Task 1 ($O(N^2)$ soln).

Here I introduced two pointers.
and a boolean variable = "False".
~~I~~ This boolean variable will work as an indicator for my checking.
So, I introduced two loops. First one is the first ~~in~~ pointer that remains fixed on an element for one iteration.
The second loop is the second pointer which iterates the other element ~~fro~~ next from the first pointer and sums them. If my summation is equal to the 'S' or sum then my boolean variable is changed and my loop is broken!

Task 1(ii) O(N[illegible])↑

It's more like the O(N^2) solution, I've Just implemented the reverse algorithm. Here I implemented "substraction".
So, I kept every element in a dictionary making it the key and making the index it's value. Then I substracted it from the "S". If the substracted part was also in the dictionary then it's true that I found the sol^n.

Task 2 (i).

I used merge sort here with while loop. Where I first checked an object/element from Alice's list with Bob's list. If the the element from alice's list is ~~t~~ shorted, then it will get appended in the merged list. and vice versa.

Then I expanded all the remaining elements in the merged list.

Task 2(ii)

Here I implemented a list that is empty at the very first. Then I declared The length of Alice's list, Bob's list and a new list that I just made.

Then I check my conditions and put the elements in the merged empty list's position. Finally ~~we~~ I do the merging ~~sort~~ again about the ~~elem~~ remained elements.

Task 3

(i) I made a list empty.

(ii) Then I used lambda function to sort all the intervals based on their second element.

(iii) $\boxed{\text{tasks.sort(key = lambda x: x[1])}}$

(III) Then I took a current_end_time variable with a negative value so that for the first case my end time is ~~lesser that~~ greater than it.

(IV) Then I continiously updated that variable so that schedules don't overlap.

Task 4

(I) ~~We~~ I set two variables to store my initial start and end time.

(II) Then I iterated over the list to find my current start & ending.

(III) I compared my current and itial start end times.

(IV) On the final I ran my greedy algorithm in a loop ranged by my workers.