

Print Postorder traversal from given Inorder and Preorder traversals

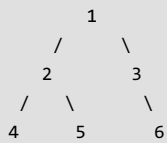
Given Inorder and Preorder traversals of a binary tree, print Postorder traversal.

Example:

```
Input:
Inorder traversal in[] = {4, 2, 5, 1, 3, 6}
Preorder traversal pre[] = {1, 2, 4, 5, 3, 6}
```

```
Output:
Postorder traversal is {4, 5, 2, 6, 3, 1}
```

Trversals in the above example represents following tree



A **naive method** is to first construct the tree, then use simple recursive method to print postorder traversal of the constructed tree.

We can print postorder traversal without constructing the tree. The idea is, root is always the first item in preorder traversal and it must be the last item in postorder traversal. We first recursively print left subtree, then recursively print right subtree. Finally, print root. To find boundaries of left and right subtrees in pre[] and in[], we search root in in[], all elements before root in in[] are elements of left subtree and all elements after root are elements of right subtree. In pre[], all elements after index of root in in[] are elements of right subtree. And elements before index (including the element at index and excluding the first element) are elements of left subtree.

```

// C++ program to print postorder traversal from preorder and inorder traversals
#include <iostream>
using namespace std;

// A utility function to search x in arr[] of size n
int search(int arr[], int x, int n)
{
    for (int i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

// Prints postorder traversal from given inorder and preorder traversals
void printPostOrder(int in[], int pre[], int n)
{
    // The first element in pre[] is always root, search it
    // in in[] to find left and right subtrees
    int root = search(in, pre[0], n);

    // If left subtree is not empty, print left subtree
    if (root != 0)
        printPostOrder(in, pre+1, root);

    // If right subtree is not empty, print right subtree
    if (root != n-1)
        printPostOrder(in+root+1, pre+root+1, n-root-1);

    // Print root
    cout << pre[0] << " ";
}

// Driver program to test above functions
int main()
{
    int in[] = {4, 2, 5, 1, 3, 6};
    int pre[] = {1, 2, 4, 5, 3, 6};
    int n = sizeof(in)/sizeof(in[0]);
    cout << "Postorder traversal " << endl;
    printPostOrder(in, pre, n);
    return 0;
}

```

Output

```

Postorder traversal
4 5 2 6 3 1

```

Time Complexity: The above function visits every node in array. For every visit, it calls search which takes $O(n)$ time. Therefore, overall time complexity of the function is $O(n^2)$