# GTU Department of Computer Engineering

# CSE 222/505 – Spring 2023

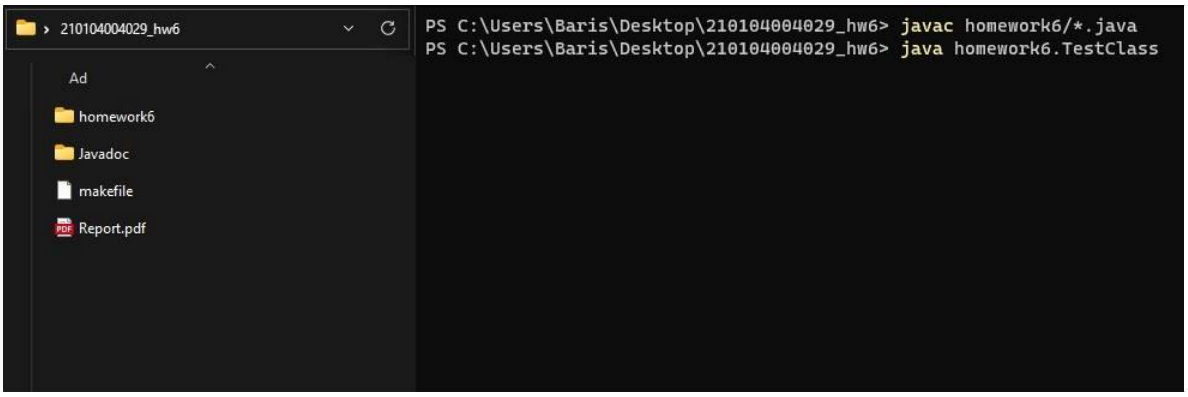
# Homework 6 Report

**Barış Batuhan Bolat**

**210104004029**

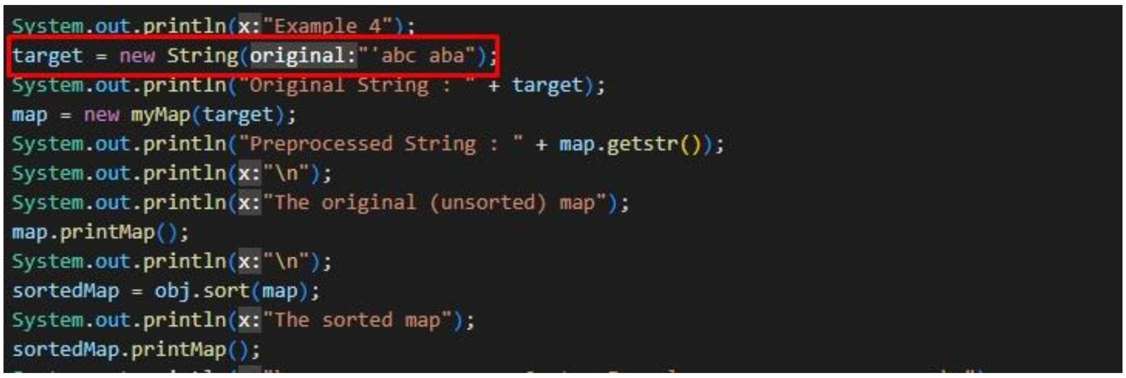
## 1. Program Usage

- Compile and run(You can use makefile too)



```
PS C:\Users\Baris\Desktop\210104004029_hw6> javac homework6/*.java
PS C:\Users\Baris\Desktop\210104004029_hw6> java homework6.TestClass
```

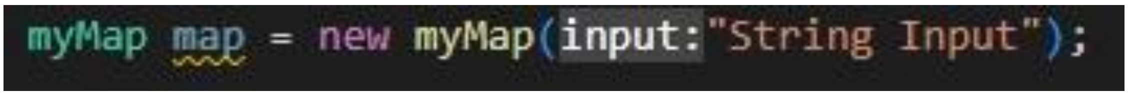
- In the TestClass.java file, there are 4 examples in the assignment pdf and two extra examples. The desired string can be entered by changing the marked place in the picture below.



```
System.out.println(x:"Example 4");
target = new String(original:"'abc aba");
System.out.println("Original String : " + target);
map = new myMap(target);
System.out.println("Preprocessed String : " + map.getstr());
System.out.println(x:"\n");
System.out.println(x:"The original (unsorted) map");
map.printMap();
System.out.println(x:"\n");
sortedMap = obj.sort(map);
System.out.println(x:"The sorted map");
sortedMap.printMap();
```

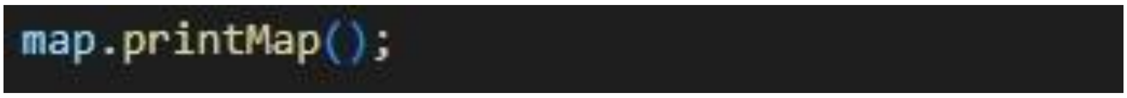
## 2. Method Usage

- Creating map with input string



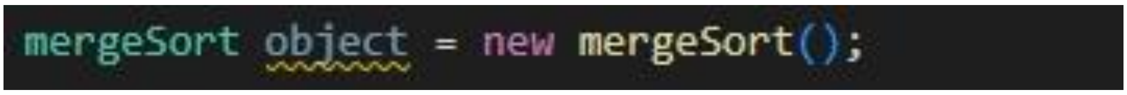
```
myMap map = new myMap(input:"String Input");
```

- Printing the map as specified in the pdf.



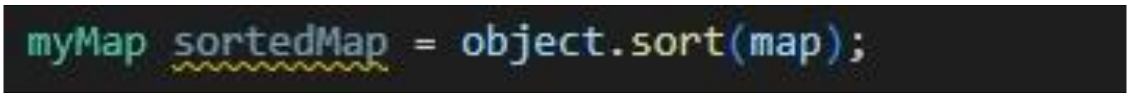
```
map.printMap();
```

- Creating “mergeSort” object for merge sort operation.



```
mergeSort object = new mergeSort();
```

- Sorting old map with “mergeSort” object and assign it to new “myMap” object



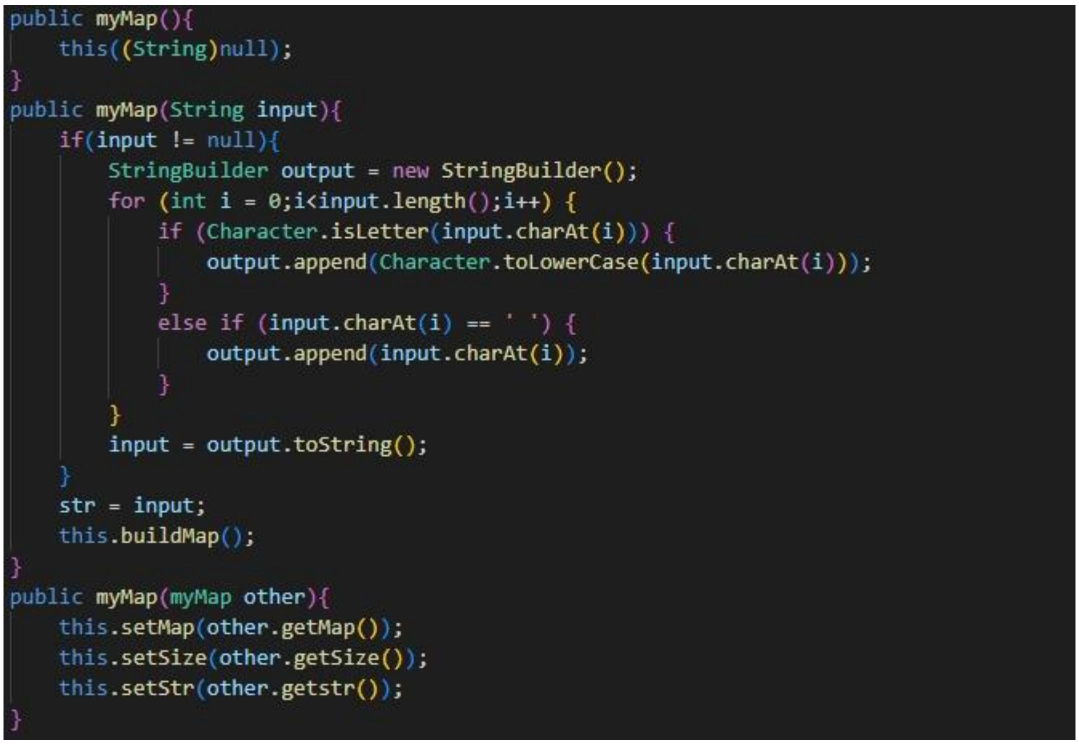
```
myMap sortedMap = object.sort(map);
```

## 3. Method Implemetations

### A. myMap Class

#### a) Constructor

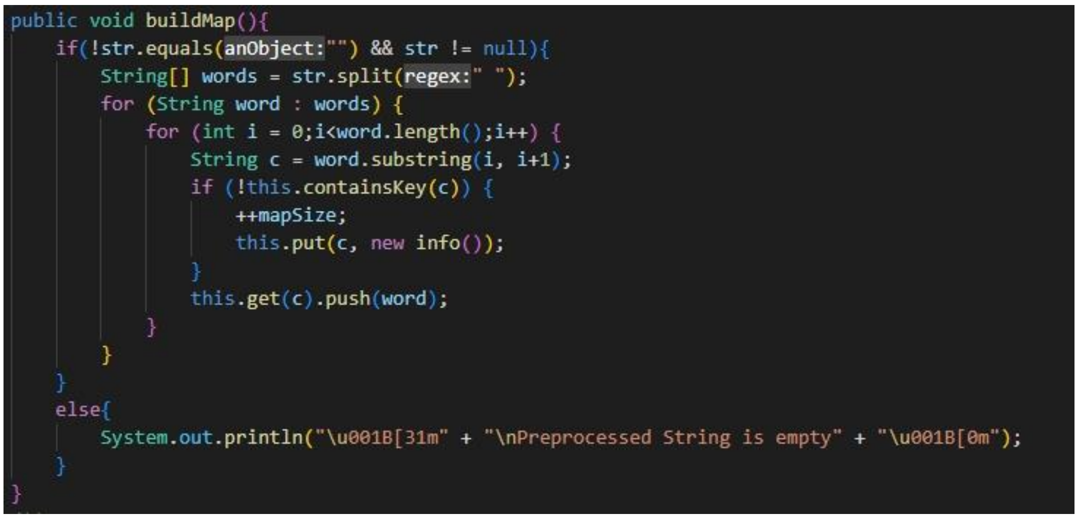
- This class has three constructor Default constructor, Parameterized Constructor and Copy Constructor
- Parameterized constructor takes a String input, removes all non-letter characters (except spaces), converts all letters to lowercase, and assigns the resulting String to the "str" variable of the myMap object.



```java
public myMap(){
    this((String)null);
}
public myMap(String input){
    if(input != null){
        StringBuilder output = new StringBuilder();
        for (int i = 0;i<input.length();i++) {
            if (Character.isLetter(input.charAt(i))) {
                output.append(Character.toLowerCase(input.charAt(i)));
            }
            else if (input.charAt(i) == ' ') {
                output.append(input.charAt(i));
            }
        }
        input = output.toString();
    }
    str = input;
    this.buildMap();
}
public myMap(myMap other){
    this.setMap(other.getMap());
    this.setSize(other.getSize());
    this.setStr(other.getstr());
}
```

## b) buildMap()

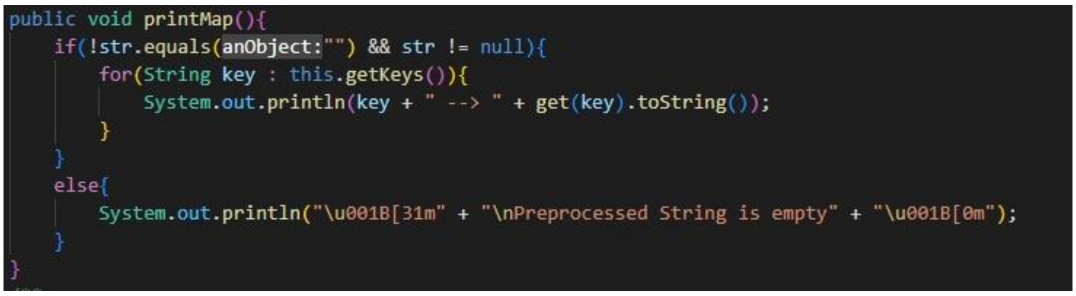
- This method builds a mapping between each unique character in the input str and a list of words that contain that character. The method stores this mapping in the map instance variable. The mapSize instance variable is also updated.



```
public void buildMap(){
    if(!str.equals(anObject:"") && str != null){
        String[] words = str.split(regex:" ");
        for (String word : words) {
            for (int i = 0;i<word.length();i++) {
                String c = word.substring(i, i+1);
                if (!this.containsKey(c)) {
                    ++mapSize;
                    this.put(c, new info());
                }
                this.get(c).push(word);
            }
        }
    }
    else{
        System.out.println("\u001B[31m" + "\nPreprocessed String is empty" + "\u001B[0m");
    }
}
```

## c) printMap()

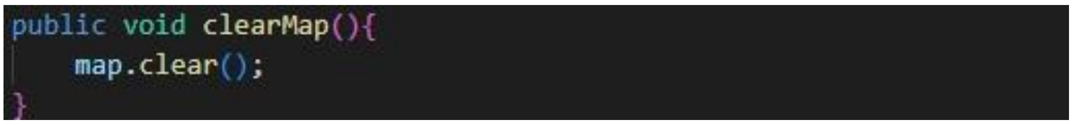
- This method prints map



```
public void printMap(){
    if(!str.equals(anObject:"") && str != null){
        for(String key : this.getKeys()){
            System.out.println(key + " --> " + get(key).toString());
        }
    }
    else{
        System.out.println("\u001B[31m" + "\nPreprocessed String is empty" + "\u001B[0m");
    }
}
```

## d) clearMap()

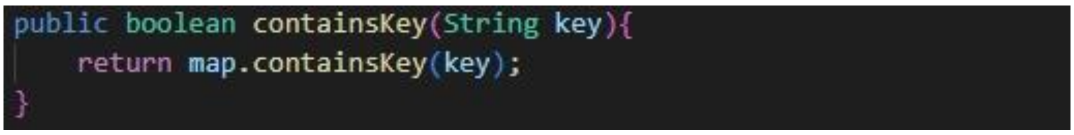
- Clears map with LinkedHashMap classes clear method



```
public void clearMap(){
    map.clear();
}
```

## e) containsKey()

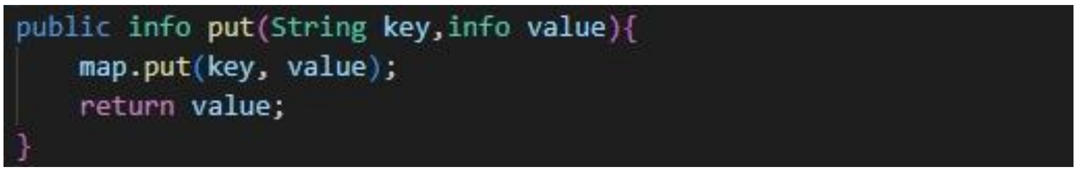
- This method checks whether the map private variable contains a key-value pair with LinkedHashMap classes containsKey method.



```
public boolean containsKey(String key){
    return map.containsKey(key);
}
```

### f) put()

- This method adds key value pair to map with LinkedHashMap classes put method.



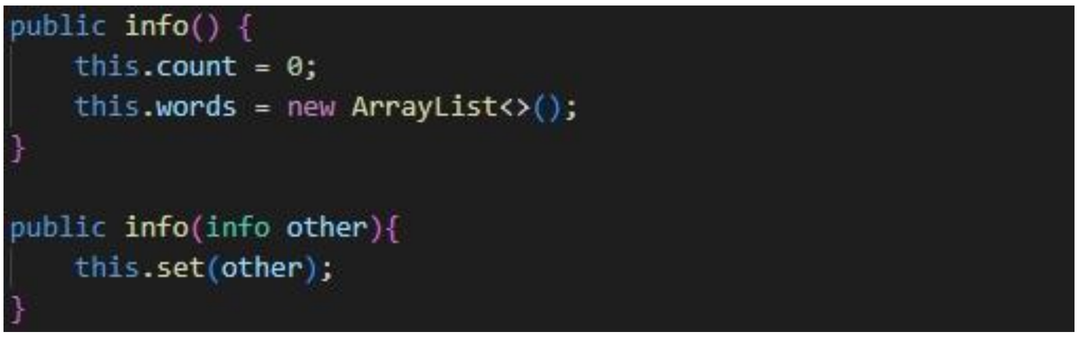
```java
public info put(String key,info value){
    map.put(key, value);
    return value;
}
```

- This class has get and set methods for private fields.

## B. info Class

### a) Constructor

- This class had two constructor. Default constructor initialize count and words list. Copy constructor works with set method of this class.

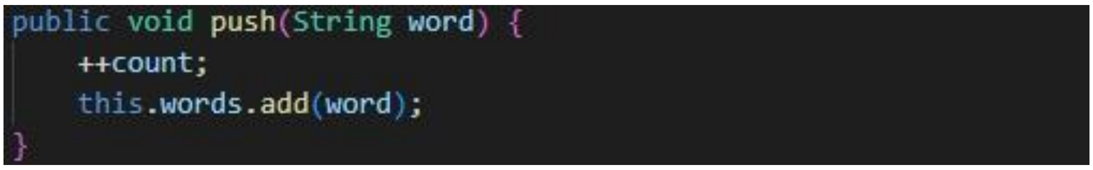


```java
public info() {
    this.count = 0;
    this.words = new ArrayList<>();
}

public info(info other){
    this.set(other);
}
```

### b) push()

- This method increase count of  maps key occurence and add word to “words” list by using ArrayList classes add method.

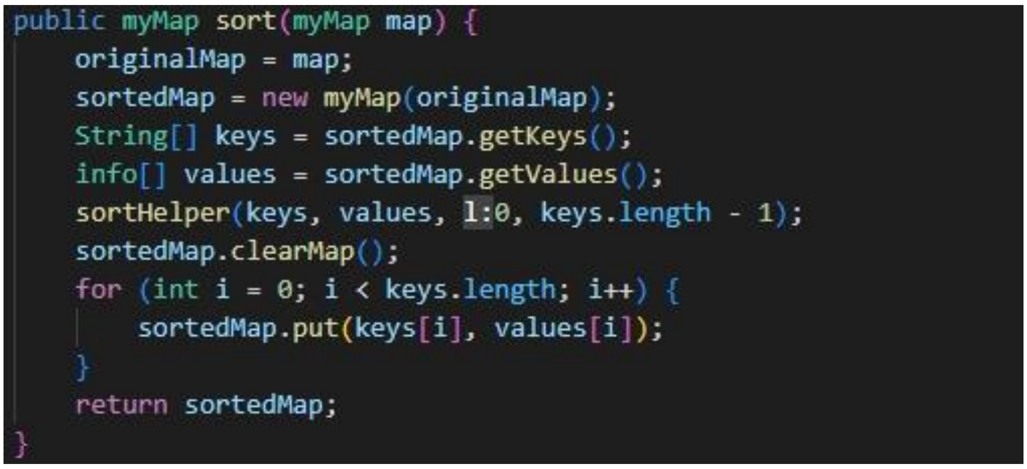


```java
public void push(String word) {
    ++count;
    this.words.add(word);
}
```

- This class has get and set methods for private fields.

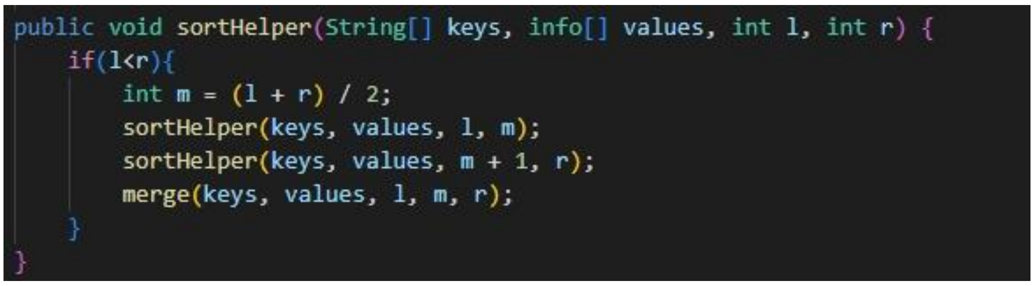
## C. mergeSort Class

- This class has only one working method . The other two methods helps that methods work.
- This primary method takes unsorted map and assign it to “originalMap” and create sortedMap by using originalMap. After taking keys and values of map sends it to the helper method to sort it. After sorting operation complete this method clears the map and add the sorted keys and values to it.



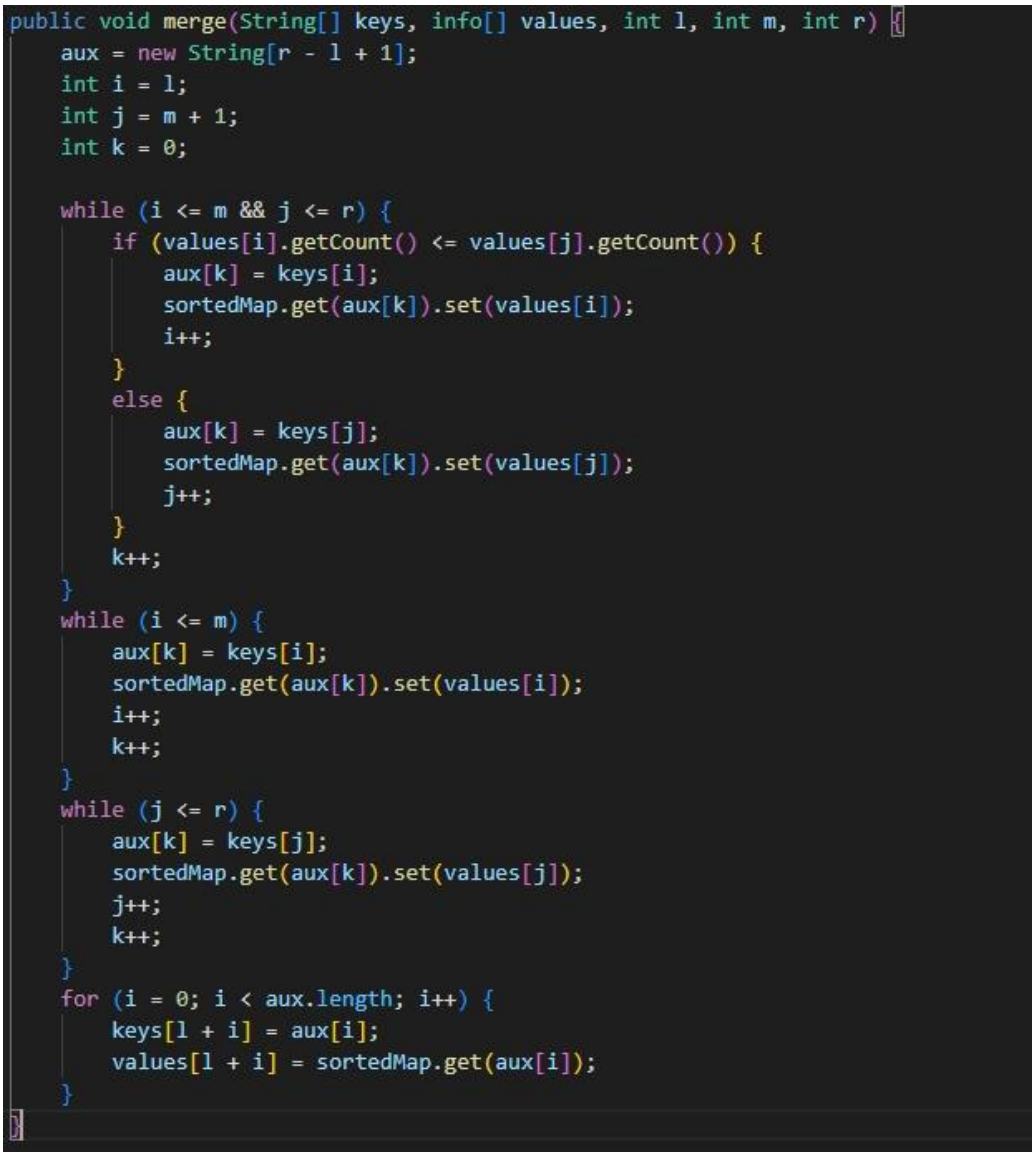
```java
public myMap sort(myMap map) {
    originalMap = map;
    sortedMap = new myMap(originalMap);
    String[] keys = sortedMap.getKeys();
    info[] values = sortedMap.getValues();
    sortHelper(keys, values, l:0, keys.length - 1);
    sortedMap.clearMap();
    for (int i = 0; i < keys.length; i++) {
        sortedMap.put(keys[i], values[i]);
    }
    return sortedMap;
}
```

- The method recursively divides the subarray into two halves until each half contains only one element. Then it uses the “merge” method to merge the two halves of the subarray into a single sorted subarray.



```java
public void sortHelper(String[] keys, info[] values, int l, int r) {
    if(l<r){
        int m = (l + r) / 2;
        sortHelper(keys, values, l, m);
        sortHelper(keys, values, m + 1, r);
        merge(keys, values, l, m, r);
    }
}
```

- This method initialize a private temporary array called aux to store the sorted keys, and then compares the values of the corresponding info objects at each index of the subarray, starting from the left and right halves, and adding the smaller one to the aux array. After all the elements from one half have been added to the aux array, the remaining elements from the other half are added to the end of the aux array. Finally, the sorted keys and values are copied back to the original arrays.



```java
public void merge(String[] keys, info[] values, int l, int m, int r) {
    aux = new String[r - l + 1];
    int i = l;
    int j = m + 1;
    int k = 0;

    while (i <= m && j <= r) {
        if (values[i].getCount() <= values[j].getCount()) {
            aux[k] = keys[i];
            sortedMap.get(aux[k]).set(values[i]);
            i++;
        }
        else {
            aux[k] = keys[j];
            sortedMap.get(aux[k]).set(values[j]);
            j++;
        }
        k++;
    }
    while (i <= m) {
        aux[k] = keys[i];
        sortedMap.get(aux[k]).set(values[i]);
        i++;
        k++;
    }
    while (j <= r) {
        aux[k] = keys[j];
        sortedMap.get(aux[k]).set(values[j]);
        j++;
        k++;
    }
    for (i = 0; i < aux.length; i++) {
        keys[l + i] = aux[i];
        values[l + i] = sortedMap.get(aux[i]);
    }
}
```