Generate binary trees using permutations

Given a random permutation, generate the corresponding rooted binary tree. A binary tree has at most two child nodes, with the value of the left child less than its parent and the value of the right node greater than its parent. The permutation is a list of integers like `[1,3,2,0]` where there are *no* repeated values. The value of the root of this tree is `1`. Proceeding from left to right add the integers from this sequence into the binary tree. For example, each node in your tree can be represented by the following class

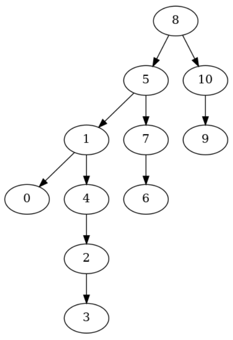
```
class Node:
    def __init__(self, val:int=0, left=None, right=None):
        assert isinstance(left,Node) or left is None
        assert isinstance(right,Node) or right is None
        self.val = val
        self.left = left
        self.right = right
```

Your function `get_tree_list(p:List[int])` takes a permutation `p` as input and returns a list that represents the binary tree. The index of each item in the list is the value of each node in your tree and the value at each index is the parent of that node. Recall that in a tree, every node has exactly one parent. Here's an example:

```
>>> p = [8, 5, 1, 10, 0, 4, 2, 3, 7, 9, 6]
>>> get_tree_list(p)
[1, 5, 4, 2, 1, 8, 7, 5, 8, 10, 8]
```

Notice that the value of `8` is at the eighth index in the list. This indicates that `8` is the root node. The zero-th item in the list is `1`. This means that the parent of `0` is `1`. Likewise, the `1` th element's parent is `5`, and so on.

Out[33]:


8
5
10
1
7
9
0
4
6
2
3