ISSIFU ALHASSAN – DSC OCTOBER PROGRAMMING CHALLENGE

I used java for easy and medium and python for hard.

Easy (Apple and Orange)

Data Structures used: none

I used one for loop that runs from 0 to ***m+n*** where m is the number of apples and n is the number of oranges. Inside the for loop, I used the ***nextInt()*** method of scanner to get the next integer. If the loop variable is less than m, then the integer is an apple distance, otherwise, it is an orange distance. For all apple distances, I check to see if it falls on the house. The I increment the counter for number of apples on the house. I did the same for oranges. To determine if a fruit falls on the house or not, I add the location of its tree to the integer read and check to see if the result falls within the house range. I think my approach is the best because I used no data structures since they take more space. I also made my computation alongside reading the input which makes my algorithm O(n) where n is the number of inputs.

Medium (Fraudulent Activity Notification)

Data Structures used: Hashtable, Array

I initialized a ***hashtable*** to contain the expenditure for trailing days and an array to keep its sorted version. I read the first n trailing days' expenditure into both the array and the ***hashtable***, sort the array and make the ***hashtable*** contain the original arrangement. For the ***hashtable***, the keys are the position of the numbers (0,1,2,3…n) as though they were in an array and the values are the expenditure. Since the array is sorted, I find the middle term or the average of the two middle terms to get the median. For subsequent trailing days, I replace the oldest value in the ***hashtable*** with the newer value and use binary search to find the location of the replaced integer in the array since the array is sorted. When I find it, I delete it and insert the new value at the right location to keep the array sorted for easy finding of the median. I think my approach is best because I only sort the array once as opposed to sorting for every set of trailing days. I also used a ***hashtable*** because both insert and delete have a time complexity of ***O(1).*** Binary search reduces my search in the array to ***O(logn***) and an overall time complexity of ***O(n²)*** though it is approximately between ***O(nlogn)*** and $\boldsymbol{O(\frac{1}{2}n^2)}$

Hard (Build a Palindrome)

Data structures used: list, dictionary

First, I created a dictionary of all the letters in "string **a**", as well as "string **b**". I used the dictionaries to check if **a** and **b** have any letters in common. If they do not, I print "-1" and jump to the next query if there are any. If **a** and **b** have strings in common, I create a list of all substrings of **a**, likewise **b**. Then I create all possible combinations of the substrings and check to see if they are palindromes. If they are, I compare to previous ones and select the best based on length and alphabetical order. I used a dictionary because its insert and delete take ***O(1)*** and I used a list because, no matter the data structure I use, I will have to loop through it which will make no difference. Overall, my program has a time complexity of $\boldsymbol{O}(\boldsymbol{n^2})$ for solving the problem for each query. I used python for this problem because I will write less code. Though it solves the problem, it might encounter runtime error for extremely large input data.

Thank you.