Özlem MALKOÇ
1901042700

HW7-REPORT

NOTE : I take commented line the map display functions in order to see the durations correctly, but all algorithms are working correctly.

PART A)

Bubble Sort :

Each element in the list is compared with the next element. If the value of the first element is greater than the value of the second element, the two elements are swapped.

Best Case -> If the array is already sorted, this is an example of the best case. In this case, the algorithm
will only compare the value next to the value. The algorithm will be completed after a single iteration, as it will not make any displacement. Since the number of comparisons is proportional to the size of the array, we can say that the time complexity is O(n).

Average Case -> If the array is randomly sorted, this is an example of an average case. It compares neighboring elements each time and replaces them when necessary. It makes n-1 comparisons each time. The average number of passes (in the case of the average case) is (n-1)*(n) because it is n/2. We can say /2)= O(n^2).

Worst Case -> If the array is in reverse order, this is an example of worst case. Each time it will do n-1 comparisons and n-1 displacements.(n-1)*(n-1)= O(n^2).

İnsertion Sort :

The Insertion sort algorithm scans the array from left to right, comparing each element with the elements to its left, shifting it to the right as needed.

Best Case -> If the array is already ordered, this is an example of a best case. N-1 comparison will be made and no substitution will be made. Therefore, the time complexity is O(n).

Average Case -> If the array is in random order, this is an example of an average case. n-1 comparison will be made. Since it is an average case, we can say that the average number of passes is n/2. for (n-1)*(n/2)= O(n^2).

Worst Case -> If the array is in reverse order, this is an example of a worst case. Each of them will do n-1 comparisons and n-1 displacements.(n-1)*(n-1)= O(n^2).

Selection Sort:

In an unordered list with the selection sort algorithm, the smallest element is found in each iteration. At the end of the iteration, the smallest element determined is replaced with the element at the beginning of the list. It is not included in the next iteration. In the second iteration, the smallest of the array elements is found again and this time it replaces the second element in the list and continues in this way until the list is sorted.

Best case -> If the array is already ordered, this is an example of the best case. In this case, the algorithm will only compare. The algorithm will be completed after a single iteration, as it will not make any displacement. Since the number of comparisons is proportional to the size of the array, we can say that the time complexity is O(n).

Average Case -> If the array is randomly sorted, this is an example of an average case. It compares the elements each time and replaces them when necessary. It makes n-1 comparisons each time. The average number of passes (in the case of the average case) is (n-1)*(n/) because it is n/2. We can say 2)= O(n^2).

Worst Case -> If the array is in reverse order, this is an example of a worst case. The number of comparisons will decrease by one at a time. (n-1) + (n-2) + (n-3) + ... + 1 = n*(n-1)/2 = (n^2 - n) / 2 = O(n^2).

Quick Sort:

It aims to sort by selecting a number (pivot) in the array to be sorted and classifying all other numbers as greater or less than this number.

Best Case -> If the number of elements of an array is n and each time the pivot element is selected in the middle of the array: At the end of the 1st step, n/2 elements remain. As the number of steps increases, each step is multiplied by ½. In total, log n steps are performed. At each step, n comparisons are made to split the subsequences. So we can say that the time complexity is O(n log n).

Average Case -> In this scenario, the pivot is chosen randomly. We can say that it has approximately

the same time complexity as the best case. O(n logn).

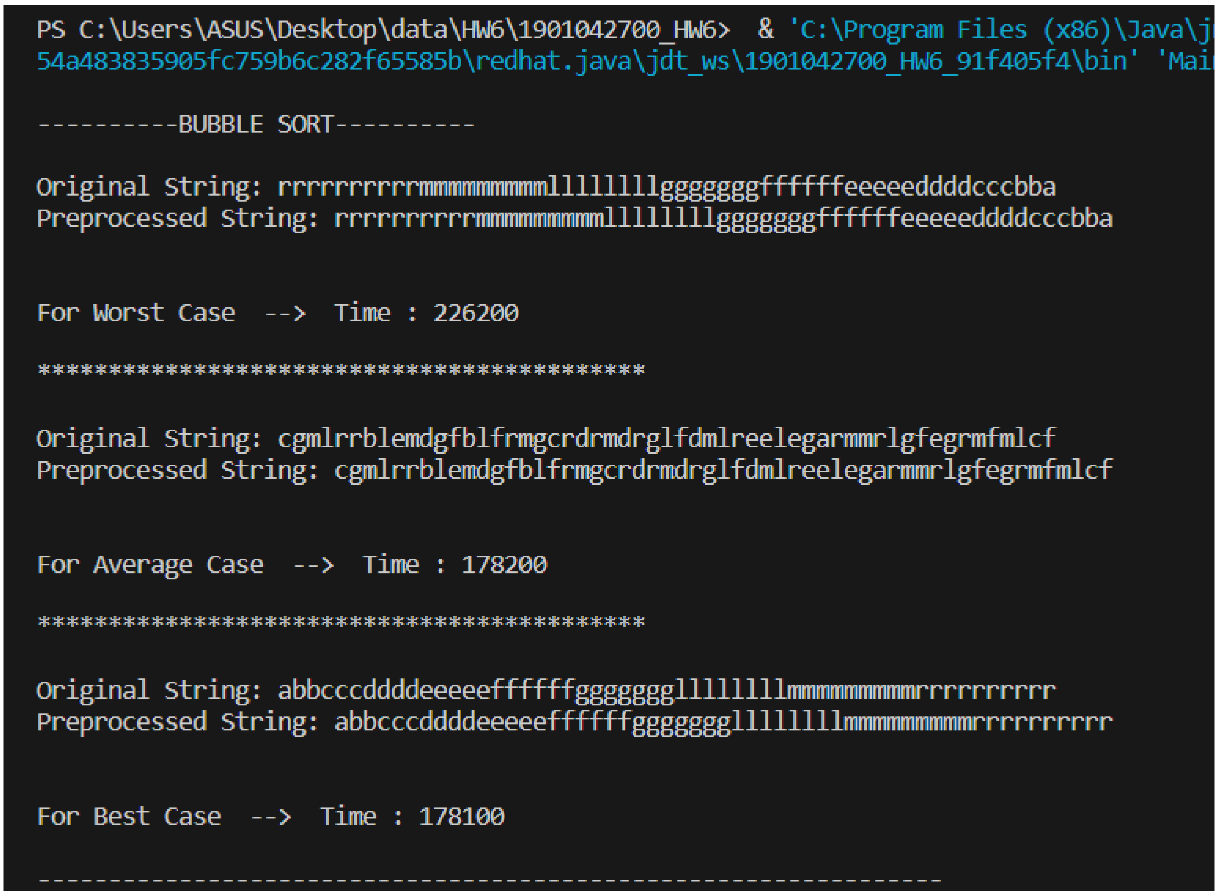
Worst Case -> If an array has n elements and the pivot element is always chosen as the largest element, this is an example of a worst case. (n-1) + (n-2) + (n-3) + ... + 1 = n*(n-1)/2 = (n^2 - n) / 2 = O(n^2) .

Merge Sort:

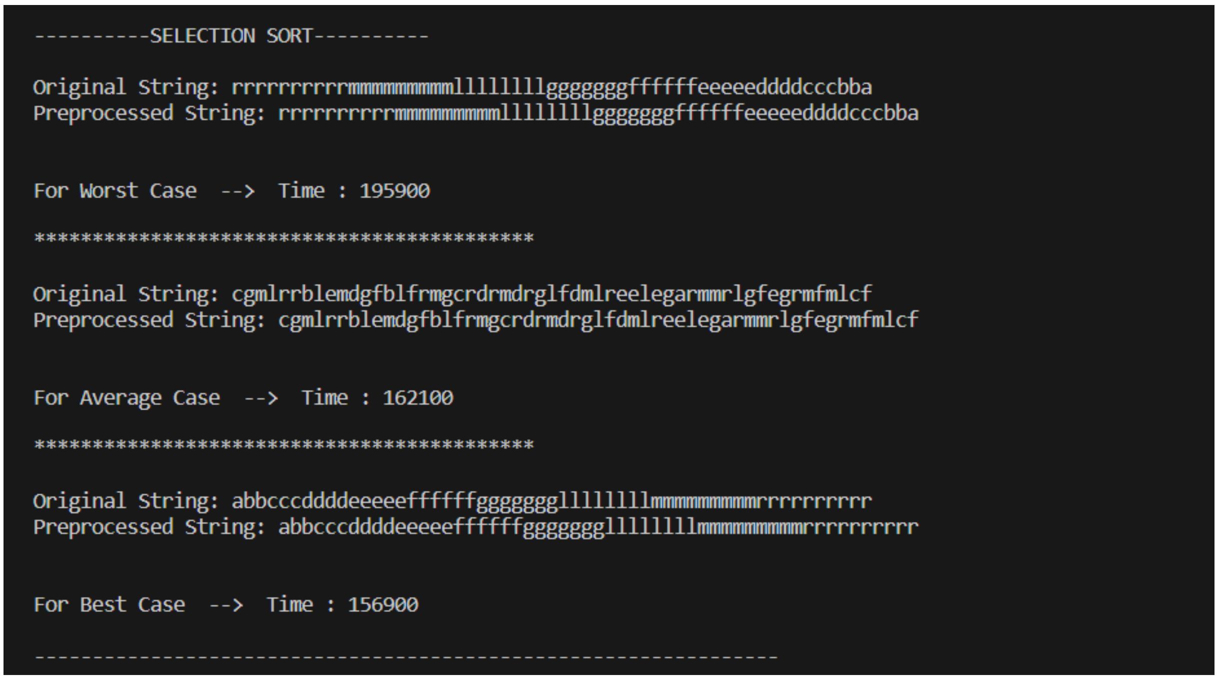
In the Merge Sort algorithm, the time complexity is the same in best case, average case, worst case. Because it performs the same steps in every scenario. It recursively divides the array in half and then sorts and combines the smallest parts. O(n logn).

PART 2)

Case/Sorting Name	**Bubble**	**Selection**	**Insertion**	**Quick**	**Merge**
Worst	226200	195900	236300	186500	121300
Average	178200	162100	193200	133000	96300
Best	178100	156900	139500	107600	119000

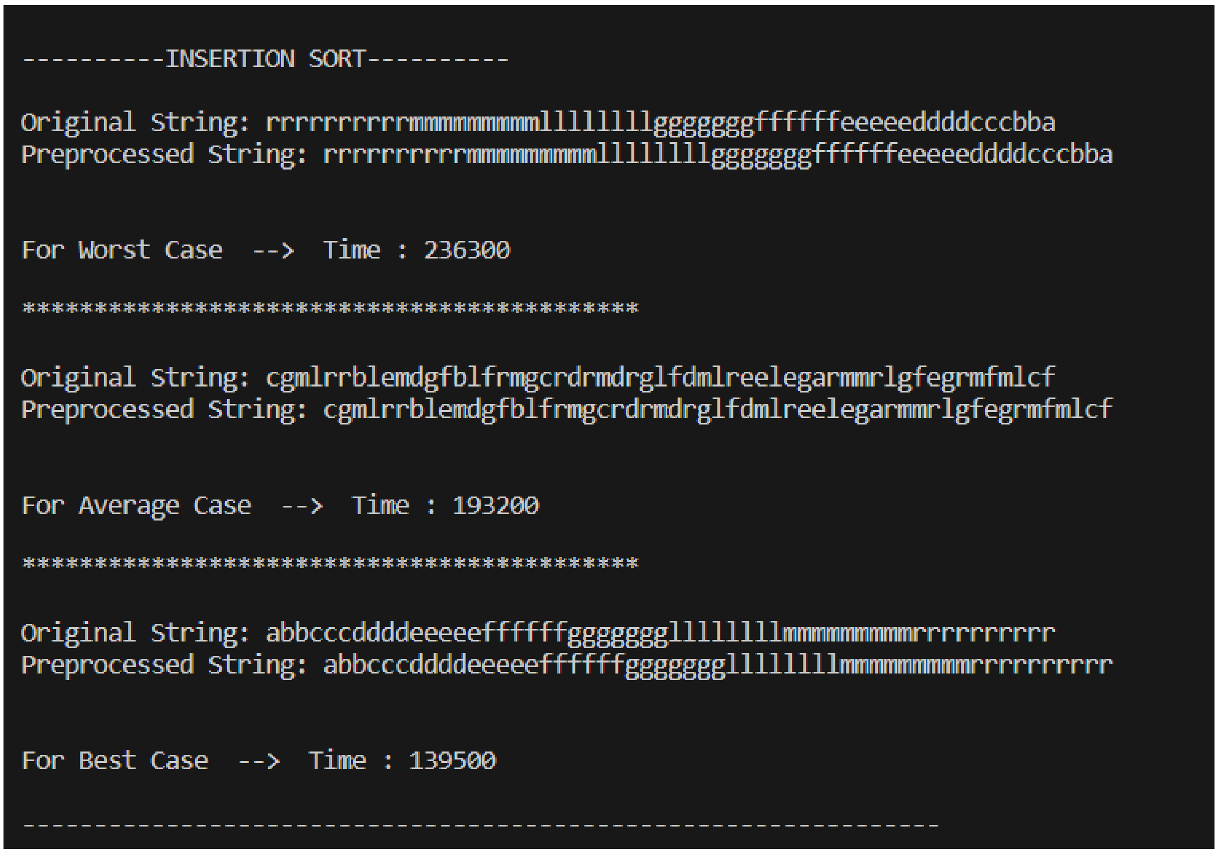


```
PS C:\Users\ASUS\Desktop\data\HW6\1901042700_HW6>  & 'C:\Program Files (x86)\Java\j
54a483835905fc759b6c282f65585b\redhat.java\jdt_ws\1901042700_HW6_91f405f4\bin' 'Mai

----------BUBBLE SORT----------

Original String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba
Preprocessed String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba


For Worst Case  -->  Time : 226200

*******************************************

Original String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf
Preprocessed String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf


For Average Case  -->  Time : 178200

*******************************************

Original String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr
Preprocessed String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr


For Best Case  -->  Time : 178100

----------------------------------------------------------------
```

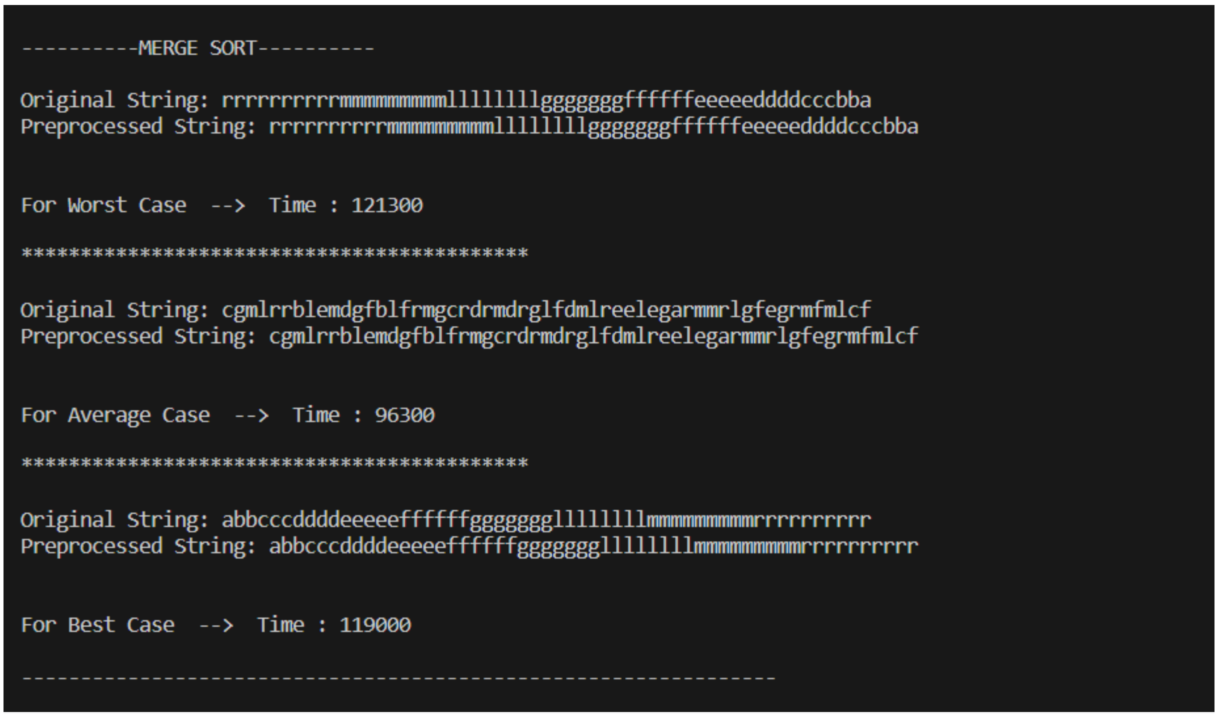


```
----------SELECTION SORT----------

Original String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba
Preprocessed String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba


For Worst Case  -->  Time : 195900

*******************************************

Original String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf
Preprocessed String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf


For Average Case  -->  Time : 162100

*******************************************

Original String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr
Preprocessed String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr


For Best Case  -->  Time : 156900

---------------------------------------------------------------
```

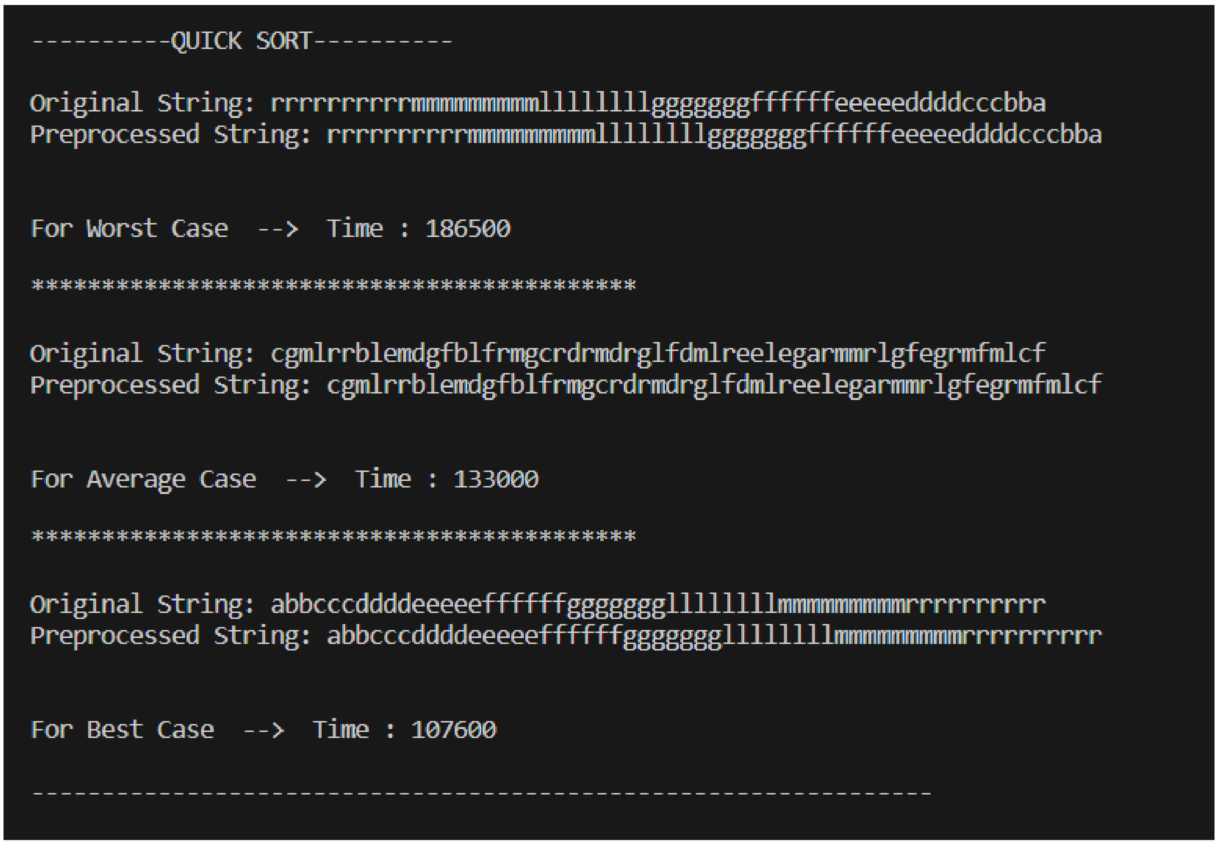
Özlem MALKOÇ
1901042700



```
----------INSERTION SORT----------

Original String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba
Preprocessed String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba


For Worst Case  -->  Time : 236300

********************************************

Original String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf
Preprocessed String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf


For Average Case  -->  Time : 193200

********************************************

Original String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr
Preprocessed String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr


For Best Case  -->  Time : 139500

----------------------------------------------------------------
```



```
----------MERGE SORT----------

Original String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba
Preprocessed String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba


For Worst Case  -->  Time : 121300

********************************************

Original String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf
Preprocessed String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf


For Average Case  -->  Time : 96300

********************************************

Original String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr
Preprocessed String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr


For Best Case  -->  Time : 119000

----------------------------------------------------------------
```



```
----------QUICK SORT----------

Original String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba
Preprocessed String: rrrrrrrrrrmmmmmmmmmllllllllgggggggffffffeeeeeddddcccbba


For Worst Case  -->  Time : 186500

*******************************************

Original String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf
Preprocessed String: cgmlrrblemdgfblfrmgcrdrmdrglfdmlreelegarmmrlgfegrmfmlcf


For Average Case  -->  Time : 133000

*******************************************

Original String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr
Preprocessed String: abbcccddddeeeeeffffffgggggggllllllllmmmmmmmmmrrrrrrrrrr


For Best Case  -->  Time : 107600

-----------------------------------------------------------------
```

PART 3)

In the Best Case, the theoretical time complexity of BubbleSort, Selectionsort, Insertionsort is O(n). The durations of these three algorithms are close. The time complexity of Quick and Merge Sort is O(nlogn). The values of Quick and Merge sort are also close to each other. However, when these values are compared, it is expected that the algorithms with o(n) time complexity will have less duration in the best case case. However, the durations are higher. There may be different reasons for this inconsistency. Actual Time Complexity, Constant Factors and Overheads, Data Distribution, Language and Compiler Optimizations.

In the Average Case, the theoretical time complexity of BubbleSort, Selectionsort, Insertionsort is O(n^2). .The times of these three algorithms are close. The time complexity of Quick and Merge Sort is O(nlogn). The values of Quick and Merge sort are also close to each other. When I compared the values, I saw that values with O(nlogn) time complexity are actually less than values with O(n^2) time complexity.

Özlem MALKOÇ
1901042700

In the Worst Case, the theoretical time complexity of BubbleSort, Selectionsort, Insertionsort and Quicksort is O(n^2). .The times of these four algorithms are close. The time complexity of Merge Sort is O(nlogn). When I compared the values, I found that the value with O(nlogn) time complexity is actually less than the value with O(n^2) time complexity.

PART 4)

In the quick sort algorithm, a pivot value is selected and the array is divided into parts considering the pivot value. While large elements are placed to the right of the pivot, small elements are placed to the left. According to the pivot value, if there are always large numbers on the right and small numbers on the left, a new pivot is determined and the process continues until the array becomes sorted. .Elements with the same value may not be sorted in order of input. Because the sorting criteria are only values. I will try to explain with an example.

Key->Value

c->5

d->4

e->1

f->3

g->8

l->1

Suppose the pivot is the middle 1 .

If we sort by Value values;

```
// l d e f g c
   1 4 1 3 8 5


//l e d f g  c
  1 1 4 3 8 5
```

It goes on like this. What we need to pay attention to here is that if we used a different algorithm, "e" value would come before "l" value, while the input order of different characters with the same value was not preserved in this algorithm.