GTU DEPARTMENT OF COMPUTER ENGINERRING

CSE 222/505 – SPRING 2023

HOMEWORK #07 REPORT

Ahmet Yiğit
200104004066

1. **Best – Average- Worst Case Time Complexity of Code**
    a. **Part 1- A TIME COMPLECITY ANALYSIS**

	Best Case	Average Case	Worst Case
Merge Sort	O(n log(n))	O(n log(n))	O(n log(n))
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O (n)	O(n^2)	O(n^2)
Bubble Sort	O (n)	O(n^2)	O(n^2)
Quick Sort	O(n log(n))	O(n log(n))	O(n^2)

    b. **Part 2- B RUNNING TIME**

Nanoseconds	Best Case	Average Case	Worst Case
Merge Sort	51000	52000	50000
Selection Sort	23000	30000	21000
Insertion Sort	8000	10000	16000
Bubble Sort	14000	15000	16000
Quick Sort	15000	15000	17000

    c. **Part 2- C COMPARISION**
        i. **Best Case**
            1. *Merge Sort: O(N log N)*
            2. *Selection Sort: O(N^2)*
            3. *Insertion Sort: O(N)*
            4. *Bubble Sort: O(N)*
            5. *Quick Sort: O(N log N)*

                a. *In the best case scenario, the time complexity of Merge Sort, Quick Sort, Insertion Sort, and Bubble Sort is the best. Among them, Merge Sort and Quick Sort have the same time complexity, but Merge Sort is generally considered more efficient due to its robust nature and consistent performance*

        ii. **Average Case**
            1. *Merge Sort: O(N log N)*
            2. *Selection Sort: O(N^2)*
            3. *Insertion Sort: O(N^2)*
            4. *Bubble Sort: O(N^2)*
            5. *Quick Sort: O(N log N)*

                a. *On average, Merge Sort and Quick Sort are characterized by a time complexity of O(N log N). They are generally faster compared to Selection Sort, Insertion Sort, and Bubble Sort which have quadratic time complexities of O(N^2).*

iii. **Worst Case**
   1. *Merge Sort: O(N log N)*
   2. *Selection Sort: O(N^2)*
   3. *Insertion Sort: O(N^2)*
   4. *Bubble Sort: O(N^2)*
   5. *Quick Sort: O(N^2)*

      a. *In the worst case, all the sorting algorithms except Merge Sort have a time complexity of O(N^2). Merge Sort remains efficient with its time complexity of O(N log N).*

d. **PART 2- D DIFFERENCE**
   i. **Merge Sort**
      1. *Merge sort is a stable sorting algorithm, which means it maintains the relative order of elements with equal values. Therefore, it preserves the input ordering when multiple elements have the same count value.*
   ii. **Selection Sort**
      1. *Selection sort is not a stable sorting algorithm. When multiple elements have the same count value, their order may change during the sorting process. The algorithm does not guarantee to preserve the input ordering.*
   iii. **Insertion Sort**
      1. *Insertion sort is a stable sorting algorithm. It compares elements from left to right and shifts them if necessary, but it only moves an element if it is strictly smaller than the previous element. Therefore, when multiple elements have the same count value, their order will remain unchanged, preserving the input ordering.*
   iv. **Bubble Sort**
      1. *Similar to insertion sort, bubble sort is also a stable sorting algorithm. It compares adjacent elements and swaps them if they are in the wrong order. When multiple elements have the same count value, their order will be maintained, ensuring the input ordering.*
   v. **Quick Sort**
      1. *Quick sort is not a stable sorting algorithm, meaning it does not guarantee to preserve the relative order of elements with equal values. Therefore, Quick sort does not ensure the preservation of input ordering when multiple elements have the same count value.*

e. **Merge Sort**
   i. *Best-case complexity: O(n log n)*
   ii. *Average-case complexity: O(n log n)*
   iii. *Worst-case complexity: O(n log n)*

1. *Merge sort has a consistent time complexity of O(n log n) in all cases. It consistently divides the input into halves until individual elements are obtained, and then merges them back in sorted order. The recursive nature of merge sort guarantees a balanced splitting, resulting in the same time complexity for best, average, and worst cases.*
2. *Also we can understand that merge sort does not depends on the cases because of its algorithm.*

f. Selection Sort

i. Best-case complexity: O(n^2)
ii. Average-case complexity: O(n^2)
iii. *Worst-case complexity: O(n^2)*

1. *Selection sort involves repeatedly selecting the minimum element from the unsorted portion and placing it in the sorted portion. Regardless of the initial order, the algorithm performs the same number of comparisons and swaps, resulting in a time complexity of O(n^2) for all cases.*

g. Insertion Sort

i. Best-case complexity: O(n)
ii. Average-case complexity: O(n^2)
iii. *Worst-case complexity: O(n^2)*
1. *In the best case, if the input is already sorted, insertion sort performs only one comparison per element, resulting in a linear time complexity of O(n). However, on average and in the worst case, the algorithm may require shifting multiple elements for each element, resulting in a quadratic time complexity of O(n^2).*

h. Bubble Sort

i. Best-case complexity: O(n)
ii. Average-case complexity: O(n^2)
iii. Worst-case complexity: O(n^2)

1. *Bubble sort compares adjacent elements and swaps them if they are in the wrong order, repeatedly iterating over the list until it becomes sorted. In the best case, if the input is already sorted, bubble sort requires only a single pass, resulting in a linear time complexity of O(n). However, on average and in the worst case, bubble sort requires multiple passes, leading to a quadratic time complexity of O(n^2).*

i. Quick Sort

i. Best-case complexity: O(n log n)
ii. Average-case complexity: O(n log n)
iii. Worst-case complexity: O(n^2)

1. *Quick sort selects a pivot element and partitions the input around it. In the best and average cases, the pivot selection and partitioning lead to a balanced split, resulting in a time complexity of O(n log n). However, in the worst case, if the pivot selection consistently results in the smallest or largest element, quick sort may exhibit a quadratic time complexity of O(n^2). Various techniques, like randomized pivot selection, can mitigate the likelihood of worst-case behavior.*

2. Screen Shoots of Cases

a. Best Case

```
nens1s@Ahmet-MacBook-Air src % javac *.java
nens1s@Ahmet-MacBook-Air src % java main.java
BEST CASE STRING:
Original String:                    a b b c c c d d d d e e e e e f f f f f f g g g g g g g h h h h h h h h i i i i i i i i i j j j j j j j j j k k k k k k k k k k k
Preprocessed String:           a b b c c c d d d d e e e e e f f f f f f g g g g g g g h h h h h h h h i i i i i i i i i j j j j j j j j j k k k k k k k k k k k
The original (unsorted) map:
Letter: a - Count: 1 - Words: [a]
Letter: b - Count: 2 - Words: [b, b]
Letter: c - Count: 3 - Words: [c, c, c]
Letter: d - Count: 4 - Words: [d, d, d, d]
Letter: e - Count: 5 - Words: [e, e, e, e, e]
Letter: f - Count: 6 - Words: [f, f, f, f, f, f]
Letter: g - Count: 7 - Words: [g, g, g, g, g, g, g]
Letter: h - Count: 8 - Words: [h, h, h, h, h, h, h, h]
Letter: i - Count: 9 - Words: [i, i, i, i, i, i, i, i, i]
Letter: j - Count: 10 - Words: [j, j, j, j, j, j, j, j, j, j]
Letter: k - Count: 12 - Words: [k, k, k, k, k, k, k, k, k, k, k, k]


The sorted map using mergeSort:
MERGE SORT Time taken: 52000 milliseconds


The sorted map using selectionSort:
SELECTION SORT Time taken: 19000 milliseconds


The sorted map using insertionSort:
INSERTION SORT Time taken: 10000 milliseconds


The sorted map using bubbleSort:
BUBBLE SORT Time taken: 13000 milliseconds


The sorted map using quickSort:
QUICK SORT Time taken: 16000 milliseconds


The sorted map:
Letter: a - Count: 1 - Words: [a]
Letter: b - Count: 2 - Words: [b, b]
Letter: c - Count: 3 - Words: [c, c, c]
Letter: d - Count: 4 - Words: [d, d, d, d]
Letter: e - Count: 5 - Words: [e, e, e, e, e]
Letter: f - Count: 6 - Words: [f, f, f, f, f, f]
Letter: g - Count: 7 - Words: [g, g, g, g, g, g, g]
Letter: h - Count: 8 - Words: [h, h, h, h, h, h, h, h]
Letter: i - Count: 9 - Words: [i, i, i, i, i, i, i, i, i]
Letter: j - Count: 10 - Words: [j, j, j, j, j, j, j, j, j, j]
Letter: k - Count: 12 - Words: [k, k, k, k, k, k, k, k, k, k, k, k]
```

b. Average Case

```
nens1s@Ahmet-MacBook-Air src % javac *.java
nens1s@Ahmet-MacBook-Air src % java main.java
BEST CASE STRING:
Original String:                        a b b c c c d d d d e e e e e f f f f f f g g g g g g g h h h h h h h h k k k k k k k k k k k k k j j j j j j j j j i i i i i i i i i
Preprocessed String:            a b b c c c d d d d e e e e e f f f f f f g g g g g g g h h h h h h h h k k k k k k k k k k k k k j j j j j j j j j i i i i i i i i i
The original (unsorted) map:
Letter: a - Count: 1 - Words: [a]
Letter: b - Count: 2 - Words: [b, b]
Letter: c - Count: 3 - Words: [c, c, c]
Letter: d - Count: 4 - Words: [d, d, d, d]
Letter: e - Count: 5 - Words: [e, e, e, e, e]
Letter: f - Count: 6 - Words: [f, f, f, f, f, f]
Letter: g - Count: 7 - Words: [g, g, g, g, g, g, g]
Letter: h - Count: 8 - Words: [h, h, h, h, h, h, h, h]
Letter: k - Count: 12 - Words: [k, k, k, k, k, k, k, k, k, k, k, k]
Letter: j - Count: 10 - Words: [j, j, j, j, j, j, j, j, j, j]
Letter: i - Count: 9 - Words: [i, i, i, i, i, i, i, i, i]


The sorted map using mergeSort:
MERGE SORT Time taken: 57000 milliseconds


The sorted map using selectionSort:
SELECTION SORT Time taken: 27000 milliseconds


The sorted map using insertionSort:
INSERTION SORT Time taken: 10000 milliseconds


The sorted map using bubbleSort:
BUBBLE SORT Time taken: 14000 milliseconds


The sorted map using quickSort:
QUICK SORT Time taken: 13000 milliseconds


The sorted map:
Letter: a - Count: 1 - Words: [a]
Letter: b - Count: 2 - Words: [b, b]
Letter: c - Count: 3 - Words: [c, c, c]
Letter: d - Count: 4 - Words: [d, d, d, d]
Letter: e - Count: 5 - Words: [e, e, e, e, e]
Letter: f - Count: 6 - Words: [f, f, f, f, f, f]
Letter: g - Count: 7 - Words: [g, g, g, g, g, g, g]
Letter: h - Count: 8 - Words: [h, h, h, h, h, h, h, h]
Letter: i - Count: 9 - Words: [i, i, i, i, i, i, i, i, i]
Letter: j - Count: 10 - Words: [j, j, j, j, j, j, j, j, j, j]
Letter: k - Count: 12 - Words: [k, k, k, k, k, k, k, k, k, k, k, k]
```

c. Worst Case

```
^[[A
nens1s@Ahmet-MacBook-Air src % java main.java
BEST CASE STRING:
Original String:                          k k k k k k k k k k k k j j j j j j j j j j i i i i i i i i i h h h h h h h h g g g g g g g g f f f f f f e e e e e e d d d d c c c b b a
Preprocessed String:            k k k k k k k k k k k k j j j j j j j j j j i i i i i i i i i h h h h h h h h g g g g g g g g f f f f f f e e e e e e d d d d c c c b b a
The original (unsorted) map:
Letter: k - Count: 12 - Words: [k, k, k, k, k, k, k, k, k, k, k, k]
Letter: j - Count: 10 - Words: [j, j, j, j, j, j, j, j, j, j]
Letter: i - Count: 9 - Words: [i, i, i, i, i, i, i, i, i]
Letter: h - Count: 8 - Words: [h, h, h, h, h, h, h, h]
Letter: g - Count: 7 - Words: [g, g, g, g, g, g, g]
Letter: f - Count: 6 - Words: [f, f, f, f, f, f]
Letter: e - Count: 5 - Words: [e, e, e, e, e]
Letter: d - Count: 4 - Words: [d, d, d, d]
Letter: c - Count: 3 - Words: [c, c, c]
Letter: b - Count: 2 - Words: [b, b]
Letter: a - Count: 1 - Words: [a]


The sorted map using mergeSort:
MERGE SORT Time taken: 55000 milliseconds


The sorted map using selectionSort:
SELECTION SORT Time taken: 27000 milliseconds


The sorted map using insertionSort:
INSERTION SORT Time taken: 13000 milliseconds


The sorted map using bubbleSort:
BUBBLE SORT Time taken: 14000 milliseconds


The sorted map using quickSort:
QUICK SORT Time taken: 14000 milliseconds


The sorted map:
Letter: a - Count: 1 - Words: [a]
Letter: b - Count: 2 - Words: [b, b]
Letter: c - Count: 3 - Words: [c, c, c]
Letter: d - Count: 4 - Words: [d, d, d, d]
Letter: e - Count: 5 - Words: [e, e, e, e, e]
Letter: f - Count: 6 - Words: [f, f, f, f, f, f]
Letter: g - Count: 7 - Words: [g, g, g, g, g, g, g]
Letter: h - Count: 8 - Words: [h, h, h, h, h, h, h, h]
Letter: i - Count: 9 - Words: [i, i, i, i, i, i, i, i, i]
Letter: j - Count: 10 - Words: [j, j, j, j, j, j, j, j, j, j]
Letter: k - Count: 12 - Words: [k, k, k, k, k, k, k, k, k, k, k, k]
```

3. Example Inputs

a. Selection Sort

```
mens1s@Ahmet-MacBook-Air src % javac *.java
mens1s@Ahmet-MacBook-Air src % java main.java
Original String:                        'Hush, hush!' whispered the rushing wind.
Preprocessed String:            hush hush whispered the rushing wind
The original (unsorted) map:
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: p - Count: 1 - Words: [whispered]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: t - Count: 1 - Words: [the]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: g - Count: 1 - Words: [rushing]


The sorted map using selectionSort:


The sorted map:
Letter: p - Count: 1 - Words: [whispered]
Letter: t - Count: 1 - Words: [the]
Letter: g - Count: 1 - Words: [rushing]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
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 using selectionSort:


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]
mens1s@Ahmet-MacBook-Air src %
```

b. Insertion Sort

```
mens1s@Ahmet-MacBook-Air src % javac *.java
mens1s@Ahmet-MacBook-Air src % java main.java
Original String:                        'Hush, hush!' whispered the rushing wind.
Preprocessed String:            hush hush whispered the rushing wind
The original (unsorted) map:
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: p - Count: 1 - Words: [whispered]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: t - Count: 1 - Words: [the]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: g - Count: 1 - Words: [rushing]


The sorted map using Insertion Sort:


The sorted map:
Letter: p - Count: 1 - Words: [whispered]
Letter: t - Count: 1 - Words: [the]
Letter: g - Count: 1 - Words: [rushing]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
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 using Insertion Sort:


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]
mens1s@Ahmet-MacBook-Air src %
```

c. Bubble Sort

```
mens1s@Ahmet-MacBook-Air src % javac *.java
mens1s@Ahmet-MacBook-Air src % java main.java
Original String:                        'Hush, hush!' whispered the rushing wind.
Preprocessed String:            hush hush whispered the rushing wind
The original (unsorted) map:
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: p - Count: 1 - Words: [whispered]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: t - Count: 1 - Words: [the]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: g - Count: 1 - Words: [rushing]


The sorted map using Bubble Sort:


The sorted map:
Letter: p - Count: 1 - Words: [whispered]
Letter: t - Count: 1 - Words: [the]
Letter: g - Count: 1 - Words: [rushing]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
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 using Bubble Sort:


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]
mens1s@Ahmet-MacBook-Air src %
```

d. Quick Sort

```
mens1s@Ahmet-MacBook-Air src % java main.java
Original String:                        'Hush, hush!' whispered the rushing wind.
Preprocessed String:            hush hush whispered the rushing wind
The original (unsorted) map:
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: p - Count: 1 - Words: [whispered]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: t - Count: 1 - Words: [the]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: g - Count: 1 - Words: [rushing]


The sorted map using Quick Sort:


The sorted map:
Letter: g - Count: 1 - Words: [rushing]
Letter: t - Count: 1 - Words: [the]
Letter: p - Count: 1 - Words: [whispered]
Letter: n - Count: 2 - Words: [rushing, wind]
Letter: w - Count: 2 - Words: [whispered, wind]
Letter: d - Count: 2 - Words: [whispered, wind]
Letter: r - Count: 2 - Words: [whispered, rushing]
Letter: i - Count: 3 - Words: [whispered, rushing, wind]
Letter: u - Count: 3 - Words: [hush, hush, rushing]
Letter: e - Count: 3 - Words: [whispered, whispered, the]
Letter: s - Count: 4 - Words: [hush, hush, whispered, rushing]
Letter: h - Count: 7 - Words: [hush, hush, hush, hush, whispered, the, rushing]
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 using Quick Sort:


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]
mens1s@Ahmet-MacBook-Air src %
```