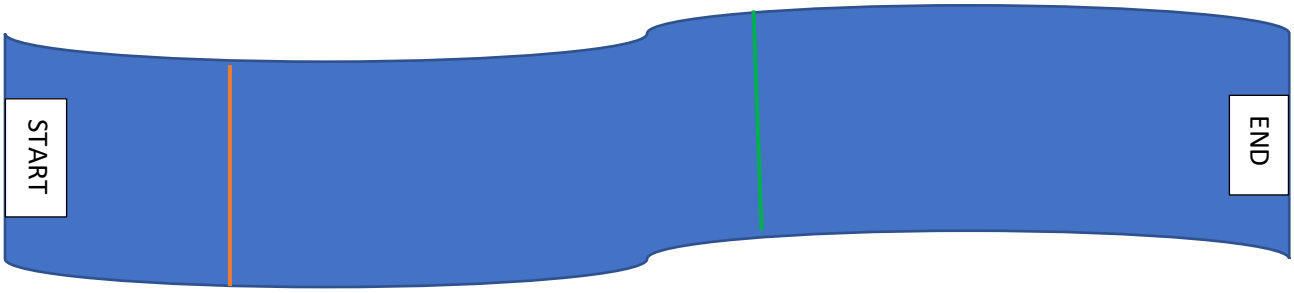
The brute force solution to this problem would be to calculate the cost of each possible order in which to place the dams and to return the smallest cost. There are Y! permutations (where Y is the number of dams to be built) and the bigO to calculate the cost of each permutation is YlogY. (I calculate the cost by putting start and end into a list and then for each dam in the sequence, I do a binary search to find the insertion point, calculate the distance from the dam in the list before the insertion point to the dam in the list at the insertion point [to find the distance between the two dams that this one will be place between], and then I add the current dam to the list). Thus, the total cost would be Y!YLogY.

My key insights were as follows. First, calculating the cost to place a dam is the value of the dam to the right of it minus the value of the dam to its left. Thus, calculating the cost of adding a series of dams in any order, will depend on the values of the dam to the left of the sequence and the dam to the right of the sequence but nothing else. Thus, if I place a dam at a point, the optimal way to place all the dams to the left of it, and the optimal way to place all the dams to the right of it, are independent issues, as the calculations for each cannot include anything on the other side of the dam just placed.

The diagram below illustrates a set of two dams to be placed between two existing dams. The optimal way to place them incurs a cost of the current size of the river (for whichever dam is placed first). If the red dam is placed first, then the optimal way to place them would incur a cost of the size of the river plus the cost of the two subproblems that now exist: namely the cost of the optimal order in which to place any dams between start and red plus the cost of the optimal order in which to place any dams between red and end. If the green dam is chosen first, then the optimal way to place them would incur a cost of the size of the river plus the cost of the optimal order in which to place any dams between start and green plus the cost of the optimal order in which to place any dams between green and end. So which dam would be placed first in the optimal solution? Well, whichever incurs the minimum cost, and we can find which incurs the minimum cost simply by taking the min of the cost of placing red and then the rest in optimal order (length of river +cost of optimal placement order of dams from start to red +cost of optimal placement order of dams from red to end) and of the cost of the placing green and then the rest in optimal order (length of river +cost of optimal placement order of dams from start to green +cost of optimal placement order of dams from green to end).



With all of this in mind, an optimal substructure begins to emerge. Let OPT(i,j) denote the minimum cost accrued by placing all of the dams d in Y, where i<d<j, in the optimal order. There are two different cases possible when we consider OPT(i,j). If there are no dams d in Y that satisfy the above conditions, then by the problem definition, since 0 dams have to be placed, OPT(i,j)=0. The second case, where there is at least one dam that must be placed in between i and j, is slightly more complicated. One of those dams must be placed first, and as per the problem definition, no matter which one is first, the cost will be j-i. Then, if we call the unknown dam we chose to place D, since the

subproblems on each side of D are independent (see above), we can express the minimum cost to place the remaining dams as OPT(i,d) and OPT(d,j).

With all of the above it mind, it becomes clear that if OPT(i,j) denotes the minimum cost accrued by placing all of the dams between i and j in the optimal order, then, when there are no dams between i and j, the minimum cost would be 0, and if there are, then the minimum cost would be j-i + the minimum for all d between i and j of OPT(i,d)+OPT(d,j). Written as a more formal recurrence:

$$\mathrm{OPT(i,j)} = \begin{cases} 0, & if\ no\ d\ in\ Y\ where\ i < d < j \\ \text{j-i} + \min_{\text{for all d in Y where i<d<j}}\{\mathrm{OPT(i,d)} + \mathrm{OPT(d,j)}\}, & if\ at\ least\ one\ d\ in\ Y\ where\ i < d < j \end{cases}$$

In order to code this dp solution, I created a HashMap of HashMaps to store each i, j, and the corresponding OPT(i,j). Then, to calculate each OPT value, I add 0 to the beginning of Y and riverEnd to the end of Y. Then, I start by calculating the OPT of adjacent dams as the i and j values. Then, I calculate the OPT values from each dam to the dam two places away from it. Then, I do the same for each dam to the dam three places away from it. And then 4…. All the way until I get to the calculation from the first dam to the last. The loop is shown in the pseudocode below:

```
For i from 1 to Y.length-1:
    For j from 0 to Y.length-i:
        int leftDam=this.y[j]
        int rightDam=this.y[j+i]
        calculate OPT(leftDam,rightDam)
        store calculated OPT in HashMap for lookup later
```

This double loop is, in reality, a lot less than n^2 (where n is the size of the dams array), but for big-O purposes, it is O(n^2). However, each time we calculate OPT(leftDam, rightDam), that method must consider each dam d in Y, and, if it is in between leftDam and rightDam, do two lookups based on it. Thus, this method has a runtime of O(n). Therefore, the total runtime of this DP algorithm is n^2*n, or O(n^3).