GTU Department of Computer Engineering CSE 222/505 - Spring 2023

Homework 7 – Report

Erkut Dere - 1801042646

a) Best, average, and worst-case time complexities analysis of each sorting algorithms:

At the beginning of each sorting algorithm, I constructed the aux with the map's key values so I will not include that for loop into my calculations for the clarity of my explanations.

Merge Sort:

In my merge sort algorithm, I use a recursive method to divide the map by halves. After that, I merge them by comparing the count value of each character in the map. The effort to do each merge is O(n). Each recursive step splits the map in half while doing that the number of lines that require merging is logn. So, the total time complexity for reconstructing the aux array through merging is O(nlogn). In this sorting algorithm time complexity does not depend on whether the map is constructed in order or not. So, time complexity in best, average and worst case is nlogn. This sorting algorithm would work in the same way as constructing the sorted aux array in each case.

Selection Sort:

For the selection sort class there are two for loop goes through to the map, one loop to select a select element that will be the minimum O(n) and inside of the main loop will compare that element with other elements in the map, that will take O(n) time too. So, the total time complexity of this algorithm will take O(n^2). The time complexity will not change for the other scenarios whether the characters count value is sorted or not.

Insertion Sort:

The algorithm takes O(n^2) in worst and Θ(n^2) average case time complexities. The best case is Ω(n), because if the count values in the map is sorted, we don't need to check the beforehand values in the map so the algorithm will work only once with the outer for loop.

Bubble Sort:

The algorithm uses one nested for loops that iterates over aux array and compares count value of each character. That's why time complexity will be O(n^2) in worst case, in average case it will still iterate over for loop and swap values so the time complexity will be Θ(n^2). The best case is different from other the other cases because if the array is sorted, I mean the count value of each character in the input is sorted, it does not swap values so there would be only comparison and will not be swapped in the inner loop. That is why the best-case time complexity will be Ω(n).

Quick Sort:

In quick sort, we recursively perform a partition where we divide the map into two parts. We can say that this takes logn time before we reach the map of size 1. Also, each recursive call makes O(n) time work so, overall time complexity is O(n logn) in the best-case. When partitioning the map sometimes we may end up with a map that is empty, in those situations the other maps will have one element less than the one just split. Thus, we might have n levels of recursive call instead of logn so, overall time complexity will be O(n^2) in those situations. (Worst-case scenario)

Table for summary:	Best Case	Average Case	Worst Case
Merge Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)
Quick Sort	Ω(n log n)	Θ(n log n)	O(n^2)

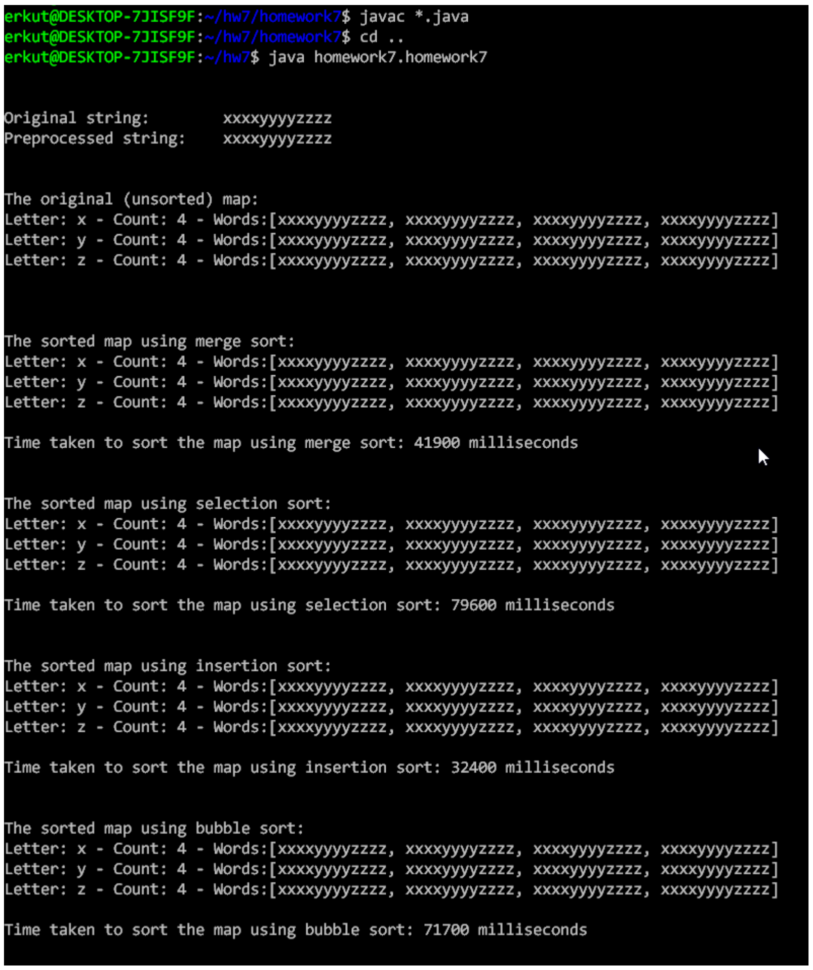
b) Running time of each algorithm:

Let’s run the program with these inputs:

Best case input:”xxxxyyyyzzzz”

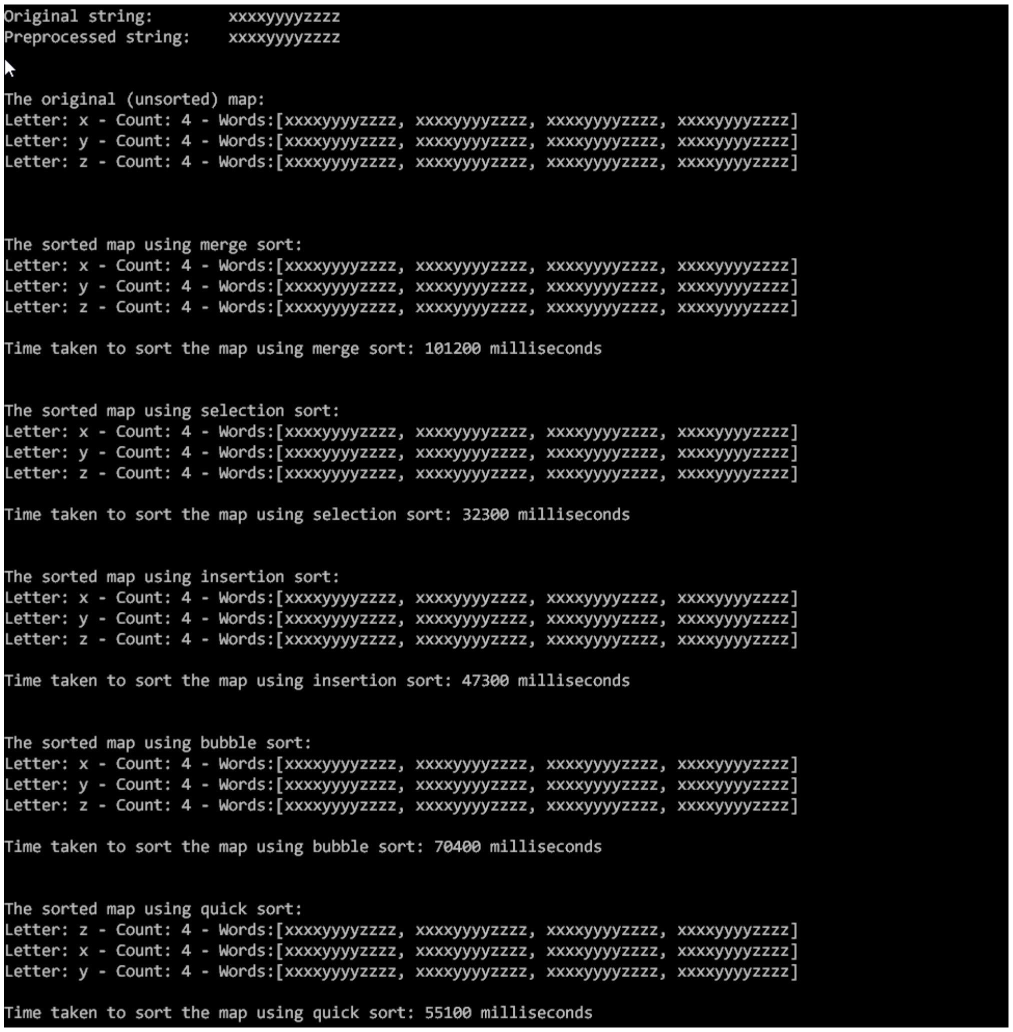
Average case input:”yxyzzzzxxxxx”

Worst-case input:”xzyxzxzxzxyx”



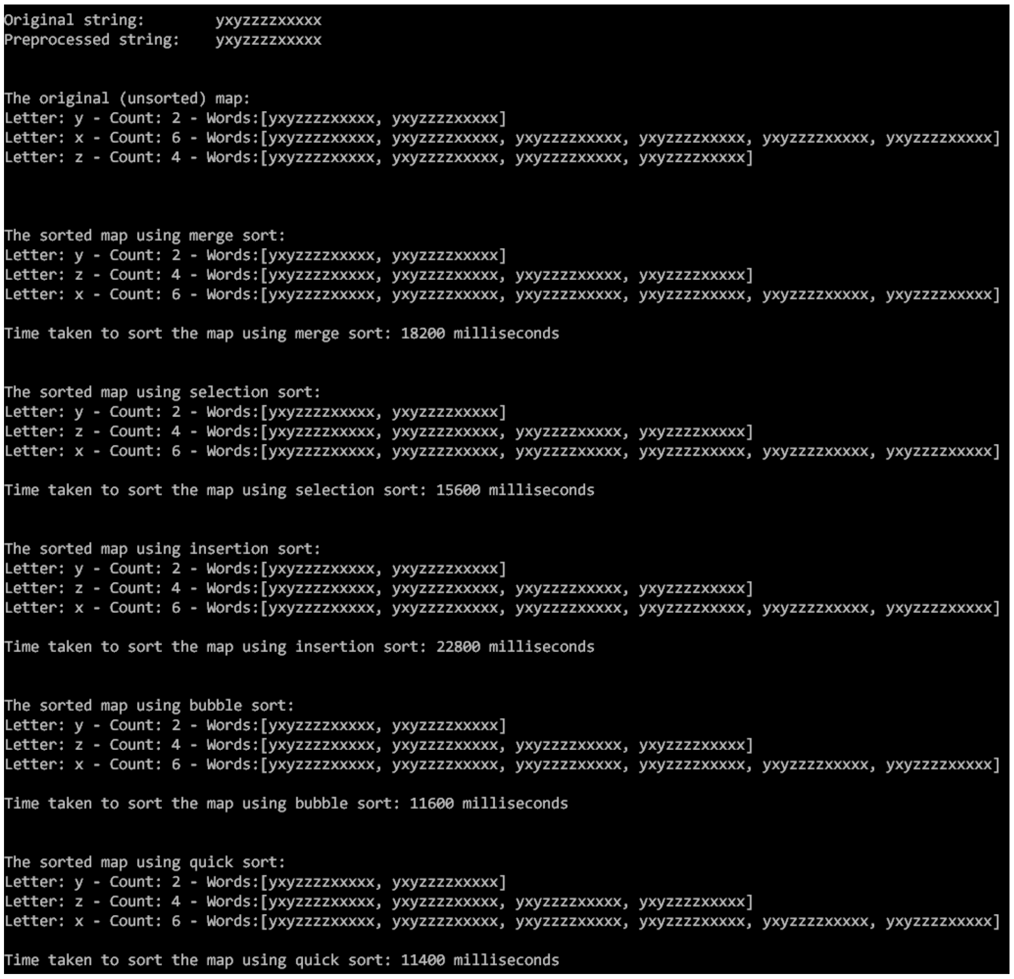
```
erkut@DESKTOP-7JISF9F:~/hw7/homework7$ javac *.java
erkut@DESKTOP-7JISF9F:~/hw7/homework7$ cd ..
erkut@DESKTOP-7JISF9F:~/hw7$ java homework7.homework7


Original string:        xxxxyyyyzzzz
Preprocessed string:    xxxxyyyyzzzz


The original (unsorted) map:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]



The sorted map using merge sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using merge sort: 41900 milliseconds


The sorted map using selection sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using selection sort: 79600 milliseconds


The sorted map using insertion sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using insertion sort: 32400 milliseconds


The sorted map using bubble sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using bubble sort: 71700 milliseconds
```

Best case input and output:

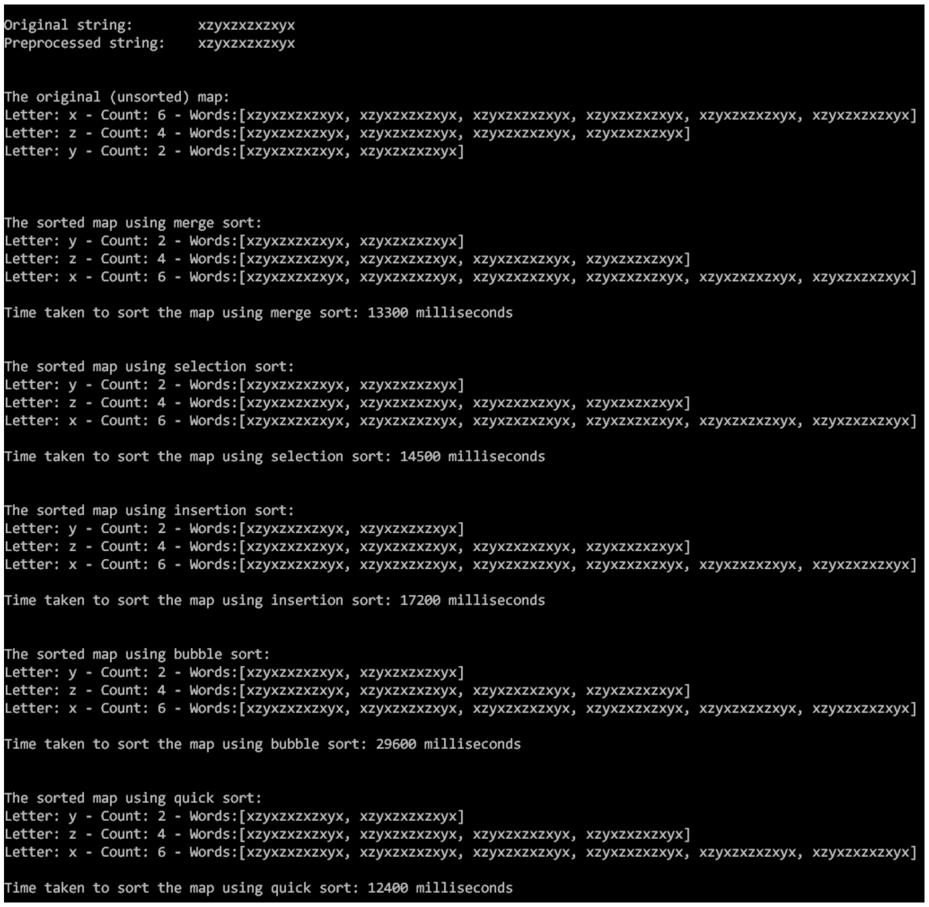


```
Original string:        xxxxyyyyzzzz
Preprocessed string:    xxxxyyyyzzzz


The original (unsorted) map:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]



The sorted map using merge sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using merge sort: 101200 milliseconds


The sorted map using selection sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using selection sort: 32300 milliseconds


The sorted map using insertion sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using insertion sort: 47300 milliseconds


The sorted map using bubble sort:
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using bubble sort: 70400 milliseconds


The sorted map using quick sort:
Letter: z - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: x - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]
Letter: y - Count: 4 - Words:[xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz, xxxxyyyyzzzz]

Time taken to sort the map using quick sort: 55100 milliseconds
```

Average case input and output:


```
Original string:        yxyzzzzxxxxx
Preprocessed string:    yxyzzzzxxxxx


The original (unsorted) map:
Letter: y - Count: 2 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: x - Count: 6 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: z - Count: 4 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]



The sorted map using merge sort:
Letter: y - Count: 2 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: z - Count: 4 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: x - Count: 6 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]

Time taken to sort the map using merge sort: 18200 milliseconds


The sorted map using selection sort:
Letter: y - Count: 2 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: z - Count: 4 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: x - Count: 6 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]

Time taken to sort the map using selection sort: 15600 milliseconds


The sorted map using insertion sort:
Letter: y - Count: 2 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: z - Count: 4 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: x - Count: 6 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]

Time taken to sort the map using insertion sort: 22800 milliseconds


The sorted map using bubble sort:
Letter: y - Count: 2 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: z - Count: 4 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: x - Count: 6 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]

Time taken to sort the map using bubble sort: 11600 milliseconds


The sorted map using quick sort:
Letter: y - Count: 2 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: z - Count: 4 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]
Letter: x - Count: 6 - Words:[yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx, yxyzzzzxxxxx]

Time taken to sort the map using quick sort: 11400 milliseconds
```

Worst case input and output:



```
Original string:       xzyxzxzxzxyx
Preprocessed string:   xzyxzxzxzxyx


The original (unsorted) map:
Letter: x - Count: 6 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: z - Count: 4 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: y - Count: 2 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx]



The sorted map using merge sort:
Letter: y - Count: 2 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: z - Count: 4 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: x - Count: 6 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]

Time taken to sort the map using merge sort: 13300 milliseconds


The sorted map using selection sort:
Letter: y - Count: 2 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: z - Count: 4 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: x - Count: 6 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]

Time taken to sort the map using selection sort: 14500 milliseconds


The sorted map using insertion sort:
Letter: y - Count: 2 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: z - Count: 4 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: x - Count: 6 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]

Time taken to sort the map using insertion sort: 17200 milliseconds


The sorted map using bubble sort:
Letter: y - Count: 2 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: z - Count: 4 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: x - Count: 6 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]

Time taken to sort the map using bubble sort: 29600 milliseconds


The sorted map using quick sort:
Letter: y - Count: 2 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: z - Count: 4 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]
Letter: x - Count: 6 - Words:[xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx, xzyxzxzxzxyx]

Time taken to sort the map using quick sort: 12400 milliseconds
```

Running time table for each algorithm:

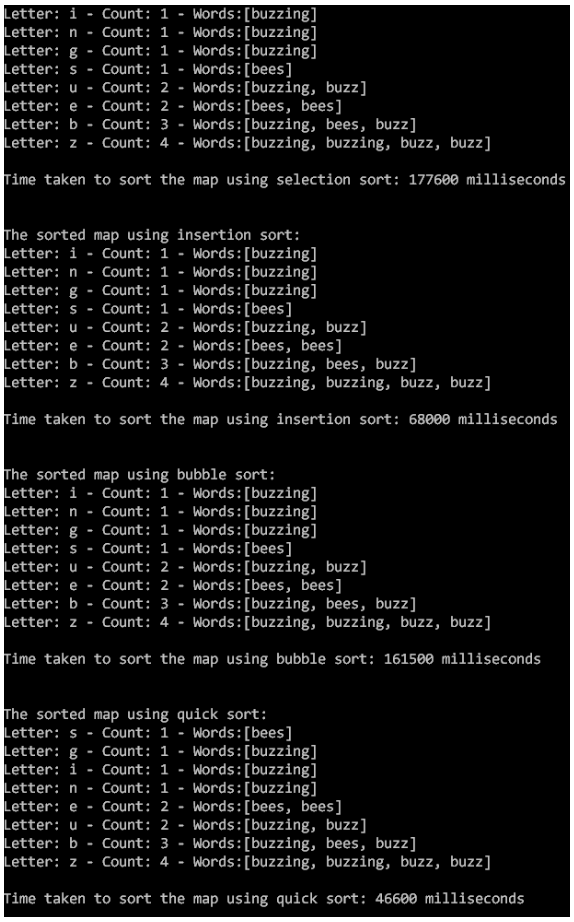
	Best Case	Average Case	Worst Case
Merge Sort	101200 ns	18200 ns	13300 ns
Selection Sort	32300 ns	15600 ns	14500 ns
Insertion Sort	47300 ns	22800 ns	17200 ns
Bubble Sort	70400 ns	11600 ns	29600 ns
Quick Sort	55100 ns	11400 ns	12400 ns

c) Comparison of the sorting algorithms:

For the best-case scenario, in theory for small input data insertion sort merge sort and bubble sort's time complexity should be like each other, in my input selection sort came first and insertion sort came second. This input that I am giving in my code can be considered as small input so we can expect this to emerge. But in the average and worst case we can see that quick sort algorithm is faster than the other algorithms. In the worst-case scenario merge sort and selection sort is nearly close to quick sort. But in the end, we know that quick sort works faster than the other four algorithms in unsorted situations. Merge sort got slower than other algorithms in best-case, insertion sort and bubble sort have the slowest time in average and worst-case in scenarios respectively.

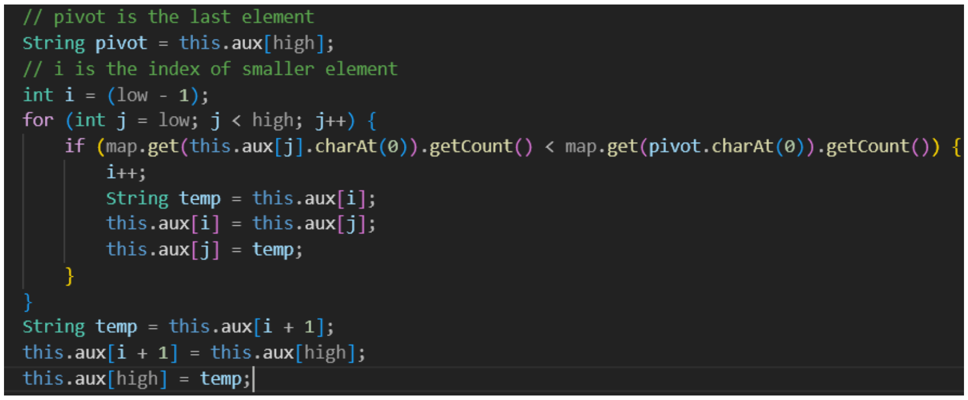
d) Which algorithm does not preserve the ordering in some situations?

If we give this input: "Buzzing bees buzz.". In quick sort I got different ordering from other four algorithm:



```
Letter: i - Count: 1 - Words:[buzzing]
Letter: n - Count: 1 - Words:[buzzing]
Letter: g - Count: 1 - Words:[buzzing]
Letter: s - Count: 1 - Words:[bees]
Letter: u - Count: 2 - Words:[buzzing, buzz]
Letter: e - Count: 2 - Words:[bees, bees]
Letter: b - Count: 3 - Words:[buzzing, bees, buzz]
Letter: z - Count: 4 - Words:[buzzing, buzzing, buzz, buzz]

Time taken to sort the map using selection sort: 177600 milliseconds


The sorted map using insertion sort:
Letter: i - Count: 1 - Words:[buzzing]
Letter: n - Count: 1 - Words:[buzzing]
Letter: g - Count: 1 - Words:[buzzing]
Letter: s - Count: 1 - Words:[bees]
Letter: u - Count: 2 - Words:[buzzing, buzz]
Letter: e - Count: 2 - Words:[bees, bees]
Letter: b - Count: 3 - Words:[buzzing, bees, buzz]
Letter: z - Count: 4 - Words:[buzzing, buzzing, buzz, buzz]

Time taken to sort the map using insertion sort: 68000 milliseconds


The sorted map using bubble sort:
Letter: i - Count: 1 - Words:[buzzing]
Letter: n - Count: 1 - Words:[buzzing]
Letter: g - Count: 1 - Words:[buzzing]
Letter: s - Count: 1 - Words:[bees]
Letter: u - Count: 2 - Words:[buzzing, buzz]
Letter: e - Count: 2 - Words:[bees, bees]
Letter: b - Count: 3 - Words:[buzzing, bees, buzz]
Letter: z - Count: 4 - Words:[buzzing, buzzing, buzz, buzz]

Time taken to sort the map using bubble sort: 161500 milliseconds


The sorted map using quick sort:
Letter: s - Count: 1 - Words:[bees]
Letter: g - Count: 1 - Words:[buzzing]
Letter: i - Count: 1 - Words:[buzzing]
Letter: n - Count: 1 - Words:[buzzing]
Letter: e - Count: 2 - Words:[bees, bees]
Letter: u - Count: 2 - Words:[buzzing, buzz]
Letter: b - Count: 3 - Words:[buzzing, bees, buzz]
Letter: z - Count: 4 - Words:[buzzing, buzzing, buzz, buzz]

Time taken to sort the map using quick sort: 46600 milliseconds
```

We can see that the order of the letter's count value is not preserved in the quick sort algorithm. This was caused because of the quick sort algorithms working principle.



```
// pivot is the last element
String pivot = this.aux[high];
// i is the index of smaller element
int i = (low - 1);
for (int j = low; j < high; j++) {
    if (map.get(this.aux[j].charAt(0)).getCount() < map.get(pivot.charAt(0)).getCount()) {
        i++;
        String temp = this.aux[i];
        this.aux[i] = this.aux[j];
        this.aux[j] = temp;
    }
}
String temp = this.aux[i + 1];
this.aux[i + 1] = this.aux[high];
this.aux[high] = temp;
```

We can see in the code that we are swapping the elements according to the pivot's position, that is why ordering is not preserved, that means that we are neglecting their original positions, and consequently we got different outcomes from other algorithms.