Question 1.
The running time order of child function is O(N).
In this function, I just go over the Parent array and find out if there are some elements in it the same as the integer given, if so, then the index of the element will show which node is the children of the node with the given integer. Thus, the time complexity is O (N).

Question 2.
2.1) Method printTree is inorder traversal. Because it first prints the left child and then the element, last print the right children.

2.3) When we call preorder function in q2.cpp, it will print the whole tree in preorder traversal. This function will first print the element in the node, and then calls itself twice for node's left child and right child. This function will recursively call itself until the node is null.

2.5) AVL tree is a binary search tree with a balance condition, so the maximum difference between max and min depth of the tree is 2.

```
Tree1 max= 38 min= 7 diff= 31
```

After adding 100000 random nodes, the difference becomes 31 in this test.

Question 3.
The Space complexity is O(n)
As I create 5 variables in this recursive function, each time it was called, the space plus 5. As a result, the Space complexity is O (5n) = O(n)

The Time complexity is O (n)
The worst case in this function is go through the whole tree, thus, the time complexity of this function is O (n).

Question 5.
The Space complexity is O (log n)
I wrote another function called WayToNode to help get the path from root to the node and push all data on the way to a vector. So, the space complexity for this function is the height of the tree, which is O (log n). And I call this function twice in LCA to get the path of both nodes, so the space complexity for LCA should be 2*log n, which is O (log n).

The Time complexity is $O((\log n)^2)$
As function WayToNode pushes all data on the way from root to the node to a vector, the time complexity is O (log n). And I call this function twice in LCA, which make the time complexity 2*log n. There is also a nested for loop in this function, so the time complexity for this function is $(\log\ n)^2$. This makes the time complexity for the whole function become $O(2 * \log n + (\log n)^2) = O(\log n + \ (\log n)^2) = O((\log n)^2)$