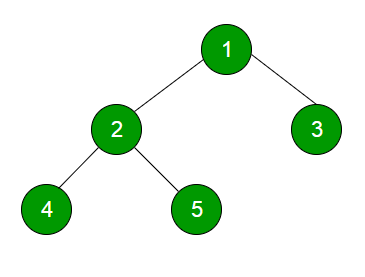
Level Order Tree Traversal

Level order traversal of a tree is [breadth first traversal f](http://en.wikipedia.org/wiki/Breadth-first_traversal)or the tree.



Level order traversal of the above tree is 1 2 3 4 5

[**Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.**](https://practice.geeksforgeeks.org/problems/level-order-traversal/1)

**Method 1 (Use function to print a given level)**

**Algorithm:**  
There are basically two functions in this method. One is to print all nodes at a given level (printGivenLevel), and other is to print level order traversal of the tree (printLevelorder). printLevelorder makes use of printGivenLevel to print nodes at all levels one by one starting from root.

/\*Function to print level order traversal of tree\*/

**printLevelorder(tree)**

for d = 1 to height(tree)

printGivenLevel(tree, d);

/\*Function to print all nodes at a given level\*/

**printGivenLevel(tree, level)**

if tree is NULL then return;

if level is 1, then

print(tree->data);

else if level greater than 1, then

printGivenLevel(tree->left, level-1);

printGivenLevel(tree->right, level-1);

**Implementation:**

* C++
* C
* Java
* Python
* C#

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Recursive C program for level order traversal of Binary Tree  #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, \*right;  };    /\* Function protoypes \*/  void printGivenLevel(struct node\* root, int level);  int height(struct node\* node);  struct node\* newNode(int data);    /\* Function to print level order traversal a tree\*/  void printLevelOrder(struct node\* root)  {      int h = height(root);      int i;      for (i=1; i<=h; i++)          printGivenLevel(root, i);  }    /\* Print nodes at a given level \*/  void printGivenLevel(struct node\* root, int level)  {      if (root == NULL)          return;      if (level == 1)          printf("%d ", root->data);      else if (level > 1)      {          printGivenLevel(root->left, level-1);          printGivenLevel(root->right, level-1);      }  }    /\* Compute the "height" of a tree -- the number of      nodes along the longest path from the root node      down to the farthest leaf node.\*/  int height(struct node\* node)  {      if (node==NULL)          return 0;      else      {          /\* compute the height of each subtree \*/          int lheight = height(node->left);          int rheight = height(node->right);            /\* use the larger one \*/          if (lheight > rheight)              return(lheight+1);          else return(rheight+1);      }  }    /\* 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);  }    /\* 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("Level Order traversal of binary tree is \n");      printLevelOrder(root);        return 0;  } |

**Output:**

Level order traversal of binary tree is -

1 2 3 4 5

**Time Complexity:** O(n^2) in worst case. For a skewed tree, printGivenLevel() takes O(n) time where n is the number of nodes in the skewed tree. So time complexity of printLevelOrder() is O(n) + O(n-1) + O(n-2) + .. + O(1) which is O(n^2).  
**Space Complexity:** O(n) in worst case. For a skewed tree, printGivenLevel() uses O(n) space for call stack. For a Balanced tree, call stack uses O(log n) space, (i.e., height of the balanced tree).

**Method 2 (Using queue)**

**Algorithm:**  
For each node, first the node is visited and then it’s child nodes are put in a FIFO queue.

printLevelorder(tree)

1) Create an empty queue q

2) temp\_node = root /\*start from root\*/

3) Loop while temp\_node is not NULL

a) print temp\_node->data.

b) Enqueue temp\_node’s children (first left then right children) to q

c) Dequeue a node from q and assign it’s value to temp\_node

**Implementation:**  
Here is a simple implementation of the above algorithm. Queue is implemented using an array with maximum size of 500. We can implement queue as linked list also.

* C++
* C
* Java
* Python
* C#

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Iterative Queue based C program to do level order traversal  // of Binary Tree  #include <stdio.h>  #include <stdlib.h>  #define MAX\_Q\_SIZE 500    /\* 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;  };    /\* frunction prototypes \*/  struct node\*\* createQueue(int \*, int \*);  void enQueue(struct node \*\*, int \*, struct node \*);  struct node \*deQueue(struct node \*\*, int \*);    /\* Given a binary tree, print its nodes in level order     using array for implementing queue \*/  void printLevelOrder(struct node\* root)  {      int rear, front;      struct node \*\*queue = createQueue(&front, &rear);      struct node \*temp\_node = root;        while (temp\_node)      {          printf("%d ", temp\_node->data);            /\*Enqueue left child \*/          if (temp\_node->left)              enQueue(queue, &rear, temp\_node->left);            /\*Enqueue right child \*/          if (temp\_node->right)              enQueue(queue, &rear, temp\_node->right);            /\*Dequeue node and make it temp\_node\*/          temp\_node = deQueue(queue, &front);      }  }    /\*UTILITY FUNCTIONS\*/  struct node\*\* createQueue(int \*front, int \*rear)  {      struct node \*\*queue =          (struct node \*\*)malloc(sizeof(struct node\*)\*MAX\_Q\_SIZE);        \*front = \*rear = 0;      return queue;  }    void enQueue(struct node \*\*queue, int \*rear, struct node \*new\_node)  {      queue[\*rear] = new\_node;      (\*rear)++;  }    struct node \*deQueue(struct node \*\*queue, int \*front)  {      (\*front)++;      return queue[\*front - 1];  }    /\* 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);  }    /\* 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("Level Order traversal of binary tree is \n");      printLevelOrder(root);        return 0;  } |

**Output:**

Level order traversal of binary tree is -

1 2 3 4 5

**Time Complexity:** O(n) where n is number of nodes in the binary tree  
**Space Complexity:** O(n) where n is number of nodes in the binary tree