Steven Bui, Ethan Ng, Alex Young
CS 325

Group Assignment 2

Description of Algorithm

To start, the function opens the input file and stores the information in an array.

We have two nested for-loops with variables that start at the maximum values and iterate backwards to zero. These variables represent our position in the array of arrays. We also have an array that represents the maximum value possible based on the possible paths for each of the columns.

The first thing we do is check the base case where our position is in the bottom right corner of the array of arrays. We put the value into the corresponding column of the possible path array, and set it equal to the maximum value that we have so far since it's the only value.

The second case happens when we're on the bottom row. If we are, we check to see if the number to the right is greater than zero. If it is, we add the number to the right to our current number, and put it into our highest possible column array. We then check the new number against the current highest value and replace it if the new value is higher.

The third case is if we're on the rightmost column. Since there's no value to the right of it, we check to see if the value in the highest value column array is greater than zero. If it is, we add it to the current value and update the highest column value. If not, we just replace the highest column value with itself. After this, we check the new column value against the current highest value and replace it if the new value is greater.

If the current index doesn't fall under any of these cases, we check the current column value against the column value to the column value to the right and see which one is higher. We add the higher value to the current index and replace the current highest column value. We then check the value against the current highest value and replace it if larger.

Essentially we are iterating through the entire array from bottom right corner to the top left. As the algorithm loops around each row from right to left, it is also updating the total maximum value based on the values to the right and below. The maximum value is saved and returned.

Runtime Analysis

$T(n) = O(n^2) + O(n^2) = O(n^2)$

Reading in the lines from the input file and splitting each line into separate elements takes $O(n^2)$ runtime because it runs through n^2 inputs.

The iterative process that calculates the maximum sum also has a runtime of $O(n^2)$ because it iterates through two nested for-loops n times each in order to calculate using every integer input. Overall, the total runtime is $O(n^2)$ because the function is reading a maximum of n^2 inputs at a time.

Correctness of Algorithm

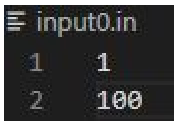
Base case: If n = 1, the board has the dimensions of 1 x 1 and the algorithm returns the only integer input as the max total

Induction hypothesis: We assume that the algorithm holds true for n = k.

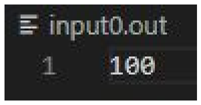
Induction step: Show that n = k + 1 holds true for the algorithm

- Our algorithm holds true for n = k + 1 because now the only difference is that it would be iterating through an array of length $(k+1)^2$
- $(k+1)^2 = k^2 + 2k + 1$ -> this is a subset of k^2 inputs because we can ignore the 2k + 1 in our runtime analysis, which follows with the original $O(n^2)$
- There will not be any difference in the iteration through the two nested for-loops because they would still be looping through a variable number of integers and return a value based on the integers themselves.

Example of BC:

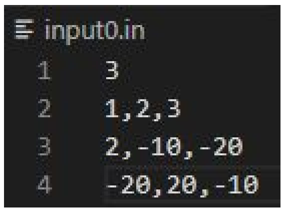


```
input0.in
1
100
```

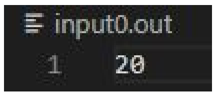


```
input0.out
100
```

Example of IH:

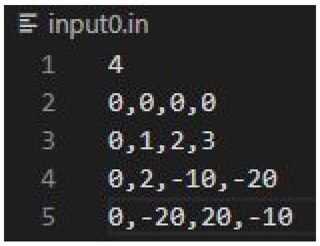


```
input0.in
3
1,2,3
2,-10,-20
-20,20,-10
```

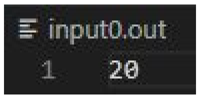


```
input0.out
20
```

Example of IS:



```
input0.in
4
0,0,0,0
0,1,2,3
0,2,-10,-20
0,-20,20,-10
```



```
input0.out
20
```

It can be seen that using n = k + 1 still gives a correct solution. Assuming that n = k holds true for the algorithm and that n = 1 is proven, then by induction we can see that the algorithm holds true for every value of n.