CSE221
SEC: 20

B M RAUF
ID: 22201782

LAB02

Task01

① It iterates through each number in the list. For each number, it calculates the complement. Then it searches for the complement within the remaining elements of the list. If the complement is found, their indices are written to the output file. If no match is found for the current number, the loop moves on to the next element. Lastly, if the entire list is traversed without finding a match, "IMPOSSIBLE" is written to the output file.

Since, I use two nested loops and they run 'n' times the overall time complexity of the code is $O(n*n) = O(n^2)$

② It starts with two pointers, l and r, at the beginning and end of the list, and iteratively compares the sum of the elements pointed by l and r. If the sum matches the target, their indices are written to the output file and the search ends. If the sum is less than the target, the left pointer (l) moves forward, searching for a larger number and if the sum is greater than the target, the right pointer (r) moves backward, searching for a smaller number. Lastly, If no match is found, "IMPOSSIBLE" is written to the output file.

Since, it is done using single loop, the overall time complexity of the code is O(n).

Task02

① This function basically implements the merge sort algorithm. to sort a list of elements. It divides the list recursively into halves until each half contains only one element means already sorted. Then, it merges the sorted halves back together, comparing elements from each half and placing the smaller one in the original list at the correct position. This process continues until the entire list is sorted. By repeatedly dividing and conquering, the function sorts the lists and it's overall time complexity is O(nlogn)

② It starts with pointers i and j and an empty list to store the merged elements. It iterates as long as both pointers haven't reached the ends of the lists. In each iteration, it compares the elements pointed by i and j and append the smaller element to the lists and the corresponding pointer i or j is incremented. If both elements are equal, then append the two elements and increment both pointers. Lastly, if any elements left to add, remaining elements are added to the list and the merged list is returned. It takes O(n) time.

Task03

Using selection sort technique, it iterates through tasks and find the one with the earliest end time. If multiple tasks share the same end time, it chooses any of them and ~~in this way it sorts the list~~ swap the current task with the ~~rem~~ earliest end time task, ensure the tasks with earlier end times are processed first and in this way it sorts the list. Then it iterates through tasks again, checking if their start times are after the previous end time. If so, these feasible tasks are added to the result list and update the endT for the next iteration. Finally, it writes the number of completed tasks and their details to the output file. Overall, time complexity is $O(n^2)$.

Task04

~~Sa~~: It sorts the tasks ~~a~~ using same technique of task03. Then, it initializes a list to store the last completed task time for each person. It iterates through the tasks and for each task, it finds the person who can finish the task the earliest. If a person is available, the task is assigned to him and the count variable is incremented. Lastly, it ~~returns the~~ writes the count/maximum number of completed task. ~~is retu~~ Overall, time complexity is $O(n^2)$.

Brainstorm:

Yes, I can solve the problem in $O(n \log n)$. Since, ~~sor it~~ sorting dominates in this task and in my code it takes $O(n^2)$ time, if I use merge sort technique ~~it~~ to sort the tasks instead of selection sort technique the overall time complexity of the problem will be $O(n \log n)$.