(n^5).

b. 1) Selection Sort: (/ denotes the beggining of sorted list)

İnitial Algorithm: 18 , 4 , 47 , 24 , 15 , 24 , 17 , 11 , 31 , 23 /
Stage 1 : 18 , 4 , 23, 24 , 15 , 24 , 17 , 11 , 31 / 47
Stage 2 : 18 , 4 , 23 , 24 , 15 , 24 , 17 , 11 / 31 , 23
Stage 3 : 18 , 4 , 23 , 11 , 15 , 24 , 17 / 24 , 31 , 23
Stage 4 : 18 , 4 , 23 , 11 , 15 , 17 / 24 , 24 , 31 , 23
Stage 5 : 18 , 4 , 17 , 11 , 15 / 23 , 24 , 24 , 31 , 23
Stage 6 : 15 , 4 , 17 , 11 / 18 , 23 , 24 , 24 , 31 , 23
Stage 7 : 15 , 4 , 11 / 17 , 18 , 23 , 24 , 24 , 31 , 23
Stage 8 : 11 , 4 / 15 , 17 , 18 , 23 , 24 , 24 , 31 , 23
Stage 9 : 4 / 11 , 15 , 17 , 18 , 23 , 24 , 24 , 31 , 23

Selection sort creates two arrays on that is original and the second who is unsorted located at the end of the first array. It makes passes through the first array and in each pass it selects the bigger element then swaps it the last element of the array and adds it to the sorted array list. It makes n-1 passes to create the sorted array.

2) Bubble Sort (/ denotes the beggining of sorted list)

İnitial Algorithm: 18 , 4 , 47 , 24 , 15 , 24 , 17 , 11 , 31 , 23 /
Stage 1 : 4 , 18 , 24, 15 , 24 , 17 , 11 , 31 , 23 / 47
Stage 2 : 4 , 18 , 15, 24 , 17 , 11 , 24 , 23 / 31 , 47
Stage 3 : 4 , 15 , 18, 17 , 11 , 24 , 23 / 24 , 31 , 47

Stage 4 : 4 , 15 , 17, 11 , 18 , 23 / 24 , 24 , 31 , 47

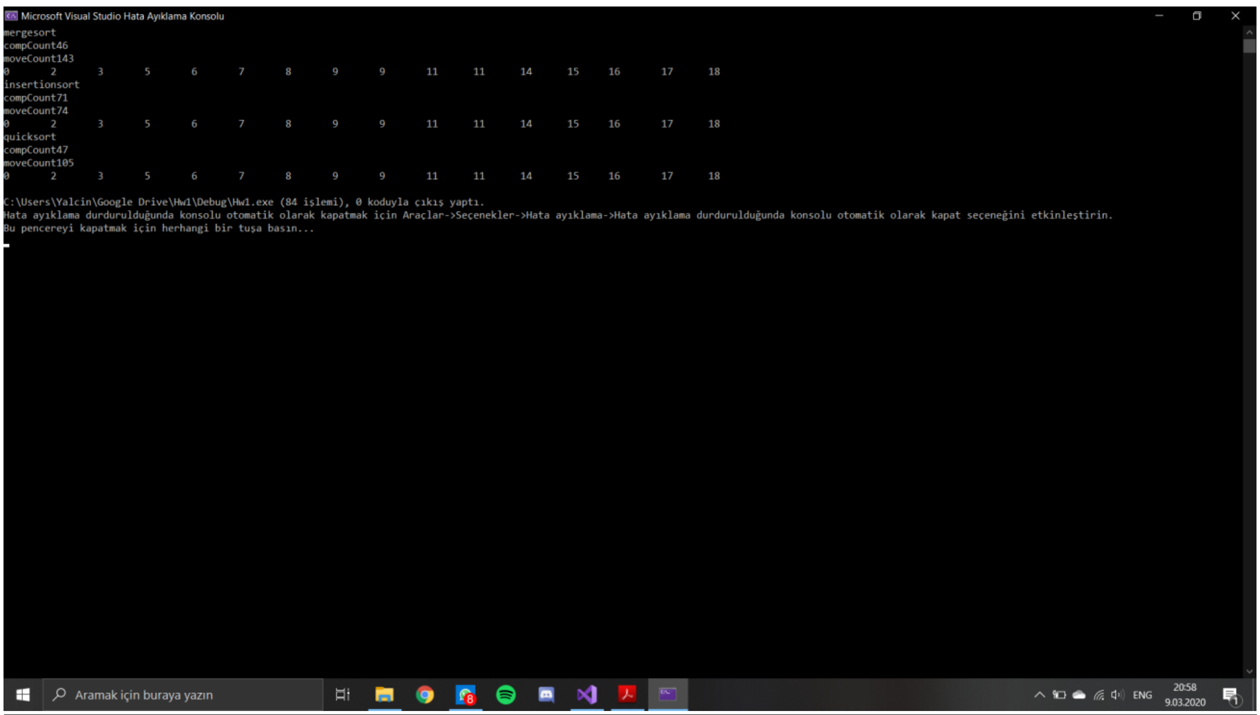
Stage 5 : 4 , 15 , 11, 17 , 18 / 23 , 24 , 24 , 31 , 47

Stage 6 : 4 , 11 , 15, 17 / 18 , 23 , 24 , 24 , 31 , 47

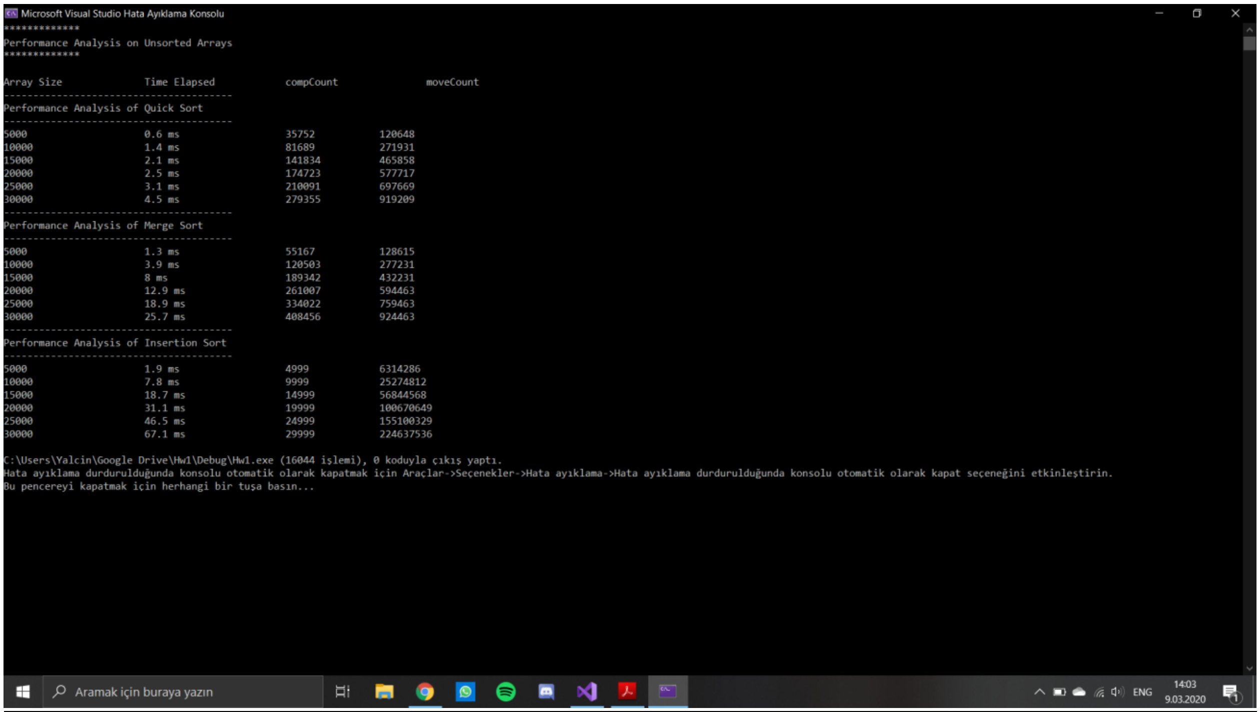
There is actually one more step where bubble sort traces all the array and check if it is sorted or not and it is done. In each step it takes two integer and compares them and each time it moves the bigger to the end where it enters the sorted list and as it compares others on the way it is faster than selection sort.

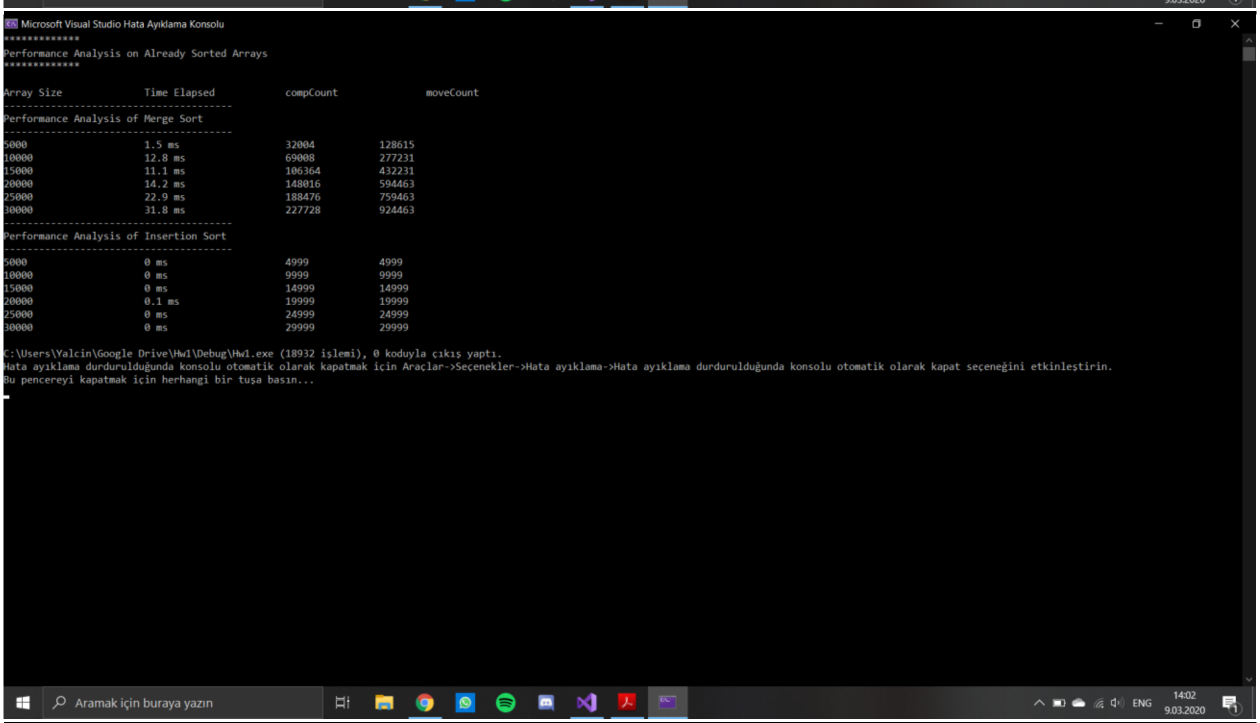
Question 2

b) Screenshot of part b.



c) Screenshots of part c.





d) Analysis of the results of the question2:

Looking at the data that we obtained from experimental results, we can compare the theorical and experimental results and the sorting algorithms in themselves.

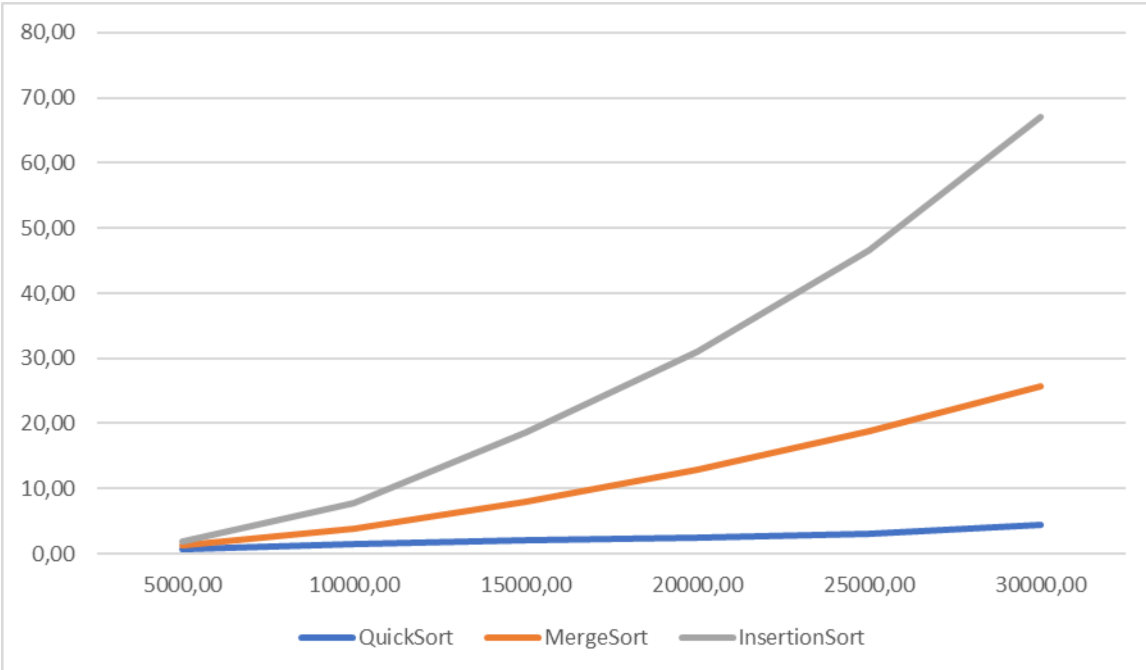
- The most obvious result is that Insertion sort is much slower than the other comparison types in random arrays. Also, its increasement rate with array size is much higher than the others algorithms. It can be clearly seen that insertion sort is more polynomial, than the mergeSort and quickSort which seems like more linear.
- We can also observe that as the array size gets smaller the difference between the algorithms also gets smaller which means that in the smaller array sizes their efficiencies are closer rather than the big array sizes where the main difference starts.
- We can also conclude that the efficiencies of quicksort and mergeSort are closer to one to another than the insertion sort and that the gap between them starts to grow relatively later than insertion sort.

Looking at the experimental results we can say that the more efficient algorithm is the quicksort then mergesort and lastly insertionsort. This is resuşt shows the same as theorical order and therefore could be considered relatively correct. To compare the experimental and theorical results even further we can make the following points.

- We know that the complexity of insertion sort is O(n^2), which should have given a polynomial line and the results we obtained from this experiment and the table of our results satisfies this theorical expectation.
- We know that the complexity of the remaining two algorithms are O(nlogn) which results in a more linear graph as we also obtained. This also demonstrates the reason why their graphs are more similar than the insertionsort's and why they increase in a slower way as the array gets bigger.

Therefore, looking at all this we can conclude that even with all the error margins that we obtained in the experimental results we can say that our experiment overall satisfies the theorical expectations and fall in the margin of reason. They can also be accepted as consistent between themselves and others in random arrays.

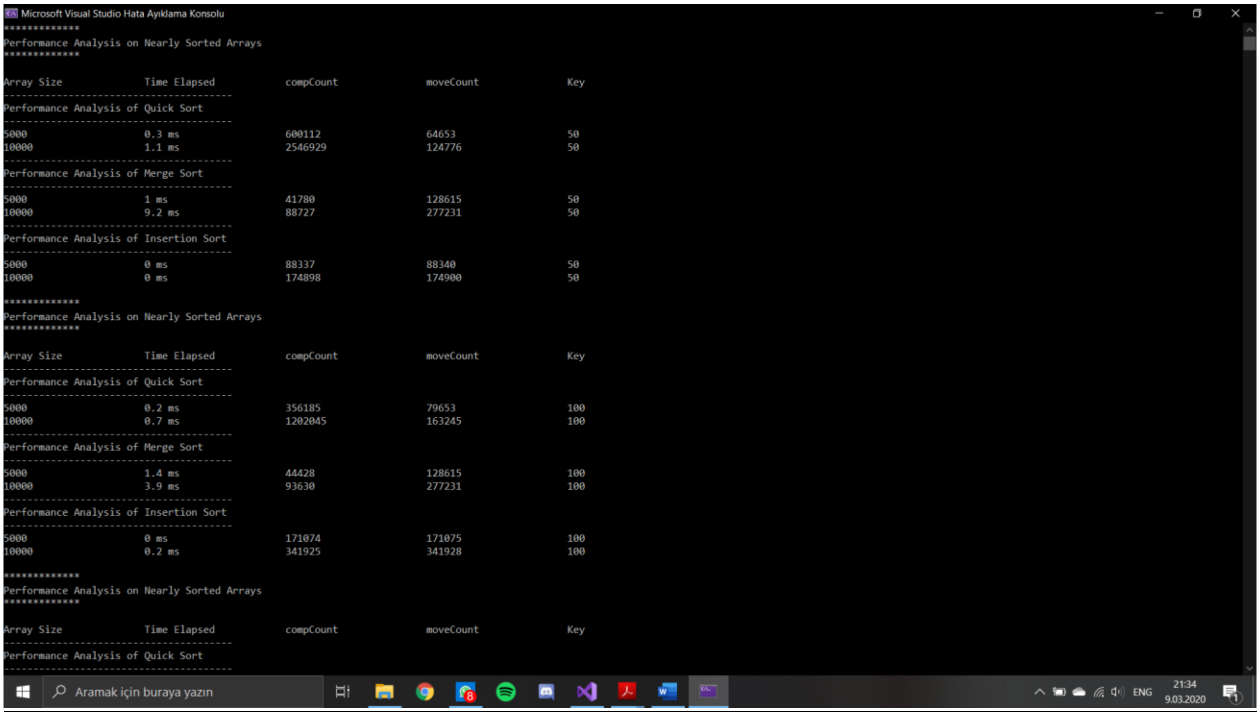
During already sorted arrays however we see that insertion sort is much faster that almost always gives 0ms as it only goes ones on the array and do nothing more like the recursive functions merge and quick sort. This gives him a real advantage as it does not perform any unnecessary operations like merge sort on an already sorted array. However, as the possibility of obtaining an already sorted array to use in this sorting algorithms we see that using mergesort and quicksort is still more logical.

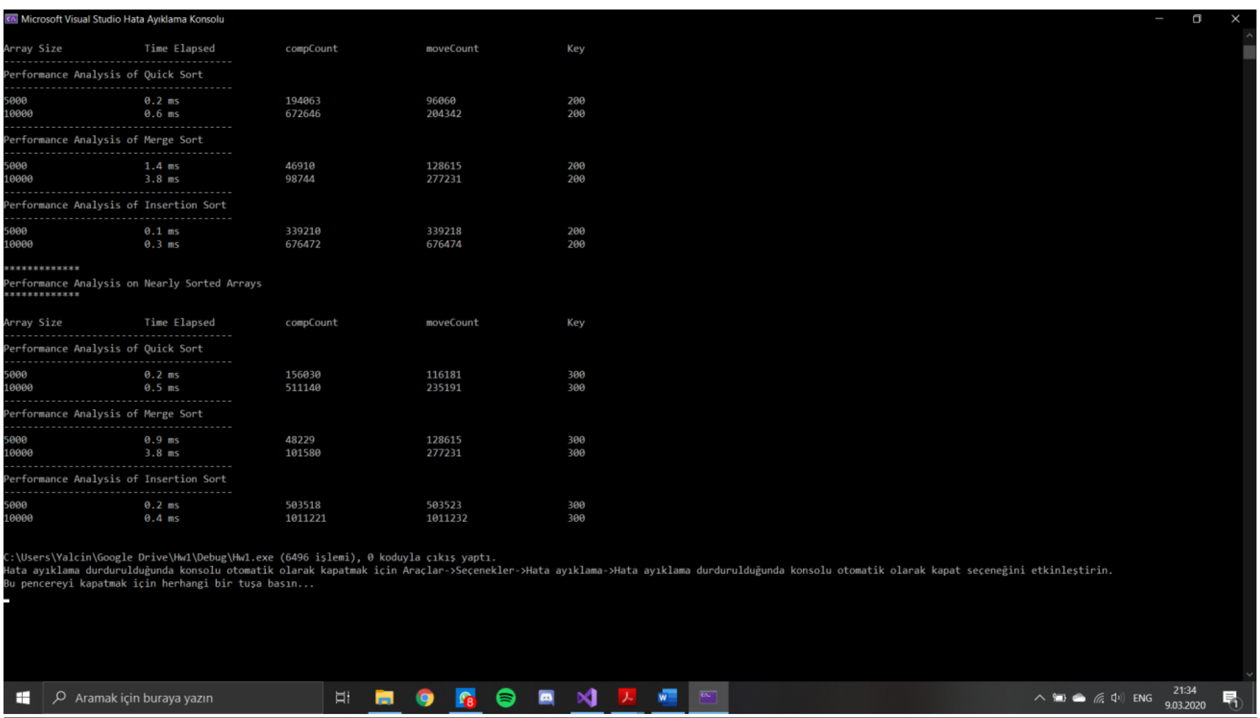


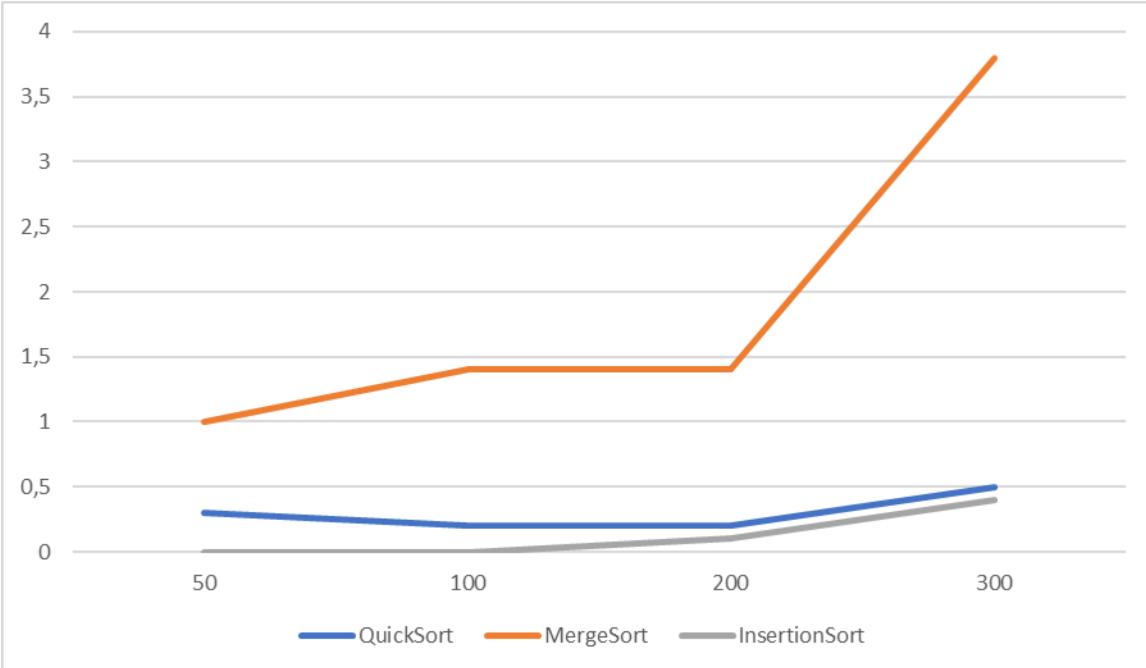
Graph: Time versus array size

<u>Question 3</u>

Here are the screenshots for question 3, with K = 50, 100, 200, 300.


Microsoft Visual Studio Hata Ayıklama Konsolu
Performance Analysis on Nearly Sorted Arrays
Array Size Time Elapsed compCount moveCount Key
Performance Analysis of Quick Sort
5000 0.3 ms 600112 64653 50
10000 1.1 ms 2546929 124776 50
Performance Analysis of Merge Sort
5000 1 ms 41780 128615 50
10000 9.2 ms 88727 277231 50
Performance Analysis of Insertion Sort
5000 0 ms 88337 88340 50
10000 0 ms 174898 174900 50
Performance Analysis on Nearly Sorted Arrays
Array Size Time Elapsed compCount moveCount Key
Performance Analysis of Quick Sort
5000 0.2 ms 356185 79653 100
10000 0.7 ms 1202045 163245 100
Performance Analysis of Merge Sort
5000 1.4 ms 44428 128615 100
10000 3.9 ms 93630 277231 100
Performance Analysis of Insertion Sort
5000 0 ms 171074 171075 100
10000 0.2 ms 341925 341928 100
Performance Analysis on Nearly Sorted Arrays
Array Size Time Elapsed compCount moveCount Key
Performance Analysis of Quick Sort
Aramak için buraya yazın
ENG
21:34
9.03.2020


Microsoft Visual Studio Hata Ayıklama Konsolu
Array Size Time Elapsed compCount moveCount Key
Performance Analysis of Quick Sort
5000 0.2 ms 194063 96060 200
10000 0.6 ms 672646 204342 200
Performance Analysis of Merge Sort
5000 1.4 ms 46910 128615 200
10000 3.8 ms 98744 277231 200
Performance Analysis of Insertion Sort
5000 0.1 ms 339210 339218 200
10000 0.3 ms 676472 676474 200
Performance Analysis on Nearly Sorted Arrays
Array Size Time Elapsed compCount moveCount Key
Performance Analysis of Quick Sort
5000 0.2 ms 156030 116181 300
10000 0.5 ms 511140 235191 300
Performance Analysis of Merge Sort
5000 0.9 ms 48229 128615 300
10000 3.8 ms 101580 277231 300
Performance Analysis of Insertion Sort
5000 0.2 ms 503518 503523 300
10000 0.4 ms 1011221 1011232 300
C:\Users\Yalcin\Google Drive\Hw1\Debug\Hw1.exe (6496 işlemi), 0 koduyla çıkış yaptı.
Hata ayıklama durdurulduğunda konsolu otomatik olarak kapatmak için Araçlar->Seçenekler->Hata ayıklama->Hata ayıklama durdurulduğunda konsolu otomatik olarak kapat seçeneğini etkinleştirin.
Bu pencereyi kapatmak için herhangi bir tuşa basın...
Aramak için buraya yazın
ENG
21:34
9.03.2020



Graph: Time versus K

In this question