PA2

Nicholas Soliman

UIN: 326009195

Q1(Explanation): For this function I used the indexes of the array to determine the values at each child node. So I would iterate through the array until the parent value is found. The index of that item in the array will be one of the children function, then it will continue iterating until it finds another or until there is no more left.

Q2(Explanation): The preorder function first outputs the element inputted then it will output the left element recursively (it will go down the left tree and set that as the new initial node). Then it will look at the right node and recursively call the same function to iterate through the right side and output values. The max depth was a function that was called recursively and added one each time it iterated through each the right and left side. Then finds the max before each iteration. For min depth there were a few base cases that covered the fact that either t was null, or t->left was null, or t->right was null then return certain values. Then it would iterate till either right or left was null then add one to the recursive call. After that it finds the min of the 2 values and process is repeated. Remove left function is very similar to the remove function but it removes the left most node and moves it to an available node with only one child.

Q2(A)- The printTree() function is an in-order traversal.

Q2(B)- It checks to see if the tree is empty or not before it calls the preorder function to output the values.

Q3(Explanation): The diameter function finds the longest route from one node to another. The way I implemented this was first calculating the max height in both the left and right tree. Then finding the diameter of the left and right tree by recursively calling the function. Then I found the max value between the 3: Diameter of left, Diameter of right and the sum of left height and right height.

Q3(A) Time complexity is O(N^2)

Q4(Explanation): For the level order function I used a queue to output the elements for each level. It stores values by pushing and popping then outputs the values from top to bottom then left to right.

Q5(Explanation): The function for the least common ancestor checks 2 major things one of them being x and y are smaller than the t-> element then we know that the LCA lies to the left. And the other, when x and y are greater than t->element then we know it lies to the right. We keep recursively narrowing it down until t-> element is the LCA.

Q5(A): Time complexity of LCA is O(N) where N is the height of the tree.