Q9:

A binary tree is said to be complete if in every level, except possibly the last, is filled, and all nodes are as far left as possible. Construct an effective **recursive** algorithm COMPLETE(x), which given a root node x for a binary tree returns -1 if the tree is not complete and otherwise the height of the tree. Give the time complexity of your solution. Higher marks shall be given for the most efficient and clean algorithm.

- Complete binary tree: every level, except possibly the last, is filled, and all nodes are as far left as possible.

Algorithm:

1) Calculate the number of nodes in the tree.
2) Set the starting index as 0 (root node) [binary tree can be represented in an array]
3) Recursively do the following:
   - If the root is null
     - Return true
   - If the index is >= than number of nodes
     - Return false
4) Recursively we check to the left and right subtrees of the current node. [ ]

Code in Java:

```
int numberOfNodes;
HashMap<Integer, Integer> index= new HashMap<Integer, Integer>();
int maxIndex=-1;

int count(Node x){
      if ( x == null){
            return 0;
      return 1 + count(x.left) + count(x.right);
}

boolean isCOMPLETE(Node x){
      if ( x == null )
            return true;
      if ( index[x] >= numberOfNodes )
            return false;
      index[x.left] = index[x]+1;
      index[x.right] = index[x]+2;

      maxIndex = math.max(index[x],maxIndex); // to print the height
      return ( COMPLETE(x.left) && COMPLETE(x.right);
}

int COMPLETE(Node x){

      if ( isCOMPLETE(x) )
            return (int) (Math.log(maxIndex) / Math.log(2)); // height
      else
            return -1;

}
```

To run the code:

1) First calculate the number of nodes by calling count(x).
2) Run COMPLETE.

Time Complexity: All the nodes are traversed at most once, so the overall complexity would be O(n).