Gebze Technical University
CSE222 – HW6 Report

Name: Emre Oytun
Student ID: 200104004099

1) System Design and Method Working Explanations:

info class:

This class is used to keep the words and the total count of the words inside. This is used to keep the values in the myMap class.

Fields:

- int count: This keeps the total number of words inside. It is incremented when a new word is added.
- ArrayList<String> words: This keeps the list of the words.

Methods:

- public info():
This is the no-args constructor to initialize the fields with the initial values.

- public void push(String word):
This method is used to add a new word into the words array. It increments the count variable afterwards. It is needed to add new words.

- public int getCount():
This method returns the count value.

- public String toString():
This method converts the total count and all the words into a string in a nice way, and returns the resulting string.

myMap class:

This class keeps the Character-info pairs constructed from the given String, it also keeps the given String and the map size.
The given String is taken and mapped in the one type of the constructors such that the map keeps the letters and the words that they are contained inside.

Fields:

- LinkedHashMap<Character, info> map: This is the map that keeps our letter-info pairs that is constructed using the given string or adding new pairs directly.
- int mapSize: This keeps the size of the map.
- String str: This keeps the string that is preprocessed and mapped into the map.

Methods:

- public myMap():
This method initializes the map with the initial values. It initializes an empty map object and mapSize with 0.

- public myMap(String str):
This class initializes the myMap instance using the given string such that it first preprocess the string and builds the map using this string afterwards using the buildMap() method.
In preprocessing, it first converts all the letters to lower case using str.toLowerCase() method; then it deletes all the characters except letters using regex.
After preprocessing, it calls checkValidity(str) method to check if the preprocessed string has any characters or not. If it doesn't have any characters, then it prints a warning message.

- private boolean checkValidity(String str):
This method returns true, if the given preprocessed string is valid meaning it has at least 1 character.

- private void buildMap():
This method builds map using the str.
It first splits the str into tokens using str.split(" ") method. Then it traverses through all words and all the letters inside the words each time. It checks if this letter is mapped before, if it does then it uses it, otherwise it creates a new info instance and adds the new word into that info instance. Finally, it puts the resulting letter-info pair to the map.

- public void put(Character ch, info chInfo):
This method adds or updates the Character-info pair. If the character is not mapped before, then it increments the map size. It does all of these operations using the map object, so it wraps the map's method in a way.

- public info get(Character ch):
This method returns the info instance related to the given character.
It does this operation using the map object.
It returns null, if there is no such character inside the map.

- public int size():
This method returns the size of the map using the mapSize variable.

- public Set<Character> keySet():
This method returns the key set of the map inside using the map.keySet() method.
This is needed to initialize the auxilary array inside the merge sort.

- public void printMap():
This method prints the map in a nice way as described in the homework screenshots.

mergeSort class:

This class sorts the given map using the merge sort algorithm.

Fields:

- myMap originalMap: This always keeps the original map as it is.
- myMap sortedMap: This keeps the sorted map.
- Character[] aux: This keeps the auxilary array to make sorting properly.

Methods:

- public mergeSort(myMap originalMap):
This is the contructor that initializes an instance of the mergeSort class.
Firstly, it assign the given map to the originalMap variable and initializes the sortedMap as an empty map.
Secondly, it initializes the auxilary array using the originalMap such that auxilary array contains the list of keys in the same order as the map using the initializeAux() method.
Then, it calls the sort(0, aux.length-1) method to sort the auxilary using mergesort algorithm.
Finally, it calls setSortedMap() method to set the sortedMap variable using the auxilary array and the originalMap.

- private void initializeAux():
This method initializes the aux as the array of characters with the size of the original map's keys in the map. Then assign the keys to this auxilary array in the same order as the originalMap.

- private void sort(int firstIdx, int lastIdx):
This is a recursive method to sort the map/auxilary array using the mergesort algorithm. It splits the current sub-array into two sub-arrays each time, then merges the resulting arrays. Splitting is done by using the indexes such that the first sub-array is from firstIdx to midIdx((firstIdx+lastIdx) / 2) and the second sub-array is from midIdx+1 to lastIdx. Merging is done by calling merge(firstIdx, midIdx, lastIdx) method.

- private void merge(int firstIdx, int midIdx, int lastIdx):
This method merges the two sub-arrays into one sorted array with using an extra space for the newly created array.
Firstly, it initializes the resulting array with the size of total length(lastIdx-firstIdx+1).
Secondly, it goes through the first and second sub-arrays elements. Until one of the sub-arrays is done, it checks if the left element is smaller than right element; if it is, it puts the left element to the resulting array, otherwise it puts the right element.
Finally, it puts the remaining elements to the resulting array if there are any and puts the resulting array back to the auxilary array.

- private void setSortedMap():
This method sets the sortedMap variable.
It iterates through the auxilary array. Each time it gets the character and the info that is responded to that character from the originalMap and it puts this pair to the sortedMap.

- public void printMaps():
This methods prints the originalMap and the sortedMap using the myMap's printMap() method.

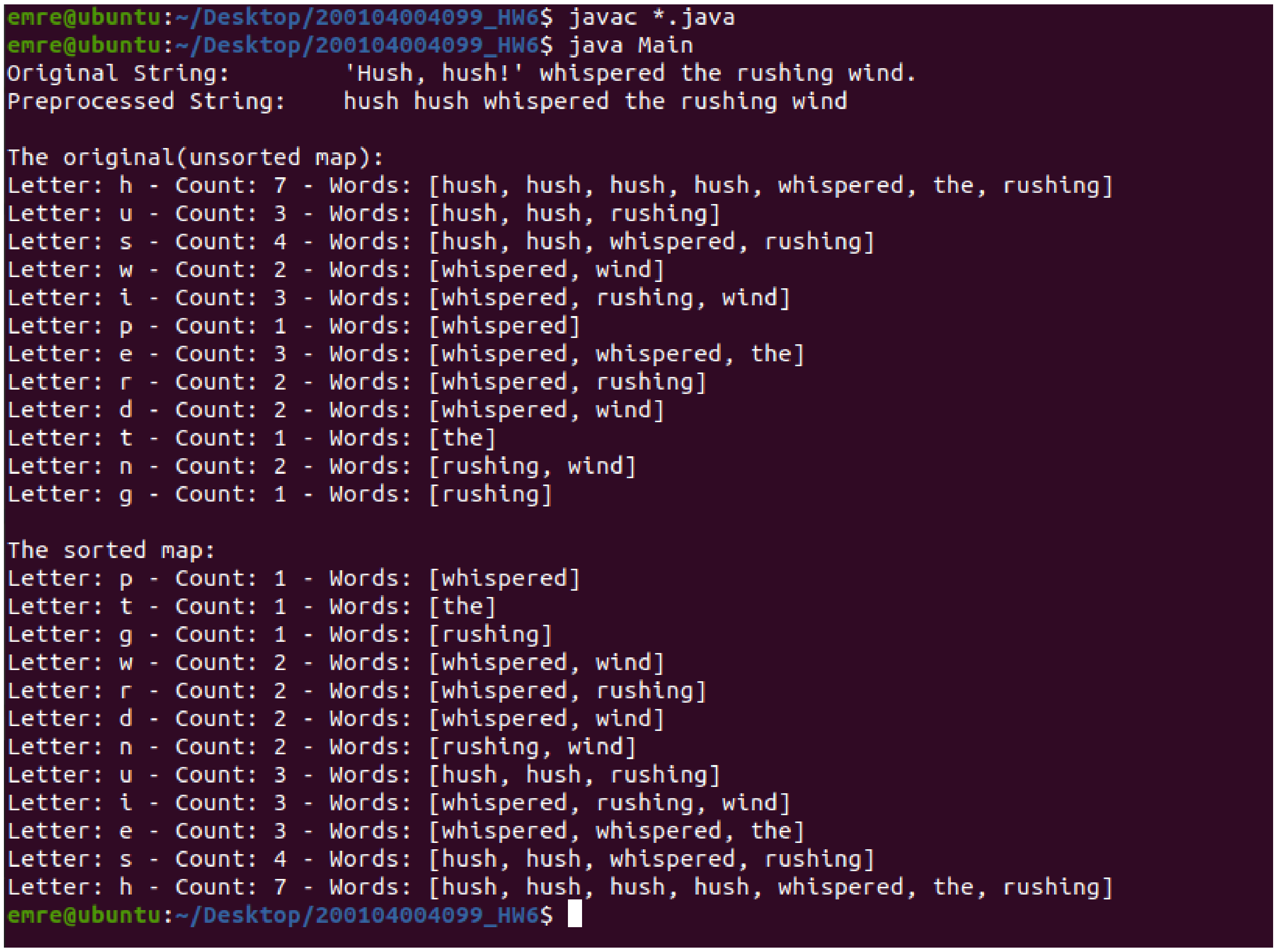
Main class:

This class contains the main method that is used to test the preprocessing, constructing the map and sorting the map using the mergesort algorithm.
It creates an original map firstly with a string, then it creates a new mergeSort instance.
Finally, it prints the resulting maps.

Note: To test the algorithms with different String inputs, you can change String in the constructor of myMap in initializing the originalMap.

You can find a screenshot below that shows my program is working on my linux machine properly.

2) Screenshot of The Main Program:



```
emre@ubuntu:~/Desktop/200104004099_HW6$ javac *.java
emre@ubuntu:~/Desktop/200104004099_HW6$ java Main
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:
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]
emre@ubuntu:~/Desktop/200104004099_HW6$
```