GIT Department of Computer Engineering

CSE 222/505 - Spring 2023 Homework #7 Report

Pelin Erdoğan

210104004266

1. PROBLEM SOLUTION APPROACH

Problem solution of Creation of mymap:

Mymap class has string str,linkedHashMap map and integer mapsize as a property of class. Constructor that takes string as a parameter arranges string and creates map. It splits string by its spaces and adds letters in words one by one. If a letter is already existing in map, it just pushes the word into info's arraylist of words.

Problem solution of Merge sort:

Merge Sort methods:

Void sort(): calls divideandmerge method in a proper way and initializes sortedmap with mymap constructor that takes string array and original map to create my map object.

Void divideandmerge(int left,int right): Method is private because I wanted to be only called by sort method. It divides the map into little pieces with than calls merge function and it is a recursive function. It calls itself until left is bigger or equal to right.

Void merge(int left, int mid, int right): Method is private because I wanted to be only called by divideandmerge method. creates 4 other array for left and right sides to merge. 2 integer arrays 2 string arrays. Puts the elements we want to merge inside them and then compares left ones with right ones and whichever is smaller assigns that one into aux and increases necessary variables.after comparison if there are any elements in right or left map that is not putted in place in original aux array. Puts them after the ones added before.

Problem solution of Bubble sort:

The sort method of class has nested for loops. Method compares the counts of info objects belong to the key I store in aux element. If next elements count value is smaller I swap them. When the first for loop ends if any element hasn't swapped during loop sorting ends. I used a flag to do this. After loops are over method initializes sortedmap with mymap constructor that takes string array and original map to create my map object.

Problem solution of Insertion sort:

The sort method of class has a while loop inside of a for loop. In the for loop I store current key in variable current and then I find the value of that key from map and store count (info object's property in currentcount. In while loop I move the current element untill I find a smaller one. After sort of aux array is over method initializes sortedmap with mymap constructor that takes string array and original map to create my map object.

Problem solution of Selection sort:

The sort method of class has nested for loops.Method finds the smallest element in the searched area in map and swaps it with the element that was in its place.

After sort of aux array is over method initializes sortedmap with mymap constructor that takes string array and original map to create my map object.

Problem solution of Quick sort:

Quick sort methods:

Void sort(int low,int high):

In this recursive method first we find the pivot using patriation method and then call function again with dividing the map by pivot point.

Int patriation (int low, int high):

Method is private because I wanted to be only called by sort method. In the beginning of method program chooses the last element as pivot and it has variable i which start as low-1. Method has a for loop.In the for loop program compares the elements with the pivot if they are smaller than pivot program increases i and swaps the element smaller than pivot with the one in index i.After the for loop program swaps the pivot with the element in index i + 1 and then returns i + 1.

2. TIME ANALYSİS

Normally efficiencies of sorting algorithms are like this

Sort	Best case	Worst case	Average case
Quick	nlogn	n^2	nlogn
Bubble	n	n^2	n^2
Selection	n^2	n^2	n^2
Insertion	n	n^2	n^2
Merge	nlogn	nlogn	nlogn

For most of the algorithms the best case is when the array is already sorted. Because most of them start sorting from left to right. So, when it is sorted bubble,merge,selection and insertion sorts don't make any swap operations and actually when array is already sorted.

For the bubble and insertion sort time complexity changes in best case because in bubble sort flag causes the loop to break since any element hasn't changes during loop and while loop never gets used in best case.

But for the quick sort sorted array is actually become the worst case because of pivot choices. To have the best case with quick sort pivot should be the median so for an array

with 7 elements sort would be like this. (red ones are pivot, grey parts are used to show recursion)

1 3 2 5 7 6 4 -> 1 3 2 4 5 7 6 ->1 2 3 4 5 6 7

The worst case is reverse ordered for all algorithms. Because it causes the most swap operations and recursions. Altough in merge sort cases don't change anything because in every case time complexity is nlogn. And selection sort is n^2 for every case but unlike merge sort amount of swap operations depends on cases.

For quick sort worse case is reverse sorted or sorted because in both cases for loop gets extended and recursion lasts longer.

The average case is random order for all algorithms.

These are measured sorting times

Sort	Best case	Worst case	Average case
Quick	16800	58300	20600
Bubble	11200	34400	13500
Selection	19600	36600	20700
Insertion	11700	29300	15800
Merge	27300	30500	28400

Which algorithm is faster in which case?

- For quick sort best and average are really close since they are both nlogn but worst is approximately really high because it is n^2.
- For bubble sort the best case time complexity is n. That is why it is the fastest one in best case. Average case and worst case are n^2 but since worst case has a lot more swap operations even though they have same time complexity worst case is slower.
- For selection sort time complexity is n^2 for all cases. So, the time difference is there because of the amount of swap operations.
- For insertion sort the best case time complexity is n. That is why it is the second fastest one in best case. Average case and worst case are n^2 but since worst case has a lot more swap operations, even though they have same time complexity worst case is slower.
- For merge sort time complexity is nlogn for all cases. It is a stable algorithm. So, the time difference is there because of the number of swap operations.

Which one is the best one to use in which cases

Quick sort

In my opinion using quick sort for small data sets is not so useful. For small data it even takes more time than bubble sort. Normally it is much more efficient than other sorting methods(selection,insertion,bubble) but that difference is more noticeable for big data sets. If I compare this algorithm with merge sort algorithm I can say that if you know data you want to sort is really random you should use quick sort and it takes less memory than merge sort. You need to keep the order of elements you can't use quick sort.

Merge sort

Just like quick sort using merge sort for small data sets is not so useful. Its efficiency would show itself more if I tried with a bigger array. The best thing about merge sort is that it is almost the same for every case. This algorithm's time complexity doesn't change no matter what. It can be used if you will not have any idea what array is like. Because cases don't change its efficiency. And if you want to sort elements and keep their order you cant use quick sort. So you should use merge sort.

Selection sort

This algorithm is not so efficient but works well for small data sets and it is easy to write. And if you need to just find top 5 you can do that easily with selection sort without sorting the whole array. if you need to keep the order of elementsthis algorithm is not useful for you.

Bubble sort

This algorithm is not so efficient too but for an array nearly sorted it works really well and when you use flag to see if array is sorted it gets more efficient than selection and insertion sort.If you need to keep the order of elements with same amount you can use bubble sort.

Insertion sort

This algorithm is not so efficient too but for small data set that is practically sorted it is better than selection sort.If you need to keep the order of elements you can use insertion sort.

Which algorithms keeps input order?

As I said before merge sort, bubble sort and insertion sort keep the order. Quick sort and selection sort don’t keep the order.

```
Original map
"a" Count: 4 Words array: [ aaaa ,aaaa ,aaaa ,aaaa ]
"b" Count: 3 Words array: [ bbb ,bbb ,bbb ]
"c" Count: 4 Words array: [ cccc ,cccc ,cccc ,cccc ]
"d" Count: 3 Words array: [ ddd ,ddd ,ddd ]
"f" Count: 6 Words array: [ ffffff ,ffffff ,ffffff ,ffffff ,ffffff ,ffffff ]
"h" Count: 2 Words array: [ hh ,hh ]
"t" Count: 5 Words array: [ ttttt ,ttttt ,ttttt ,ttttt ,ttttt ]
"y" Count: 6 Words array: [ yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ]

Sorted selection:
"h" Count: 2 Words array: [ hh ,hh ]
"b" Count: 3 Words array: [ bbb ,bbb ,bbb ]
"d" Count: 3 Words array: [ ddd ,ddd ,ddd ]
"c" Count: 4 Words array: [ cccc ,cccc ,cccc ,cccc ]
"a" Count: 4 Words array: [ aaaa ,aaaa ,aaaa ,aaaa ]
"t" Count: 5 Words array: [ ttttt ,ttttt ,ttttt ,ttttt ,ttttt ]
"f" Count: 6 Words array: [ ffffff ,ffffff ,ffffff ,ffffff ,ffffff ,ffffff ]
"y" Count: 6 Words array: [ yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ]

Sorted insertion:
"h" Count: 2 Words array: [ hh ,hh ]
"b" Count: 3 Words array: [ bbb ,bbb ,bbb ]
"d" Count: 3 Words array: [ ddd ,ddd ,ddd ]
"a" Count: 4 Words array: [ aaaa ,aaaa ,aaaa ,aaaa ]
"c" Count: 4 Words array: [ cccc ,cccc ,cccc ,cccc ]
"t" Count: 5 Words array: [ ttttt ,ttttt ,ttttt ,ttttt ,ttttt ]
"f" Count: 6 Words array: [ ffffff ,ffffff ,ffffff ,ffffff ,ffffff ,ffffff ]
"y" Count: 6 Words array: [ yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ]

Sorted Bubble:
```

```
Sorted Bubble:
"h" Count: 2 Words array: [ hh ,hh ]
"b" Count: 3 Words array: [ bbb ,bbb ,bbb ]
"d" Count: 3 Words array: [ ddd ,ddd ,ddd ]
"a" Count: 4 Words array: [ aaaa ,aaaa ,aaaa ,aaaa ]
"c" Count: 4 Words array: [ cccc ,cccc ,cccc ,cccc ]
"t" Count: 5 Words array: [ ttttt ,ttttt ,ttttt ,ttttt ,ttttt ]
"f" Count: 6 Words array: [ ffffff ,ffffff ,ffffff ,ffffff ,ffffff ,ffffff ]
"y" Count: 6 Words array: [ yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ]

Sorted quick:
"h" Count: 2 Words array: [ hh ,hh ]
"d" Count: 3 Words array: [ ddd ,ddd ,ddd ]
"b" Count: 3 Words array: [ bbb ,bbb ,bbb ]
"a" Count: 4 Words array: [ aaaa ,aaaa ,aaaa ,aaaa ]
"c" Count: 4 Words array: [ cccc ,cccc ,cccc ,cccc ]
"t" Count: 5 Words array: [ ttttt ,ttttt ,ttttt ,ttttt ,ttttt ]
"y" Count: 6 Words array: [ yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ]
"f" Count: 6 Words array: [ ffffff ,ffffff ,ffffff ,ffffff ,ffffff ,ffffff ]

Sorted merge:
"h" Count: 2 Words array: [ hh ,hh ]
"b" Count: 3 Words array: [ bbb ,bbb ,bbb ]
"d" Count: 3 Words array: [ ddd ,ddd ,ddd ]
"a" Count: 4 Words array: [ aaaa ,aaaa ,aaaa ,aaaa ]
"c" Count: 4 Words array: [ cccc ,cccc ,cccc ,cccc ]
"t" Count: 5 Words array: [ ttttt ,ttttt ,ttttt ,ttttt ,ttttt ]
"f" Count: 6 Words array: [ ffffff ,ffffff ,ffffff ,ffffff ,ffffff ,ffffff ]
"y" Count: 6 Words array: [ yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ,yyyyyy ]
```