Lab-02

Task 1 (1) time complex → $O(n^2)$

Open input file then read the line and take len of array and target sum. Here to solve this problem I use two nested for loop. first loop take a int. value from array and second loop add with next value and compare with target sum. if target sum == add value, I return the number of adding value.

Task 2 (2) time ~~com~~ → $O(n)$

I use two pointer method and one is iterate from first (i) other is from last (j). then add those two value. If the value is greater then target sum I iterate from last cause there I find smaller value. Same for when i get smaller value than target sum, iterate from first.

Task 2 (1) simply add two list together then use merge sort() and merge function. firstly separate two list from input file then add them together. Then the list is devided by divide and conquer policy we use recursion to divide the list, basis of len(lst) // 2 = mid. From mergSort() I call merge function then i compare element and add the element in the result list.

Task 2(c)

Here i use merge sort function technic. where i use two pointer method for two list. then compare the value of two list. which value is smaller add in the sortlist=[] and increase the value of pointer. if both list value is eqal then add both element in the sortlst. and increase the value of both pointer.

Task 3

take start and end list. The task schedule is sorted by end time. At the time of sortation, separate the time value of start and end and append in the start and end list. Take a variable name current assinged with 0. Then check the start time if start time is eqal on graten than current then increase a count and add the task in the task list. Then i update the current with that task's end time.

Task-4 From input sperate the task number and person number. Here i use greed algorithm make a list of person and assinged with 0. sorted the task list by their end time (ascending). [same as task 3]. Then i use greedy algorithm. every time in the loop i sort (reverse sort) the person list where assigned with the person's task end time. Then travas the task (start) list and compare with person's end time and task's start time. if matches the condition increase a count and break the inner loop.

Task 2 (1):

mergeSort() $\longrightarrow O(\log n)$

merge() $\longrightarrow O(n)$

total $\longrightarrow O(n \log n)$

merge() 4