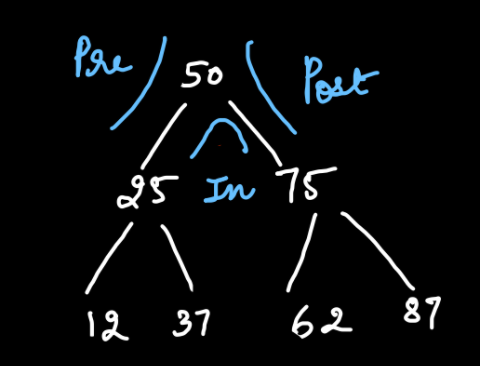
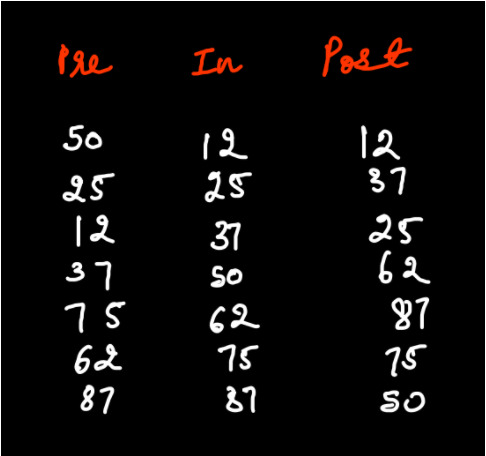
**1. Problem Discussion:**

You have already learnt previously what Pre, Post and In order are.

****

As seen in figure 1, the left side of an element is the pre area, the right side of an element is the "post" area and the area in between an element is the "in" area. Hence when we traverse through the tree given in figure 1, the order of elements for pre, in and post traversal are depicted by figure 2.

****

You are required to complete the body of "iterativePrePostInTraversal" function. The function is expected to print pre order, in order and post order of the tree in separate lines (first pre, then in and finally post order). All elements in an order must be separated by a space.

**2. Approach :**

We are going to solve this problem by using a Pair class which has 2 data members: node and state. And with the help of the same example which we are initially using in this article. Here, State =1 means Pre order State =2 means In order State =3 means Post order

1• Initially, when the stack is empty we push the root which has a state 1.

2• Then we check the "state" of top.

3• Here, the state of the top 50 is 1. Which means it is in Pre order. Hence we add top to the "pre" string and push the left node of this top i.e. 25 in the stack with state 1. Simultaneously the state of 50 is increased by one and becomes 2.

4• Now the top of the stack is 25 with state 1. Again the above process is followed and 25 is added to the "pre" string. We push the left node of 25 i.e. 12 to the stack and change the state of 25 to 2.

5• Now, the top of the stack, 25 has no left child, so we add it to the "pre" string and directly increase its state.

6• The top of the stack is again 25. Its state is 2 which means it is in inorder. Hence, we add it to the "In" string and go to the right node which doesn't exist. Hence, we again increase the state by one.

7• The state of 25 is 3 here. It means it is in post order. So, we add it to the "post" string and pop it from the stack.

8• These processes are repeated for every top element till the stack is empty.

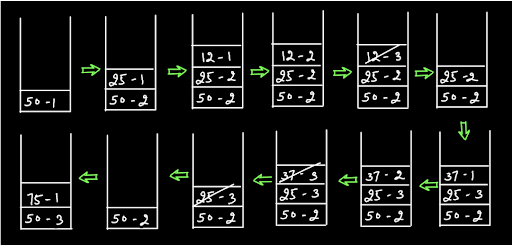
****

Figure 3 represents the stacking process of half of the tree. Can you complete the stacking process for the other half on your own?

To summarize, State 1 -> add to "Pre", state++, left node State 2 ->add to "In", state++, right node State 3 ->add to "Post", pop

If you have gone through the algorithm and its summary carefully, then you will have no problem in coding this problem. We want you to at least give it a try. If you face any obstacles, revise the algorithm and then refer to the code given below.

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class Node {

int data;

Node left;

Node right;

Node(int data, Node left, Node right) {

this.data = data;

this.left = left;

this.right = right;

}

}

public static class Pair {

Node node;

int state;

Pair(Node node, int state) {

this.node = node;

this.state = state;

}

}

public static Node construct(Integer[] arr) {

Node root = new Node(arr[0], null, null);

Pair rtp = new Pair(root, 1);

Stack< Pair> st = new Stack< >();

st.push(rtp);

int idx = 0;

while (st.size() > 0) {

Pair top = st.peek();

if (top.state == 1) {

idx++;

if (arr[idx] != null) {

top.node.left = new Node(arr[idx], null, null);

Pair lp = new Pair(top.node.left, 1);

st.push(lp);

} else {

top.node.left = null;

}

top.state++;

} else if (top.state == 2) {

idx++;

if (arr[idx] != null) {

top.node.right = new Node(arr[idx], null, null);

Pair rp = new Pair(top.node.right, 1);

st.push(rp);

} else {

top.node.right = null;

}

top.state++;

} else {

st.pop();

}

}

return root;

}

public static void display(Node node) {

if (node == null) {

return;

}

String str = "";

str += node.left == null ? "." : node.left.data + "";

str += " <- " + node.data + " -> ";

str += node.right == null ? "." : node.right.data + "";

System.out.println(str);

display(node.left);

display(node.right);

}

public static void iterativePrePostInTraversal(Node node) {

Stack < Pair> st = new Stack< >();

Pair rtp = new Pair(node, 1);

st.push(rtp);

String pre = "";

String in = "";

String post = "";

while (st.size() > 0) {

Pair top = st.peek();

if (top.state == 1) { //pre,s++,left

pre += top.node.data + " ";

top.state++;

if (top.node.left != null) {

Pair lp = new Pair(top.node.left, 1);

st.push(lp);

}

}

else if (top.state == 2) { //in,s++,right

in += top.node.data + " ";

top.state++;

if (top.node.right != null) {

Pair rp = new Pair(top.node.right, 1);

st.push(rp);

}

}

else { //post,pop

post += top.node.data + " ";

st.pop();

}

}

System.out.println(pre);

System.out.println(in);

System.out.println(post);

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

Integer[] arr = new Integer[n];

String[] values = br.readLine().split(" ");

for (int i = 0; i < n; i++) {

if (values[i].equals("n") == false) {

arr[i] = Integer.parseInt(values[i]);

} else {

arr[i] = null;

}

}

Node root = construct(arr);

iterativePrePostInTraversal(root);

}

}

Analysis:

Time Complexity: O(n) The time complexity is linear due to the traversals in the Binary Tree. Space Complexity: O(n) The space complexity is linear due to the use of stack space. We also suggest you watch the solution video of this question to understand it better.