Akarsh Kumar
EID: ak39969
EE-360 Algorithms Assignment 3
Dr. Santacruz

Part 1.

a. Explain what the dynamic programming subproblems are, and provide a recursive formula for OPT:

Let us start off by understanding the problem. Us, player 1, is picking the location of where to build a wall (in between array indices) in order to maximize the total gold we get. The enemy, player 2, is picking which direction is attack from in order minimize the total gold we get. The dynamic programming problem is to calculate the total gold we can achieve when both players are playing optimally, given an array of cities.

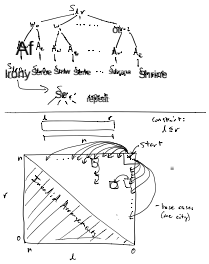
The subproblems are simply subarrays to this problem. This is because, when considering every possible choice that can be taken by both players (brute force), we realize after we build a wall at location w (immediate index left of the actual wall location), the enemy player then has to choose either the left subproblem or the right subproblem (whichever one results in less gold for us). Since the subproblems are simply subarrays, we make the parameters OPT a left and right index to define a subarray (inclusive and exclusive, respectively).

Now that we have defined the subproblems, let us find the OPT recursive formula. The brute force method we discussed earlier involves us "experimenting" and putting a wall down wherever we can and finding the gold value each one will give us and maximizing over this. At the same time, the enemy, given a subarray and a wall position, will minimize the gold value either by attack from the left or right. This exact maximizing and minimizing is defined by the following OPT definition:

$$OPT(l,r) = \max\{\{\min\big(OPT(w+1,r), OPT(l,w+1)\big)\} | w \in \{l, l+1, \dots, r-2\}\}$$

The max function is maximizing a function of w over the set of all w possible (where you can build a wall). The min function is simply taking the min of the right subarray gold value and left subarray gold value. The first call to this function will be OPT(0,n), where n is the length of city array.

b. Provide a succinct argument of correctness, and explain and justify the complexity of your algorithm.



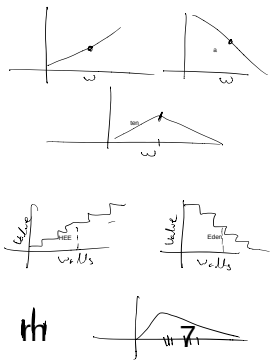
Here is a picture of the overall problem in the first diagram (the recursion tree). The second diagram is a picture of the OPT that needs to be calculated. It is a 2D array representing the left and right index of the subarray. Notice how the top right box is what we are interested in. Also given a box, to calculate it, you only need the values of the boxes left of it and below it. Because of this, OPT is calculated by from left to right and bottom to top in the square diagram shown above.

Now to build up the OPT matrix, it is clear that it will be n^2 * O(time to calculate a box). If we calculate the box value by doing a max of the wall positions, we will need to go through a worst case of n wall positions. This will result in O(time to calculate a box) = n, and we will get a $O(n^3)$ solution.

A better approach that was implemented in this project uses the following lemma:

```
Lemma: Assume that building the wall at position k+1 is at least
as good as at position k. Then building the wall at k + 1 is at
least as good as at any other position k′ such that k′ < k. (A
symmetric argument holds when position k is at least as good as
position k + 1.)
```

What this lemma essentially means is that the gold values of wall positions will be non-strictly increasing until it hits a global maximum and then non-strictly decreasing. Essentially, if we graph the gold outcome vs a wall position, the graph will look similar to this:



Because the global maximum is a single point we are trying to find, we can use binary search to find this maximum instead of going through all wall positions defined in the OPT relation. The implementation of binary search used samples OPT at a point k, k-1, and k+1. Depending on if the values are increasing, decreasing, or increasing and decreasing, it searches the right side, left side, or breaks, respectively.

This algorithm will terminate because it is calculating only n2 /2 boxes at most with a definite log n per box. This algorithm is also correct because it is literally simulating the 2 player min max game completely. It is going through iterations and choosing wall positions for player 1 and choosing attacking sides for player 2. The binary search optimizing is correct because it is a given lemma assumed to be correct by the professor.