**Code Structure**

The solution to this problem has 5 components:

* ***ReadData(String infile)*** *throws exception* which reads the supplied data to the given problem
* An inner class ***Bids*** to store the starting lot, ending lot and the bid price.
* ***LotComparator***class which compared the ending lot numbers and sort them according to increasing order with the ending lot number.
* ***PopulateBids()*** which creates an object array to hold a collection of bids.
* ***Solve(String infile)***which gives the highest amount of revenue out of the provided bids that are within the given amount of lot spaces.

There are also some data structures that were used to store the information:

* A HashMap *bids* to initially store the starting and the ending lots along with the price of each bid.
* Two global variables *numOfLots* and *numOfBids* which are used to define the sizes of arrays in the *solve(infile)* function.

In this report, it will go into detail about the ***solve(String infile)*** method which will include the following aspects:

* Problem decomposition
* Overlapping subproblems
* Optimal substructure
* A recursive relation which gives the optimal solution
* Constructing the final solution of giving the maximum revenue out of the bids

**Problem Decomposition**

The problem states that there is a list of submitted bids that buys a series of lots that are next to each other, and this algorithm needs to find the desired choices of bids that should be accepted to maximise their revenue while not having any bids of the same lot or series of lots (i.e. conflicting bids). Given these requirements, we can reorganize the given information into data structure that are easier to do computations on.

Each line of the data file holds information on the bids in the sense that it provides:

* Starting lot number
* Ending lot number
* Price associated that the bidder wants to pay for that series of lots

To see the subproblems to this solution, consider the following example:

Given a set of bids that contains the information mentioned above, the intuition for finding the bids to maximise the revenue and does not overlap with other lots is to first choose the bid that provides the highest revenue and add it to a solution collection. Once the choice is made, the total number of bids that are left is one less than the original data set. Then out of the remaining bids, iterate all remaining bids and choose the highest paying bid that does not overlap with the previous choice’s lots and add it to the solution collection. As this process repeats, we can see that at each iteration when the choice is made, we generate a problem where the input data space is strictly decreasing which is the very definition of a subproblem where the solution (the highest revenue bid that does not overlap with the solution bid collection’s lots) to the subproblem contributes to the overall problem (i.e. The answer to the subproblem is added to the overall solution bid collection).

**Overlapping Subproblems**

Once there exists subproblems, it is relatively easy to see the overlapping subproblems. In this problem, we can see that after each choice is made, the remaining subproblem will be subjected to the same criteria for choosing the next compatible bid which needs to satisfy the following two requirements:

* The choice must be compatible with the bids in the solution space. In this problem, a compatible choice is defined as the choice that has a beginning lot number bigger than the ending lot number of the previous choice.
* The choice must also be the highest paying bid out of the subproblem space.

Since all the subproblems will make a choice according to the above two requirements, this satisfy the definition of an overlapping subproblem where the subproblems is of the same nature as the overall problem.

To illustrate the above explanation through an example, take the following set of bids where:

* A close up of a sign

  Description automatically generatedfirst integer represents the bid number
* Second integer represents the starting lot number.
* Third integer represents the ending lot number.
* Fourth integer represents the price for that bid.

In the first iteration of the algorithm, we choose the activity with the highest paid price which in this case is bid number 3. After this choice, the subproblem space will include the bids [0,1,2,4,5] and out of those bids we choose the highest paid compatible bid which is bid number 0. this process continues until there are no compatible bids left.

However, this approach to solving the overall problem is not sufficient enough to give the optimal solution because in the above example, after 2 iterations it terminates and returns 31 with bids [3,0]as the optimal solution while in fact the optimal solution is 32 with the bids [1,2,4].

**Optimal Substructure**

In the previous section, we have stated that the overlapping subproblems does not produce the optimal solution. This is because although we have checked all bids in the problem space, the algorithm omits some of the bids that potentially gives a more optimal solution.

However, we can do some pre-processing and alter the choice that we make at each iteration on the problem set so that all bids can be considered. Since the requirements of the problem states that bids start from a beginning lot to an ending lot, we can assume that the starting lot number () is less than the ending lot number ().

Therefore, we can utilize this definition to sort the problem space according to the finishing times in increasing and make the choice at each iteration as the first element in the problem and subproblem space. When we sort this, the first element of the problem space will always start from the smallest lot number.

As a result of this tweak, we can see that there exists an optimal substructure. Note for this optimal substructure, the criteria for the overlapping subproblems changes when developing the optimal substructure and also the choice that is made at every iteration. The new criteria are:

* Choose the first element of the subproblem space.
* Check whether if that choice is compatible with all the elements in the solution space.

With the above criteria, it will give an optimal substructure in the sense that the remaining subproblem space will always result in the maximum amount of lot spaces compared to the previous overlapping subproblems. This leads to the maximization of the potential revenue that can be creates as the more lots that are available, the more revenue that can be made.

With the above example, when we sort the bids according to the finishing lot numbers in increasing order, we end up with:

Text

Description automatically generated

When we pick the first bid in the above set, we can see that the remaining available lots is maximised with 12 lots remaining compared to 9 lots remaining.

**The recursive relationship**

Once we have determined the optimal substructure, we can then develop the recursive relationship to solve the overall problem. The recursive relationship can be based on the criteria for arriving to a conclusion for the optimal substructure.

We can illustrate the recursive relationship through the above example:

Text

Description automatically generated

According to the choice that was specified to develop the optimal substructure which was to always choose first element of the subproblem set. The recursive relation will follow the following procedure:

* Choose the first element of the subproblem space.
* Check whether if that choice is compatible with all the elements in the solution space and choose that compatible bid.
* Check whether if the price of the choice is larger than the previous element in the solution space. If so, add the price to the previous element in the solution space.
* The process repeats until all the elements in the original problem space is added to the solution space.

Hence, we have the following recursive relationship:



**Construction of the solution to the overall problem**

After all the necessary computation, we end up with an array that contains the maximum amount of revenue for the entered bids. To get the solution to the original problem, we simply iterate through the array and obtain the largest element and return it as our solution.