Homework 7 Report

Batuhan BASAK

1 Time Complexities of MyMap and Info Classes

1.1 Time Complexity of Info Class

1.1.1 The Constructor

The Constructor method of Info class sets count value and intanciates an array list. Since both steps are takes $O(1)$, thus the time complexity of the Constructor method takes $O(1)$.

1.1.2 The getCount and setCount Methods

Both methods takes $O(1)$.

1.1.3 The push Method

The push method puts given word to underlying array list by using add method of array list. The time complexity of add method of array list is amortized $O(1)$. Therefore the time complexity of push method is amortized $O(1)$.

1.1.4 The toString Method

The toString method basically returns a string. But the string concatinates a string and the return value of toString method of the array list whose time complexity is $O(n)$. Since $T(n) = O(1) + O(n) = O(n)$, therefore the time complexity of toString method is $O(n)$.

1.2 Time Complexity of MyMap Class

1.2.1 The Constructor Methods

The default Constructor instantiate the underlying linked hash map that time complexity is $O(1)$. The second Constructor method takes a string, preprocesses the string and fills the map by invoking fillMap method. The preprocessing and filling map take $O(n)$. Therefore the second Constructor method takes $O(n)$.

1.2.2 The preprocessString Method

The preprocessString method iterates each character in given string. The iteration takes $O(n)$. Therefore preprocessString takes $O(n)$.

1.2.3 The fillMap Method

The fillMap methods fills the words array by iterating each word in string. The iteration takes $O(n)$. Therefore the time complexity of fillMap method is $O(n)$.

1.2.4 The toString Method

The toString method iterates each key value in key set of underlying linked hash map. And at each iteration it calls the toString method of pair Info data. The toString method of Info class takes O(n). The number of calling of toString method of Info is $O(n)$. So the time function is $T(n) = O(n) \cdot O(n) = O(n^2)$. Therefore the time complexity of toString method is $O(n^2)$.

1.2.5 The getMap Method

The getMap method returns the underlying linked hash map. The time complexity of getMap method is $O(1)$.

1.2.6 The put and get Methods

The put and get method calls the put and get methods of underlying linked hash map list by using delegation respectively. Since the time complexities of put and get methods of linked hash map is $O(1)$, the time complexities of put and get methods are $O(1)$.

2 The Answers of Second Part's Questions

2.1 Part A

2.1.1 Merge Sort

Before and after sorting the whole sorting algorithms implemented in this homework contains piece of codes that takes $O(n)$. So the time complexities of algorithms are equal to $O(n)+$ sorting part.
The merge sort algorithm divides the array into two halves. This process takes $O(n)$. Then calls merge sort algorithm itself recursively for left and right arrays. Finally calls merge method to merge the left and right. Since the get method of MyMap is $O(1)$, the merge method runs $O(n)$. The number of recursive calls is equal to logarithm of size of array, $O(logn)$. Since each recursive calls take $O(n)$, time takes filling left and right arrays, the time complexity of merge sort is $O(n) \cdot O(logn) = O(n \cdot logn)$. Whether array is already sorted or not, it fills the left and right halves and calls the merge sort array recursively for $logn$ times. Therefore the time complexity for all cases are equal to $O(n \cdot logn)$.

2.1.2 Selection Sort

The selection sort algorithm iterates array from 0 to n-1 and each iteration finds the minimum value in array from current position in loop to end of the array. So the inner loop runs $O(n)$ and therefore the outer loop runs $O(n) \cdot O(n) = O(n^2)$. Therefore the time complexity of selection sort is $O(n^2)$. To find the minimum value in inner loop, whether array is sorted, it will run $O(n)$. So the worst case, best case, and average case alike are $O(n^2)$.

2.1.3 Insertion Sort

The insertion sort iterates each item from second item to last item in array and inserts the current item to the correct position in array. If the array is already sorted, finding the correct position takes $O(1)$. Therefore the time complexity for best case is $O(n)$. For the worst and average cases, finding the correct position takes $O(n)$. Therefore the time complexity for worst and average cases are $O(n^2)$. The time complexity of insertion sort is

$$T(n) = T_{best}(n) + T_{worst}(n) = O(n) + O(n^2) = O(n^2).$$

2.1.4 Bubble Sort

The bubble sort pushes the biggest item to last position by iterating arrays. The time complexity of this is $O(n)$. The outer loop makes this process called for n times, where n is the length of array. So the time complexity of bubble sort is $O(n) \cdot O(n) = O(n^2)$. Since whether array is sorted or not, the algorithm traverse the array at each iteration, the time complexity of bubble sort for best, worst, and average case scenarios are equal to $O(n^2)$.

2.1.5 Quick Sort

The quick sort algorithm traverse array, rearrange values respect to pivot value, divides array into to halves and calls them recursively. Whether it is sorted or not, the traversing part takes $O(n)$. The quick sort function calls the recursive function $logn$ times. Therefore the time complexity of quick sort algorithm, whether array is sorted or not, is equal $O(n \cdot logn)$. The best and average cases are equal to $O(n \cdot logn)$. The first case happens when a pivot is selected such that one half array is empty and the other half array is filled with all elements in array, which makes the count of recursive calls n instead of $logn$. And makes the time complexity $O(n^2)$.

2.2 Part B

Inputs	"Buzzing bees buzz."	"a bb ccc dddd ddddd"	"dddd ccc bb a"
Merge Sort(ns)	607,750	71,583	81,250
Selection Sort(ns)	214,000	259,208	8,916
Insertion Sort(ns)	294,834	38,583	7,625
Bubble Sort(ns)	701,000	9,291	8,875
Quick Sort(ns)	336,208	266,916	7,375

2.3 Part C

Between $O(n^2)$ algorithms, since selection sort and bubble sort is $O(n^2)$ for all cases and insertion sort is $O(n)$ for best cases, choosing insertion sort is ideal. The merge sort and quick sort are $O(n \cdot logn)$, which is better time complexity than $O(n^2)$ algorithms. The merge sort algorithms run $O(n \cdot logn$ for all cases but takes too much space during creating subarrays. On the other hand quick sort is take less space relatively than merge sort but for the worst case it runs $O(n^2)$ rather than $O(n \cdot logn)$. For large number of inputs, to save more spaces, using Quick sort algorithm more efficient.
In practice, for the second string "a bb ccc dddd dddd" which is already sorted, the bubble sort performed better than quick and merge sort algorithms. But for the input "dddd ccc bb a" which is reversed order, quick sort algorithm performed best as expected. For all inputs, the merge sort algorithm performed the worst performance among sorting algorithms. Since the insertion sort performed second best time for all inputs, the insertion sort it the most efficient one to use for this homework respect to the empirical results.

2.4 Part D

All five algorithms alike preserve the order of insertion for the same number of counts without any exception.