**Tree Traversals**

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Example Tree

Depth First Traversals:  
(a) Inorder  
(b) Preorder  
(c) Postorder

Breadth First or Level Order Traversal  
Please see [this](http://geeksforgeeks.org/?p=2686) post for Breadth First Traversal.

**Inorder Traversal:**

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Uses of Inorder  
In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder itraversal s reversed, can be used.  
Example: Inorder traversal for the above given figure is 4 2 5 1 3.  
 **Preorder Traversal:**

Algorithm Preorder(tree)

1. Visit the root.

2. Traverse the left subtree, i.e., call Preorder(left-subtree)

3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Uses of Preorder  
Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree. Please see <http://en.wikipedia.org/wiki/Polish_notation> to know why prefix expressions are useful.  
Example: Preorder traversal for the above given figure is 1 2 4 5 3.  
 **Postorder Traversal:**

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)

2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.

Uses of Postorder  
Postorder traversal is used to delete the tree. Please see [the question for deletion of tree](http://geeksforgeeks.org/?p=654) for details. Postorder traversal is also useful to get the postfix expression of an expression tree. Please see <http://en.wikipedia.org/wiki/Reverse_Polish_notation> to for the usage of postfix expression.

Example: Postorder traversal for the above given figure is 4 5 2 3 1.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>    /\* A binary tree node has data, pointer to left child     and a pointer to right child \*/  struct node  {       int data;       struct node\* left;       struct node\* right;  };    /\* Helper function that allocates a new node with the     given data and NULL left and right pointers. \*/  struct node\* newNode(int data)  {       struct node\* node = (struct node\*)                                    malloc(sizeof(struct node));       node->data = data;       node->left = NULL;       node->right = NULL;         return(node);  }    /\* Given a binary tree, print its nodes according to the    "bottom-up" postorder traversal. \*/  void printPostorder(struct node\* node)  {       if (node == NULL)          return;         // first recur on left subtree       printPostorder(node->left);         // then recur on right subtree       printPostorder(node->right);         // now deal with the node       printf("%d ", node->data);  }    /\* Given a binary tree, print its nodes in inorder\*/  void printInorder(struct node\* node)  {       if (node == NULL)            return;         /\* first recur on left child \*/       printInorder(node->left);         /\* then print the data of node \*/       printf("%d ", node->data);         /\* now recur on right child \*/       printInorder(node->right);  }    /\* Given a binary tree, print its nodes in preorder\*/  void printPreorder(struct node\* node)  {       if (node == NULL)            return;         /\* first print data of node \*/       printf("%d ", node->data);         /\* then recur on left sutree \*/       printPreorder(node->left);         /\* now recur on right subtree \*/       printPreorder(node->right);  }    /\* Driver program to test above functions\*/  int main()  {       struct node \*root  = newNode(1);       root->left             = newNode(2);       root->right           = newNode(3);       root->left->left     = newNode(4);       root->left->right   = newNode(5);         printf("\n Preorder traversal of binary tree is \n");       printPreorder(root);         printf("\n Inorder traversal of binary tree is \n");       printInorder(root);         printf("\n Postorder traversal of binary tree is \n");       printPostorder(root);         getchar();       return 0;  } |

**Time Complexity:** O(n)  
Let us prove it:

Complexity function T(n) — for all problem where tree traversal is involved — can be defined as:

T(n) = T(k) + T(n – k – 1) + c

Where k is the number of nodes on one side of root and n-k-1 on the other side.

Let’s do analysis of boundary conditions

Case 1: Skewed tree (One of the subtrees is empty and other subtree is non-empty )

k is 0 in this case.  
T(n) = T(0) + T(n-1) + c  
T(n) = 2T(0) + T(n-2) + 2c  
T(n) = 3T(0) + T(n-3) + 3c  
T(n) = 4T(0) + T(n-4) + 4c

…………………………………………  
………………………………………….  
T(n) = (n-1)T(0) + T(1) + (n-1)c  
T(n) = nT(0) + (n)c

Value of T(0) will be some constant say d. (traversing a empty tree will take some constants time)

T(n) = n(c+d)  
T(n) = (-)(n) (Theta of n)

Case 2: Both left and right subtrees have equal number of nodes.

T(n) = 2T(|\_n/2\_|) + c

This recursive function is in the standard form (T(n) = aT(n/b) + (-)(n) ) for master method <http://en.wikipedia.org/wiki/Master_theorem>. If we solve it by master method we get (-)(n)

**Auxiliary Space :** If we don’t consider size of stack for function calls then O(1) otherwise O(n).