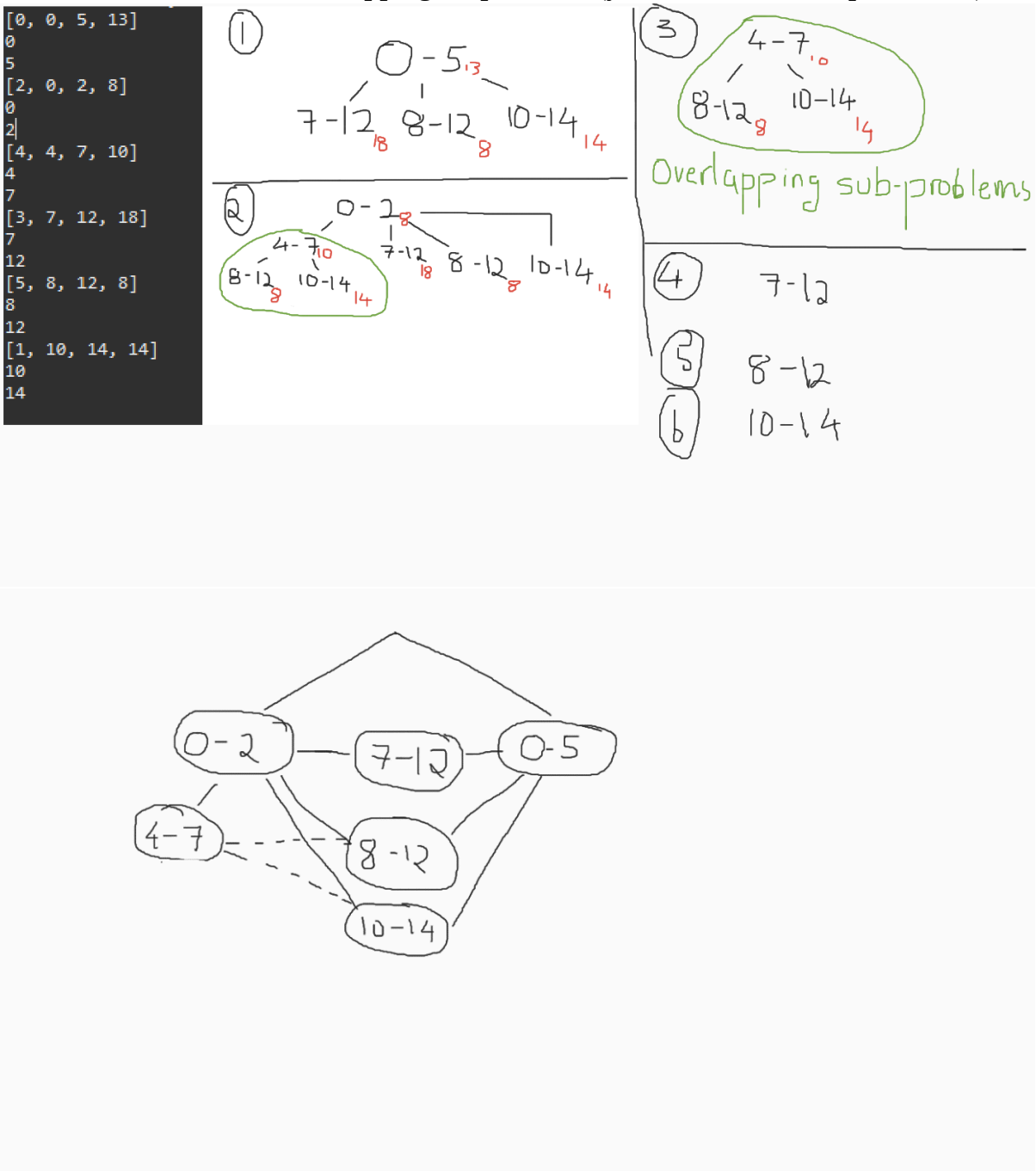
1. Show how you can decompose the problem into subproblems.

You can decompose the problem into sub-problems

So we can decompose the problem into sub problems because when we think about the problem itself, we have a situation whereby we can choose different options, but then we back track to choose a different route.

2. Show that there exists overlapping subproblems (you can use an example for this).



3. Show that the problem exhibits optimal substructure.

So we assume we know what the optimal substructure is i.e we know the correct sub-tree. As long as the problem shows that there is a optimal solutions to the subproblem then we can say

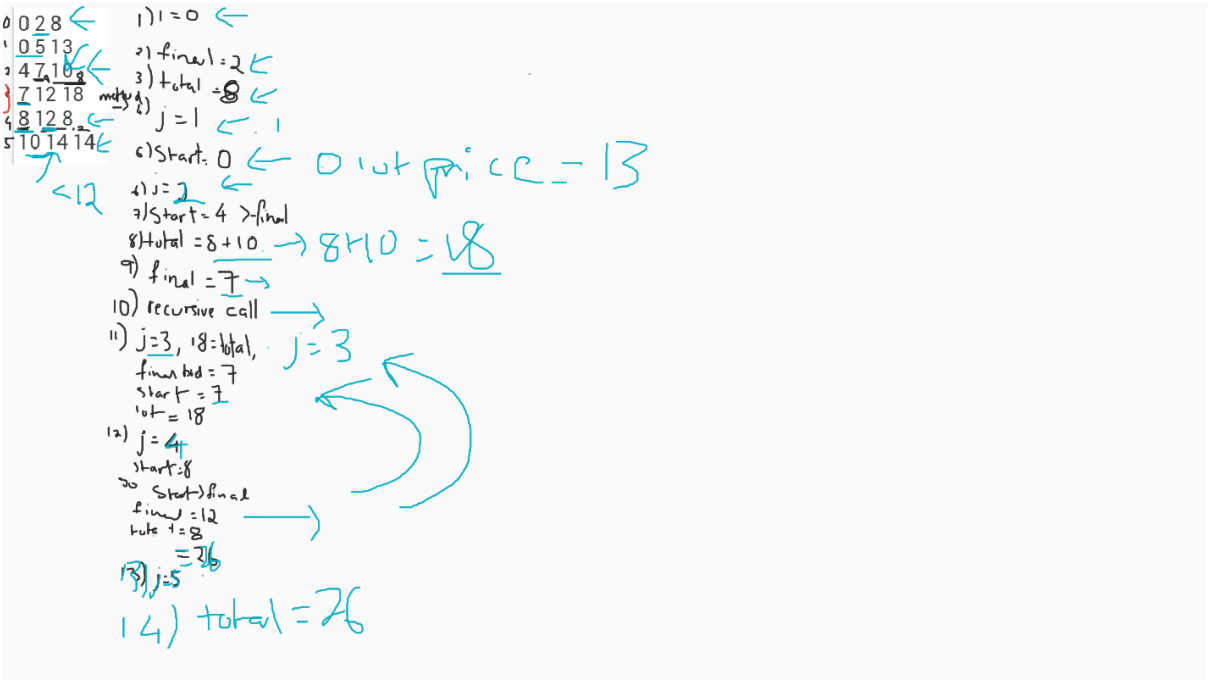
that there is optimal substructure. So the optimal solution at the top must contain the optimal solution in the substructure.

4. Show the recursive relation that gives the optimal solution, that is, the maximum income that the company can get. Explain how it works (examples would be good here as well).

Recursive relation that gives the optimal solution would be related to the recursive call made in the code.

Recursive call: recursiveCall(i, finalBid, sortedArray, lookup)

So each time a call is made, i++ and the finalBid is updated everytime. This allows us to continually compare the bids and find the best solution.



5. Show how you can derive the solution that leads to the optimal solution, that is, how to identify the bids that lead to the maximum income. You do not need to write the code for it. You may use an example to demonstrate how your algorithm works.

Having a maximum value that stores the max added value. This allows us to store the maximum income.

So for the example shown, you'd get 0-2->4-7->8-12 as one subtree. Total up the price of that subtree, and thanks to the recursive call, it will back track and give us the total for 0-2->4-7->10-14.

Not particularly writing code, but to identify the bids that lead to the maximum income each time, using the code as an example, it stores the highest value obtained from the recursion call, and if the new value is higher than the previous value, the previous value is replaced.