COMP3010
Algorithms Design and Analysis

Assignment 2

Trideep Lal Das

45532125

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

One way to show we can decompose a problem into subproblems is when we start to break down the problem, we may see that we can apply a recursive algorithm.

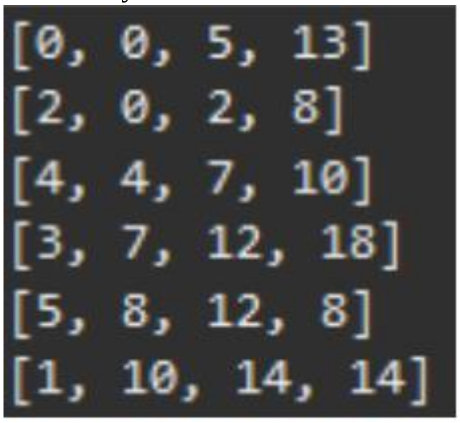
So, we know we can decompose the problem when we take a look at what we need to do. In this instance we need to find ways to break down the problem. Having a look at what we have; Start lot, final lot and Price, we know that as long as the bids do not overlap, we can add the prices up.

So, we can have 0-4 -> 5-7 but not 0-4 -> 2-5.

So, from here we can start to form an idea of the sub problem. In this case a sub problem would be if we have 15 lots and something like 0-4, 2-3, 5-7, 8-9, 8-15, We can say that we can take the lots 0-4,5-7,8-9 or 8-15 . Thus, we can say that 8-9 and 8-15 are sub problems of 5-7 and 5-7,8-9,8-15 are subproblems of 0-4.

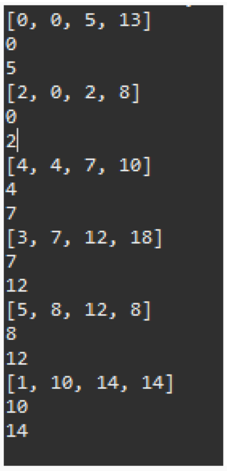
2. Show that there exists overlapping subproblems (you can use an example for this).

Taking the example in test_case_01, we sort the bids according to the starting lots and we get a sortedArray like such:

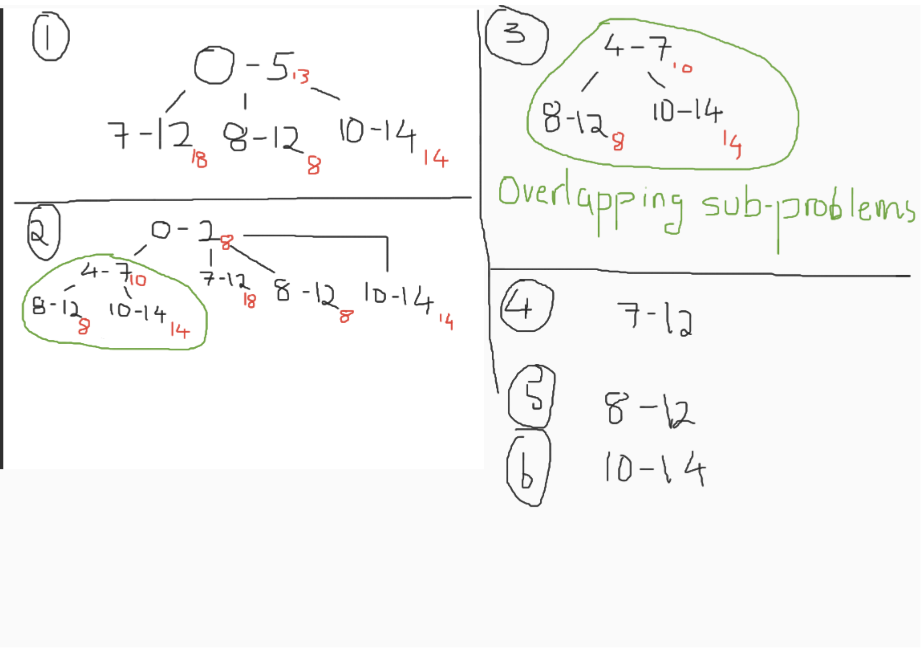


```
[0, 0, 5, 13]
[2, 0, 2, 8]
[4, 4, 7, 10]
[3, 7, 12, 18]
[5, 8, 12, 8]
[1, 10, 14, 14]
```

From here, we can draw a tree-like visualization to see the sub-trees that the code visits/the subproblems that exists.



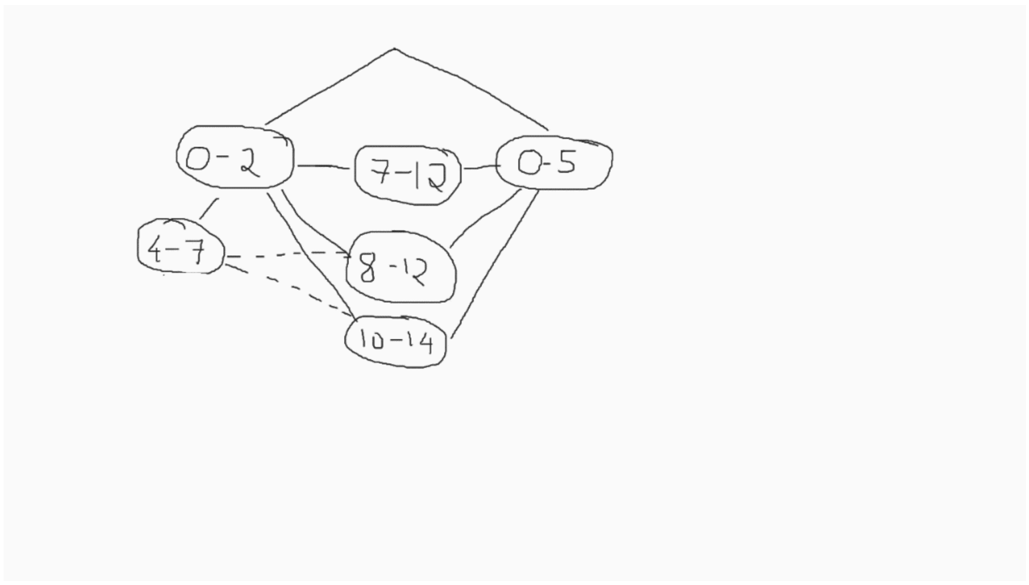
```
[0, 0, 5, 13]
0
5
[2, 0, 2, 8]
0
2
[4, 4, 7, 10]
4
7
[3, 7, 12, 18]
7
12
[5, 8, 12, 8]
8
12
[1, 10, 14, 14]
10
14
```



(Numbers in black are the lots and the numbers in red are the price).

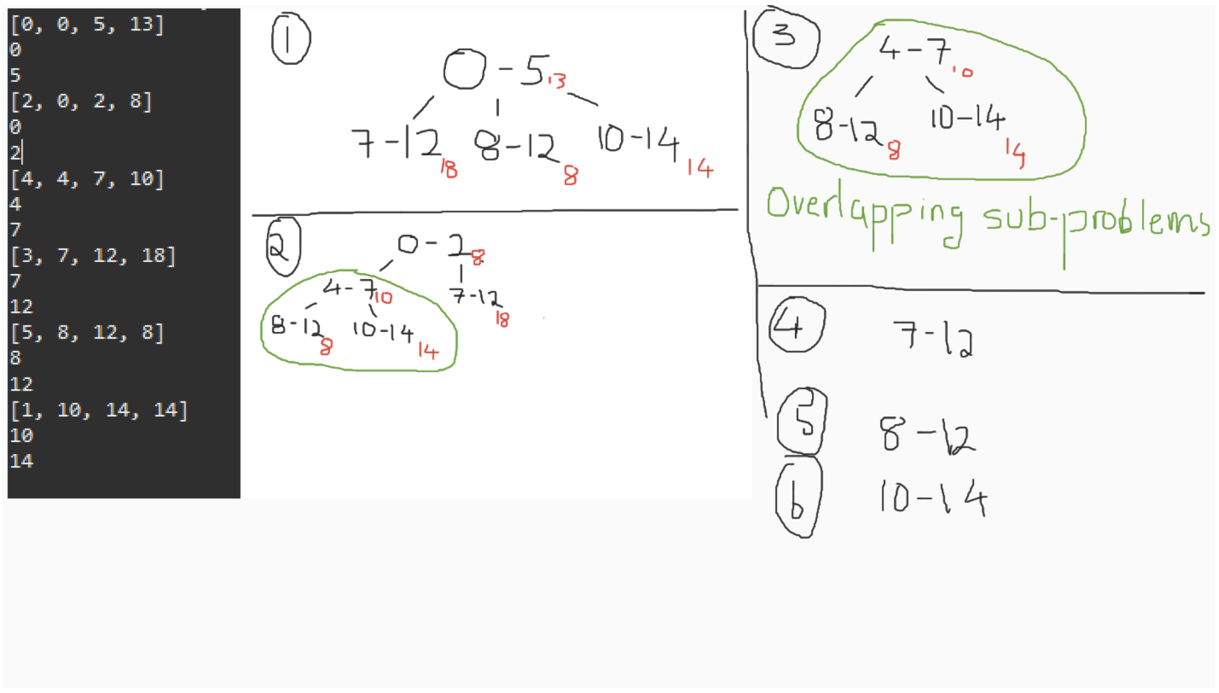
Here we can see that the overlapping subproblem is any branch that visits the 4-7 node.

(Another drawing to better illustrate)



3. Show that the problem exhibits optimal substructure.

Using the diagram from question 2 here:



```
[0, 0, 5, 13]
0
5
[2, 0, 2, 8]
0
2
[4, 4, 7, 10]
4
7
[3, 7, 12, 18]
7
12
[5, 8, 12, 8]
8
12
[1, 10, 14, 14]
10
14
```

We know that a problem exhibits optimal substructure if the optimal solution to the problem can be constructed using optimal solution to the subproblems.

And so, this means that the optimal solution at the top is contained in the sub-problem. And so, the optimal solution for the 4-7 sub-tree is to take the 10-14 to give a price of 24$. That is the optimal solution of the 4-7 subproblem.

Applying this, the optimal solution of the problem is to take the 0-2 sub tree which contains the 4-7, 7-12, 8-12,10-14 subproblem. We know that 4-7 is the optimal solution as its price is greater than the rest and has 8-12 ad 10-14 in its optimal solution. Thus, the optimal solution is to take 0-2 branch which has the 4-7branch which we have established has the optimal solution of 10-14 making the optimal solution 0-2,4-7,10-14 giving us 32.

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).

So, we need to get the income of the company. So, we have income and so every time we find a new value that is larger than the current income, we replace the current income. When we do the whole code for example in the nested for-loop, we have i to be the beginning of the tree and then j to be the subtrees and so every time we look at the subtrees, we check to see if the value is stored and if the value is not stored we store it, and have the total the highest stored value and finally returning the price of the main tree and the subtrees.

So with the subproblems we have a relationship between the prices and the lots for if the lots are not overlapping, the prices are added and so from a very basic standpoint we can say that the more bids we can take, the more prices we can add up and thus the higher the income. So bids and income are directly proportional and the lots size would be indirectly proportional as the less lots we can take in a bid, the more bids we can take as there is inherently more space/lots left for remaining bids.

The best attempt for a recurrence/recursive relation that can be provided here is income(bids), lots (1): income(bids-1) + lots(2)...income(1) + lots(bids)

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.**

Essentially, we are just brute forcing with a lookup which is what DP is right. So, we know we are getting the optimal solution by obviously placing our current best solution which we have arrived from the first subproblem with future subproblems. If future subproblems have a better solution, we replace that value and compare that value with future subproblem solutions.

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

So, for the example shown, you would 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.