Name: Pranto Roy ID: 22301261

1(a)

In this task I created a list of total inputs in next_red and sum in another list name sum. After testing every element to get the required sum in a nested loop we finally can find it by a if condition and to keep track of where the pointers were ~~we~~ I use two pointers. If we cant find the sum ~~we~~ I simply write 'impossible'. And in this code the time complexity is $O(n^2)$

1(b)

In this task I started by setting two pointers left and right at the first and last index respectivly of a sorted list. Then in the interation I set as the left pointer is less than total number of input and right pointer is greater than 0 to check the sum. Then if the sum of elements at left and right pointers equel to sum we find the solution and record the position in Checker 1 and 2 and break the loop. Next I adjusted if the sum is less than left then the pointer will move one step right and if its more than right right will move one step left. Then I ~~we~~ find the solution.

id not found then IMPOSSIBLE. In this code time complexity is O(n).

2(a)

In this task firstly I read the length of Alice and bobs and also ~~both~~ the list of inputs, arr1 and arr2 are those 2 lists. Then I created a empty list to store all elements. I converted each element to int in the list lst. Using a sort function I sorted in ascending order, and thus I get the output and here the time complexity is O(nlogn).

2(b)

In this task read the length of Alice and bobs list and then split to get arr1 and arr2 two lists containing 2 sets of inputs. After that I created a empty list to store all outputs and created two pointers for both arr1 and arr2. Then I used a while loop which will run untill both the pointers reach the end of their respective list Then I used

a comparision by conditional statement when they were being appended to lst and increasing their pointer value as well in each iteration.
After the loop ends I added the remaining elements of both list to the merged lst and get the final output here Time complexity is O(n).

(3)

In this task, by using greedy algorithm and by read line I take the total number input and used in for loop as a range and for every iteration a group of number were being appended to an empty lst. Then I used bubble Sort to sort them according to their end time. Then I set a start time and end time as a 0th index of lst and set a counter and I ans from lst1 empty list. Then I appended the Start and end time. Then I ran another loop from 1 to end for lst and ~~get a~~ if our condition was satisfied it will be append to lst1 and our output is lst1.

④

In this task I again used greedy algorithm. Firstly in my code I read the file to find how many ~~time~~ work are there and how many People. Each work is Sorted by Merge Sort based on thier end time. Then I Iterate the Sorted tasks, for each task the code will find a earliest ~~time~~ available Person who can complete it without overlapping with other tasks assigned to them. If a Suitable Person is found, then the task were assigned to him and the count was increased by 1 each time and. finally we get the output.

Brainstorming

To get O(n log n) for task 4 I will use the following steps:

① Sort by end time: I will Start by Sorting the tasks based on there end times. This operation will take O(n log n) times.

② Greedy task Selection: I, Iterate through the Sorted tasks at each step I Select the task with the earliest end time that doesn't overlap with other task (Previous). This can be done efficiently since the task are sorted by end time.

③ **Counting:** Add a counter to keep track of the selected task

④ **Output:** Finally I can get the desired output

So from this Steps ~~It~~ can be ensured that I will always assign the next activity to the Person who became available first and by Sorting the activities by their end times, I can ensure that I am always looking at the earliest possible end time for each location. And so we get a $O(n \log n)$.