Merge sort

Best : O(nlogn)

Average : O(nlogn)

Worst : O(nlogn)

merge sort doesn't care about the state of the array. It divides the whole array into two parts (reaching logn depth) and then joins the sequence from these parts again sequentially (merging is done with n). so in any case complexity is nlogn.

Selection Sort

Best : O(n^2)

Average : O(n^2)

Worst : O(n^2)

selection sort doesn't care about the state of the array. It traverses the unordered part of the array once and finds the smallest element, then adds that element to the end of the sorted part. complexity is n^2 as each element is processed n times

Insertion Sort

Best : O(n)

Average : O(n^2)

Worst : O(n^2)

In insertion sort, elements before an element are looked at and swap operation is performed with elements larger than itself.

for example in array [21,22,23.5,6] for 5 elements, respectively [21,22,5,23.6] -> [21.5,22,23.6] -> [5,21,22, 23.6] operations are performed.

so in worst case each element will be processed n times but in best case for example if the array is sorted or there are several elements that break the order it will be processed n times so at best it will be O(n) otherwise O(n^2)

Bubble Sort

Best : O(n)

Average : O(n^2)

Worst : O(n^2)

By continuously swapping in bubble sort, the largest or smallest element is placed at the end of the array. Since each element is processed n times, the time complexity becomes O(n^2). But we can reduce this situation to O(n) in the best case with a bool variable. if no swap is done, it means the array is sorted and sorting stops.

Quick Sort

Best : O(nlogn)

Average : O(nlogn)

Worst : O(n^2)

Quick sort algorithm usually has O(nlogn) complexity (we can say it is nlogn because it uses divide and conquer). However, in the worst case, the complexity of the algorithm is O(n^2) if the pivot element is chosen as the smallest or largest element of the array each time. In this case, after each fragmentation, one of the subsequences is much shorter than the other, and the fragmentation is done n times before the array is completely sorted. However, randomizing the pivot element or using pivot selection optimizing techniques can make the worst case rare and ensure that quick sort is often O(nlogn) complexity.

Running times

	Best Case	Average Case	Worst Case
Bubble Sort	28301	54127	79000
Insertion Sort	26900	58652	68600
Merge Sort	70000	38955	12600
Quick Sort	152900	79526	13500
Selection Sort	90100	64795	56800

I tried to enlarge the test cases as much as possible, but since we are going over the letters, the map size is 26 at the most, which makes it impossible to try with a large case. Because of this situation, there are illogicalities in terms of time.

Selection Sort

- It can be inefficient on large datasets as its complexity is O(n^2).
- It is an in-place sorting algorithm and uses less memory.
- It is not a stable algorithm, so the sort order of elements with the same value may change.

Insertion Sort

- Like the Selection Sort, it can be inefficient on large datasets as its complexity is O(n^2).
- It is an in-place sorting algorithm and uses less memory.
- It is a stable algorithm, so the sort order of elements with the same value does not change.

Bubble Sort

- It can be inefficient on large datasets as its complexity is O(n^2).
- It is an in-place sorting algorithm and uses less memory.
- It is a stable algorithm.

Quick Sort

- It works effectively on large datasets.
- In the mean case it has O(nlogn) complexity.
- With a good selection of pivot elements, it is generally faster than other algorithms.
- It is an in-place sorting algorithm, with this feature it stands out from merge sort in terms of advantage.
- It is not a stable algorithm, so the sort order of elements with the same value may change.

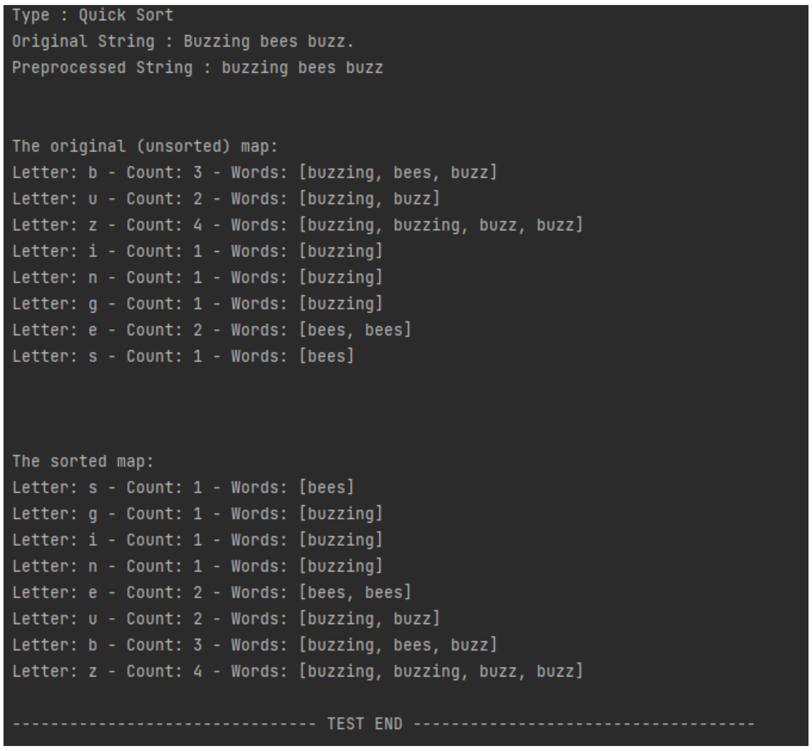
Merge Sort

- It works effectively on large datasets as its complexity is O(nlogn).
- It is not an in-place sorting algorithm, it uses more memory.
- It is a stable algorithm, so the sort order of elements with the same value does not change.

In the previous algorithm comparison section, I talked about stable and unstable algorithms. For example, while quick sort is unstable, merge sort is a stable algorithm and the ordering status of elements with the same value can change (if the first element with the same value in the unsorted array is sorted, it is stable if it comes before the others in the same order, otherwise it will be unstable)

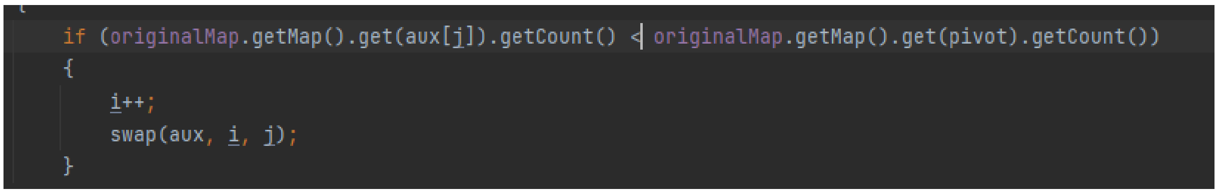
The unstable part of algorithms occurs in comparison and swapping part of code.

For example in quick sort, despite correct ordering, it unstable



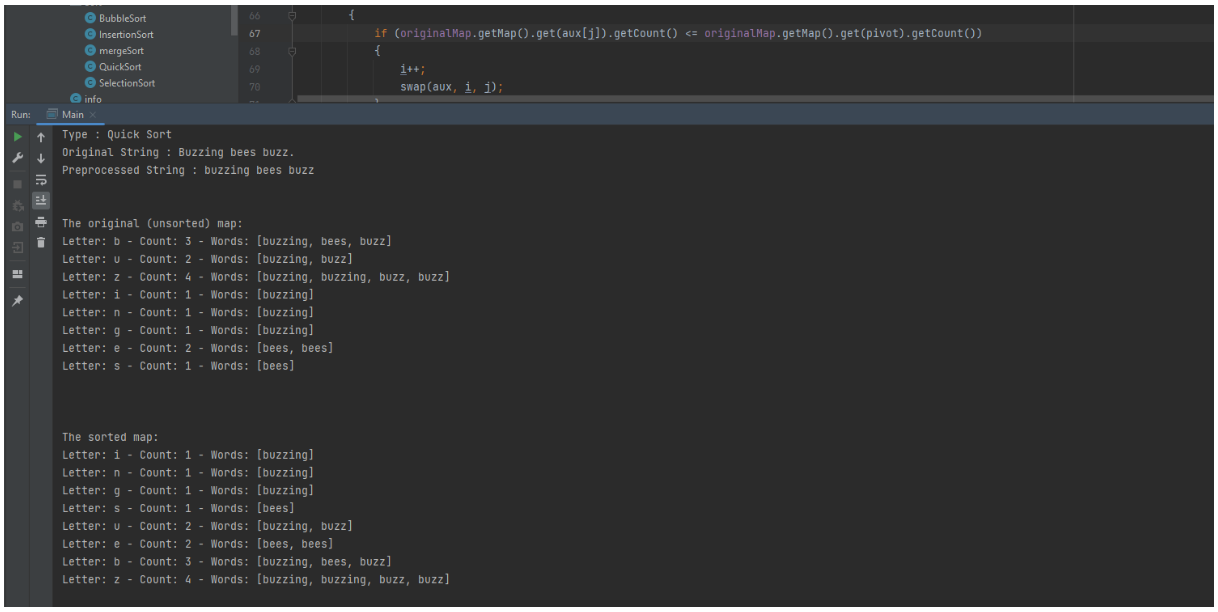
```
Type : Quick Sort
Original String : Buzzing bees buzz.
Preprocessed String : buzzing bees buzz


The original (unsorted) map:
Letter: b - Count: 3 - Words: [buzzing, bees, buzz]
Letter: u - Count: 2 - Words: [buzzing, buzz]
Letter: z - Count: 4 - Words: [buzzing, buzzing, buzz, buzz]
Letter: i - Count: 1 - Words: [buzzing]
Letter: n - Count: 1 - Words: [buzzing]
Letter: g - Count: 1 - Words: [buzzing]
Letter: e - Count: 2 - Words: [bees, bees]
Letter: s - Count: 1 - Words: [bees]



The sorted map:
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]

-------------------------------- TEST END ------------------------------------
```

Because of this part



```
    if (originalMap.getMap().get(aux[j]).getCount() < originalMap.getMap().get(pivot).getCount())
    {
        i++;
        swap(aux, i, j);
    }
```

İf we change ‘<’ with ‘<=’



```
        {
            if (originalMap.getMap().get(aux[j]).getCount() <= originalMap.getMap().get(pivot).getCount())
            {
                i++;
                swap(aux, i, j);
```

```
Type : Quick Sort
Original String : Buzzing bees buzz.
Preprocessed String : buzzing bees buzz


The original (unsorted) map:
Letter: b - Count: 3 - Words: [buzzing, bees, buzz]
Letter: u - Count: 2 - Words: [buzzing, buzz]
Letter: z - Count: 4 - Words: [buzzing, buzzing, buzz, buzz]
Letter: i - Count: 1 - Words: [buzzing]
Letter: n - Count: 1 - Words: [buzzing]
Letter: g - Count: 1 - Words: [buzzing]
Letter: e - Count: 2 - Words: [bees, bees]
Letter: s - Count: 1 - Words: [bees]



The sorted map:
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]
```

Now it sorted as stable