**Report\_Algo Project part 2**

ALL WORK IS DONE ON UBUNTU. RUN THIS CODE ON UBUNTU.

This project is implemented by multiple algorithms, each of them specifically used to minimize the time required to complete and give the minimum time and space complexity.

**FILE HANDLING**

Let us start with the file reading, which is done in 2 parts. The first part includes the data mining from the file “WHO-COVID-19”. This part uses multiple cin and getline functions to read the numbers efficiently from the file and then makes struct known as co\_node which possess the attributes for a single day for a given country. It is then inserted to a link list where there is a 2d dynamically allocated data item of co\_node. The first is the array containing all the names of the countries present in the WHO\_COVID\_19 file. It first fills the array part of it with the country name and then as the nodes are coming, it checks if two respective nodes have the same countryname. If true, it inserts it in the end of the link list or else go down the array to occupy the next available array index. Active cases are handled here.

The 2nd part of the file handling is the assignment of weights from the “country\_weight.csv”. It reads the file till end sending the values where it uses **Binary Search**, which gives the index where the countryname matches in O(log N) time. Instead of assigning all the nodes in the link list, it only assigns the first node.

**PART A**

For part A, we use a self made structure known as alpha. It is just a combination of a string and an integer value. Here we traverse to the index of the countryname by binary search ,**O(logN),** and then to the node of containing the date by linear search in the linked list,**O(L),**; assigning the value of alpha node to it, accumulating a total of **O(L\*logN).**

If a country has no node on that given day we simple assign 0 value to it.

Then we use **Pigeonhole Sort with O(n+K) (worst)**, to sort them in ascending order, ending in printing the last 20 values.

Hence the over time complexity of this part becomes **O(n+k)+O(L\*logN).**

**Part B**

Here we use yet another structure known by the name of most\_node as an innate data member with a single string data member for country name and 3 integer data items for future use as well.

For this part we refer to a function known by the name of find\_sequence. Here we start by finding the node with the country name and the starting date,(refer to part A for this process), and then starting a loop till it reaches the ending date and adding the daily confirmed cases to the most\_node’s all\_cases attribute, which adds the total time complexity of **O(logN**(binary search)\***T**(difference between the 2 dates)**) == O(TlogN).**

We perform this situation to all the countries, if they dont have the starting date, means they don’t have the ending date, we assign 0 to it.

We again perform **Pigeonhole Sort**, to sort them in ascending order, ending in printing the last 20 values, in return giving a total of **O(T\*logN + N+K);**

**Part C\_D**

For ***part c***, the implementation is to find the longest increasing sub-sequence for a following country. For this purpose we have made 2 new structures for our our use. The first structure p4 has an integer value, a pointer to another structure p5 which has an integer value as well as a string for the date.

Then we use a function that has been implemented to fill up the whole p4 array with the daily confirmed cases and the date for those cases. The number of length of those is also stored in the p4’s integer value. The time complexity for the filling up of array of p4 is **O(T)**, where T is the total number of days in that particular country for which it was active.

Then the algorithm to find the longest increasing sub-sequence is run which includes a for loop and another function to find the ceil index which is simple binary search. So this makes our LIS function’s time complexity to be **O(nlogn)**.

Then for printing it is **O(lenght of LIS)** which we will denote by **O(L).**

So the overall time complexity for part\_c is ***O(T)+O(nlogn) + O(L)*** which gives us **O(nlogn + T + L)**, which can be written as O(nlogn).

***For part\_d***,

the implementation is to find the longest decreasing sub-sequence for a following country. For this purpose, we will again use the 2 structures which were used in the part\_c and this part is also very similar to which is the previous part. We first fill up the whole p4 array with the date and the daily deaths respectively. This process takes up **O(T)**. After this we will run the longest decreasing sub-sequence with the time complexity of **O(n^2) + O(n) + O(L)**

Overall time complexity of part\_d will be as follows: **O(T) + O(n^2) + O(n) + O(L)**, which simplifies to **O(n^2)**;

***PART E***

This process also can take input from the user for the budget and can give the result. Budget == 300 is not hard-coded.

The process is completed with the help of a single function and a sorting technique. First we use another structure suitable for this process by the name of p6. This structure has 3 data members, 1 string of the name and 2 integers to hold the active cases at the last day and another to hold the weight it possess.

To reduce the time complexity, when we were inserting the node into the linklist in the file handling, we added a little more attribute to the insertion such that each first node possess another pointer with the reference to the previous country’s last node. This subtracts the time complexity with T\*K where T is the total number of countries and K is the total number of days it possess, to be precise, total number of nodes a single country has. So by just accessing the last\_node of each country we can possess the active cases on the last day thus reducing the unnessary traversals. So this takes just **O(N).**

The next part is to sort them on the basis on the daily active cases from least to most. We used heapsort for this process which gives us the time complexity of **O(nlogn).**

Finally we traverse the p6 array from highest to lowest. If the total weight of that particular country added with the preceding countries is lesser than the budget provided, we display it. Else if the budget increases from the given, we subtract that country’s weight from it and carry on. Also the country with the name “Undefined” is taken into account here.

Output of part e is the names of the country and the also the total countries under the budget which can be provided with aid.

So the total time complexity becomes **O(N)+O(NLogN)+O(N)**, thus simplifying to **O(nlogn).**

***PART F***

This particular part is the implementation of the longest common sub sequence with a slight modification of the value of K.

If the countries that are given are converted to arrays of p5 with each array consisting of the total number of days it has, which were computed while file-handling, with the value of the daily confirmed cases.

Then they are sent to compute the longest common sub-sequence. If both the country’s daily confirmed cases subtracted, (and multiplied by -1 if the answer is lesser than 0), are lesser than 0 then and only then it is incremented in the matrix. This algorithm is computed in the following time complexity, **O(n^2)+O(n^2)+O(n^2).** Hence simplifying to **O(n^2)**.   
This function give the outputs of the LCS as well as the length of it.

***INT MAIN and MENU***

Int main is implemented with extreme care and responsibility. It allows you an option to see the options. **The user is also given the option to record his whole experience in a text file. It will record all the options chosen and all the output. Everything will be on the text file and also on the terminal to enable the best possible user interface. The user is given also the option to see the pdf on the terminal screen if he wants to.** Then we ask to enter the number of the query. After that, for queries which ask for the country name, if the user enters the wrong input, I have made the function to print all the countries which match the closest to the user’s input. After that, the user is asked to pick from the countries, if printed, else the user is also given the possibility to re-enter the name of country again. Here binary search and auto\_correct functions are implemented for this. Our output screen also prints the total time that the given part took, so we can define how much time it actually takes and compare the time on paper to the time in actuality. After the part completes successfully, we again ask if the user wants to choose another option or else wants to quit. So the menu is carefully made for the user’s better interface.

***Space Complexity***

**Part A**

*Personalized sruct\_array = O(N)*

*Pigeon Hole Sort = O(2\*N)*

Overall = O(N)

**PART B**

*MOST\_NODE (personalized structure array) = O(N)*

*Pigeon Hole Sort = O(2\*N)*

Overall = O(N)

**PART C**

LIS = O(2\*N + 8) = O(N+C)

Overall = O(N);

**PART D**

*LDS = O(N + 4) = O(N+C)*

Overall = O(N);

**PART E**

*p6 (personalized structure array) = O(N)*

*Heap sort = O(1)*

Overall = O(N)

**PART F**

*p4 (personalized structure item) = O(2\*N)*

*LCS = O(N\*M) (size of 2 arrays)*

Overall = O(N+N\*M) = O(N\*M)