Elad Ohayon

The proposed time complexity for my code regarding this assignment is O(n). My assignment consists of one algorithm that conducts a few checks by comparing waterAvalible with waterRequired at each table to produce waterLevel, and O(1) operations such as addition and subtraction inside a while loop. This loop always terminates after iterating the arrays passed in twice due to a boolean value. As such, the while loop is O(n) since it iterates through the array twice, at most (meaning going from 0 to n-1 twice), without moving backwards.

We make the algorithm O(n) by making use of the idea that if we start at table **X** and the water level becomes invalid at table **X + i** meaning the waterLevel drops below 0 then all tables between [1, **X + i**] inclusive are also invalid. In other words, if we start at table 1 and on table 6 the waterLevel drops below 0 then we know tables [2,6] are invalid. This is because when we start at table 1 (assuming table 1 is valid) it is possible to gain an extra W amount of water if waterAvalible>waterRequired and the minimum amount we "enter" table 2 from table 1 is zero water if waterAvalible==waterRequired (we assume table 1-> 2 is valid), but when we start at table 2 we start with 0 water in the reservoir. If the extra **W** water gained at table 1 was not enough to keep waterLevel>0, then table 2 cannot provide us with enough water since $water_{startX} \geq water_{startX-1}$ (assuming table X->X-1) is valid and so on for all tables until **X+i.**

Using this observation, we start at table **X (X** goes from 1 to waterAvalible.length) and iterate through waterAvalible and waterRequired until we reach table **X** again which means table **X** is valid. The second key idea to reducing this algorithm from $O(n^2)$ to O(n) is that we keep track of the current water level at each table by iterating waterAvalible and waterRequired simultaneously and updating our variables likes so. Once we encounter a nonvalid table we know all the tables prior are also invalid, so we don't need to check again from table **X+1.** We do this instead of iterating through waterRequired once for every table in waterAvalible until we hit a valid table which would be $O(n^2)$. As all checks and updates to our pointers are composed of adding or subtracting values and comparisons between waterRequired with waterAvalible using array accesses they are O(1) which is negligible compared to O(n). Overall, this makes the best case of the algorithm O(n) (where table 1 is valid or no table is valid) and worst case 2O(n) if only table **N** is valid since we must iterate through 0 to **N** twice.