Task 01(1):

Here, I used two nested loops and then by adding two digits I checked wheather it is equal to target or not. I also used Flag in this code so that if the sum and targets match it will write the output and make the flag value false and it will ultimately break the code. On the other hand if the sum is not equal to the target then the flag will remain True and will write ~~Impossible~~ 'IMPOSSIBLE' in the output. As for using two nested loop the code is an $O(n^2)$ solution.

Task 01(2b):

The given list is organized in ascending order so that, I set two pointers start and end. start pointer starts from 0 index and the end pointer indicates the last index of the list. Inside the while loop I checked if the sum of the two digits is lesser than or greater than target. If the sum is less I increased the start pointer by 1 and if the sum is greater than the target I decreased end pointer by 1. For using one while loop the code is a $O(n)$ solution.

Task 02(1):

Used sort function here and the time complexity of sorting function ~~has th~~ is O(nlogn)

Task 02(2):

Here I just use the merging part of the merge sort, (which uses 1 while loop) so that the time complexity of the code is O(n).

Task 03:

Sorted the time slot using insertion sort by their end time. After ~~that~~ that I used a loop ~~to starting~~ that checks if the persons current task's ending time is less ~~equal~~ or equal than the next task's starting time. If the condition gets fulfilled it appends the task inside the list named task.

Task 04:

First I wrote almost similar code as task 3 but here I had to take a 2 dimentional ~~array~~ list and used two nested loops ~~to~~ The inner loop seeks a person with an available time slot

by comparing their end time with the start time of the current activity. Here the goal is to assign the task to the person for whom time difference is minimum and by checking the time difference with minimum it assigned the activity and increased the value of count accordingly then updated the respective time slot of 'people' list. By repeating the process I get the final count and write it in the output.

Brainstorming of Task 04:

Task 04 can be solved in O(nlogn) by using a greedy algorithm. The key thing is to sort the tasks based on their end times in ascending order. This sorting algorithm takes O(n log n) time and the subsequent greedy assignment takes linear time so that the overall time complexity can be O(nlogn)