Since we know how post-order traversal is implemented, we can be sure that the last element in the array is the root, in our case 8. We then split the array into a left subtree with elements before the root (9,5,1,7,2,12) and a right subtree with elements after the right (4,3,11).

We then look for the first element in the left subtree that shows up in the post-order traversal sequence from the right. In our case, it is 5. We know for sure that this is root of the left subtree. Now, we want the left and right of this root. We look at the index of this root in the in-order sequence, with the left being whatever is to the left of the index, and the right being whatever is to the right. In our case, left = 9, right = 1,7,2,12.

```
          8

        /

      5

   /    \

9          ( 1,7,2,12)(not yet in nodes)
```

We then look for the next number in the right subtree that appears in the post-order sequence from the right. We have 7.

```
          8

        /

      5

   /    \

9          7
```

We check the index of 7 in the in-order sequence, we see that 1 is to the left. We add that as the left of node 7, and 2,12 as the right of node 7.

```
          8

        /

      5

   /    \

9          7

          /  \

        1      (2,12)(not yet in nodes)
```

We now look for the first number in the right subtree that appears in the post-order sequence, 12. We then look at the index of 12 in the in-order sequence, and see that 2 is to the left of it.

```
            8
        /
      5
    /     \
0            7
           /   \
          1       12
                 /
                2
```

Since this is a recursive method, it will do the same logic with the right subtrees, first looking at the index of the number compared to the in-order sequence. We end up with

```
               8
        /           \
      5                 4
    /     \                \
9            7                11
           /   \              /
          1       12       3
                 /
                2
```