Data Structures, Lab
CRN: 19897

Salvador Robles Herrera
ID: 80683116

Lab #7 Report

Problem 1

This is the Flooded Island problem, where given a list of integers that represent bars in an island, when it rains certain areas of the island fill up with rainwater to form lakes. Note that any excess will run of the island. The problem requires us to return the volume of water that the bars in the island that are held in the lakes. It is needed to use Dynamic Programming.

For this problem I created an array of n+1 zeros, where n is the length of the given list, and then I would iterate over those n elements and populate the array replacing the zeros with the height of the bars including the water trapped between the bars. This is Dynamic Programming from left to right which is going to get the max value between the current height of the bar and the previous heights, which is going to be the previous element in the created array (and that's why we need n+1 elements in the array with the value of zero).

Note that the heights of the bars in the created array will represent the height including the water of the island until the biggest bar, where the next bars will equal to the height of the biggest bar which not in all cases will be the correct height of the bar, that's why we need to do a Dynamic Programming approach from right to left.

From right to left, the same thing will be done but backwards, where a new array is created with the heights of the bars of the island that will be correct until the largest bar is encounter.

Then the minimum height of bar between the correspond element from the created left and right array will be the correct height of the bars including the water from the island, so subtracting the height of the bar without the water will result in the water above the height of the bar, and then the sum of those will result in the total volume water.

The running time of this algorithm will be O(n) since the problem just requires three iterations over the length of the given list, where has constant time operations. The space memory complexity will be O(n) since two lists with n+1 elements was created to solve this problem.

This are the lists that I used, which returned the total volume of water in the lakes between bars. I used the examples from the problem and three problems I created that made me realize that the algorithm works. The examples I created have the biggest bar first found on the left and the right or if there are two bars with the same height and are the biggest.

```
print("What I got: ", flooded_island([1,3,2,4,1,3,1,4,5,2,2,1,4,2,2]), "Ex
pected: 15")
print("What I got: ", flooded_island([0,1,0,2,1,0,1,3,2,1,2,1]), "Expected
: 6")
print("What I got: ", flooded_island([5,2,1,2]), "Expected: 1")
print("What I got: ", flooded_island([5,2,1,2,5]), "Expected: 10")
print("What I got: ", flooded_island([1,3,2,3]), "Expected: 1")
```

Salvador Robles Herrera
ID: 80683116

Problem 2

This is the House Robber problem, where given a list of integers that represent the value obtained by robbing a house, you need to return the maximum total value that you can rob from but having in account that you can not rob from adjacent houses because a security system will automatically contact the police and that is not good for us the robbers. Note that Dynamic Programming is needed to solve this problems.

I created two variables called notRob and doRob. The algorithm will traverse the given list and then it will have two choices, rob hose or not rob house, which will be updated with the maximum value of robbing the houses of robbing the current house or not. At the end it will just return the biggest from the notRob and doRob variables.

To update the doRob variable, it will just get the value of notRob plus the current element in the list this because by robbing the current house you couldn't have robbed the previous hose so the maximum value by robbing houses but by not robbing the previous house from the current house plus robbing the value from the current house, that result will be the value of the maximum gain of robbing the current house.

To update the notRob house, it will be just the value of the maximum gain of not robbing the current house, this is done by getting the max value between the not robbing the previous house or robbing the previous house.

The space time complexity of this algorithm is O(1) since there are only two variables created and, on the traversal, there are constant temporal variables. The running time is O(n) where n is the length of the given list, since there is just a traversal of all the elements with constant number and constant time operations.

This are my test cases, where the first two are provided as examples in the problem, and also I included two other examples which actually made me realize that a part of my algorithm was wrong since I didn't take in account the case where you rob the first and last house without robbing the second and third house, so I modified how to update the "notRob" variable.

```
print("What I got: ", rob_houses([1,2,3,1]), "Expected: 4")
print("What I got: ", rob_houses([2,7,9,3,1]), "Expected: 12")
print("What I got: ", rob_houses([5,2,1,2]), "Expected: 7")
print("What I got: ", rob_houses([5,5,9]), "Expected: 14")
```