Dear reader, welcome to the article on the topic named **‘**[**Display a Generic Tree**](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/display-generic-tree/video)**’.**

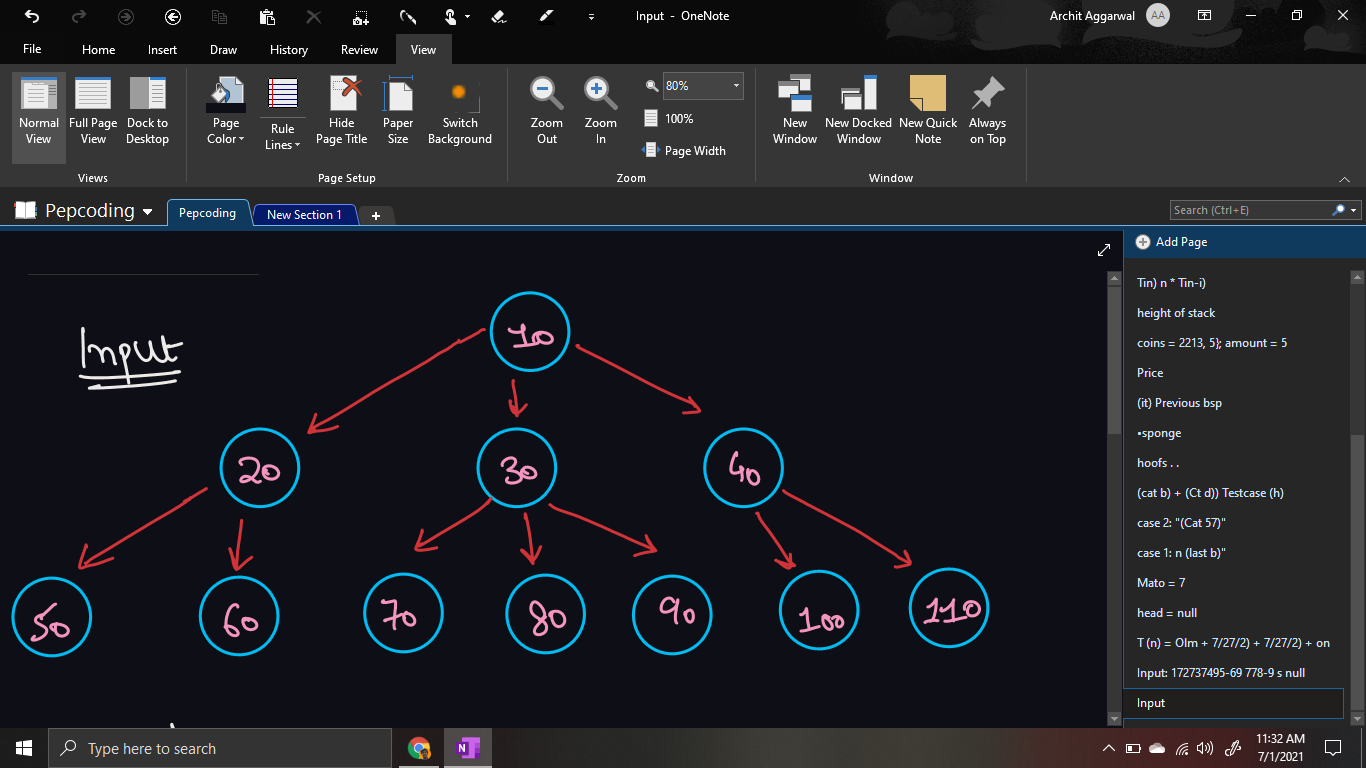
This article will discuss what all have been discussed in the video, and hence, can act as a quick revision resource for you.

If you do not know about the Generic Trees, then I recommend to you to first watch **‘**[**Introduction**](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/generic-trees-intro-official/video)**’** and [**‘Constructor’**](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/generic-tree-const-official/video)videos before moving ahead.

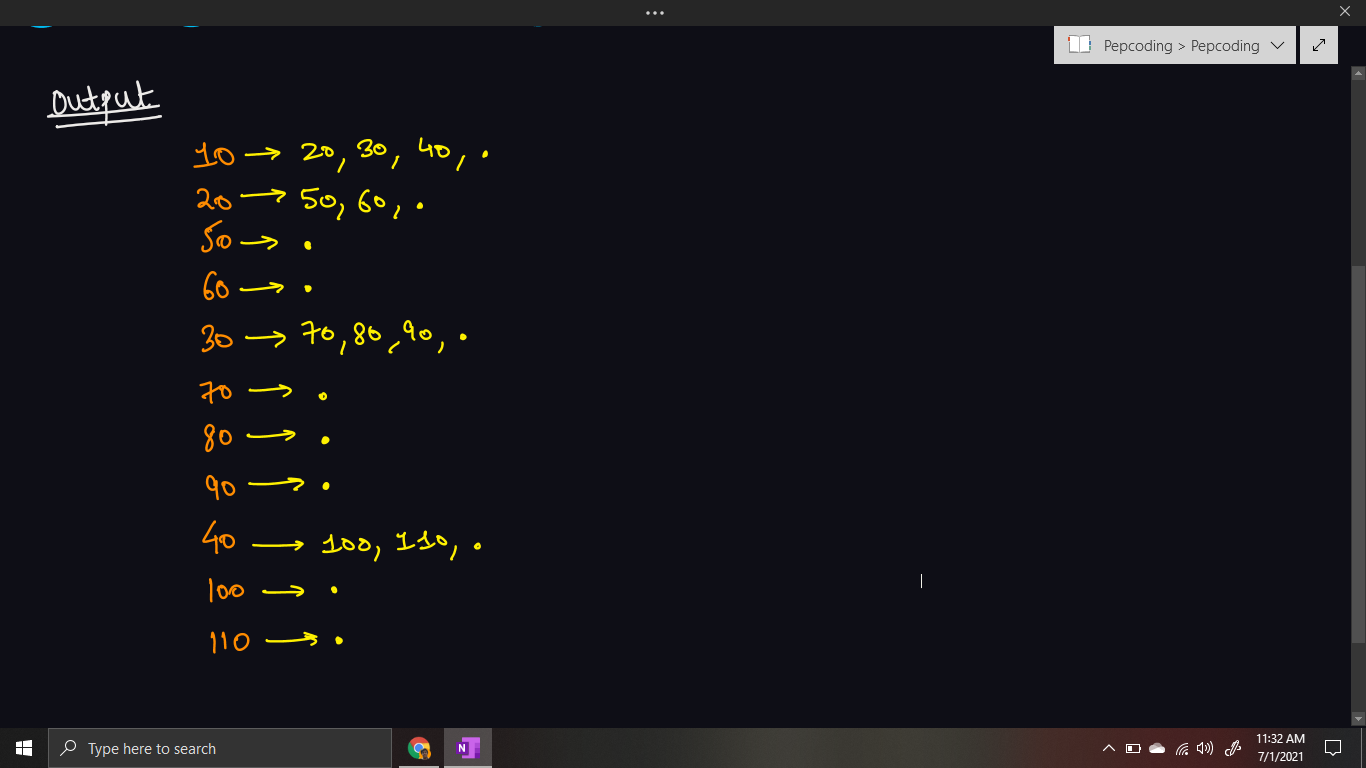
***Problem Statement:***

We need to display all the nodes of a generic tree. In particular, we need to print all the children corresponding to every node. The nodes should be printed in the same order in which we do euler traversal (preorder traversal).

For example for the following generic tree:



The output should be:



***Solution:***

This problem can be solved using the concept of ***recursion***, which we all know by heart now. To solve any problem of recursion, we need to define three aspects: faith, expectation, recursive relation to meet faith with expectation. Let us analyze the ***high-level thinking*** procedure:

**Expectation:** We have expectation with our function ***display(root)*** that it will display the entire tree, rooted at node root.

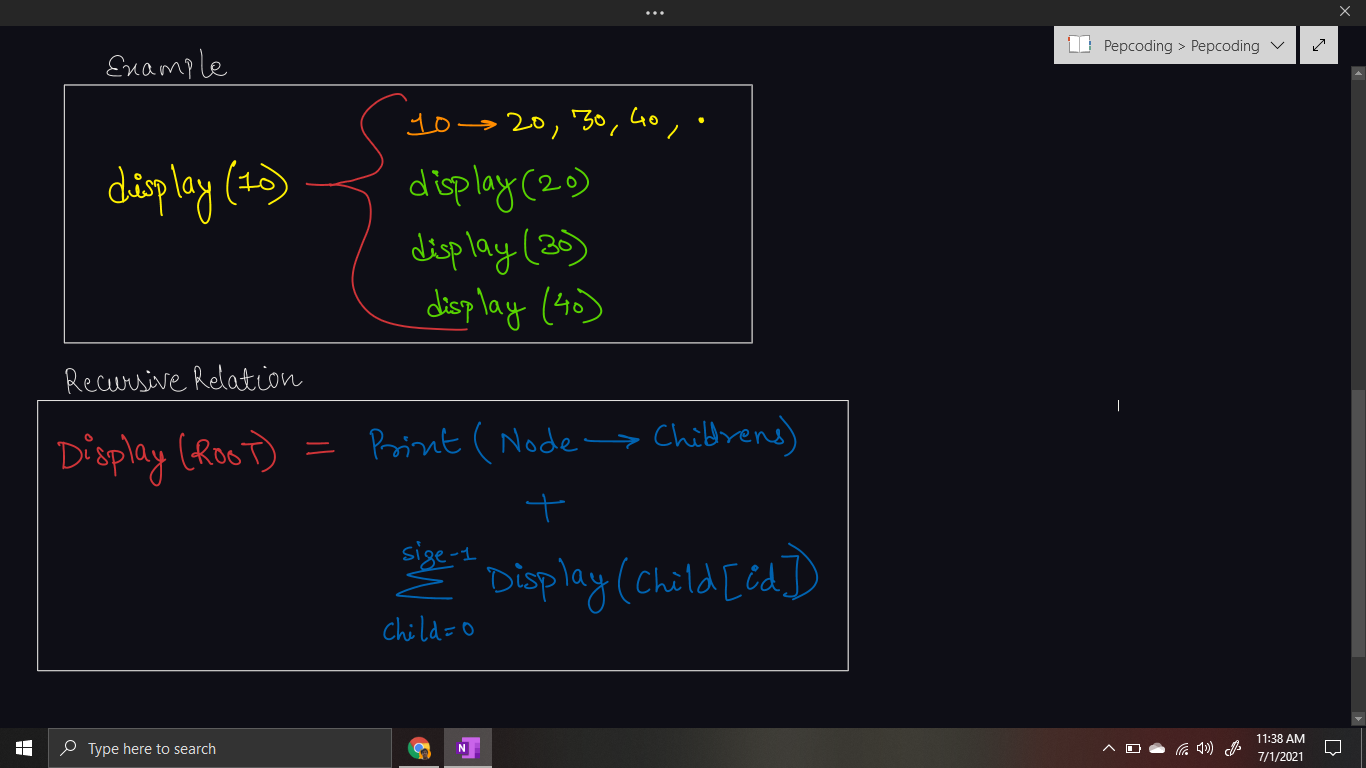
**Faith**: We must put our faith on the function ***display(child)*** that it will print/display the subtree rooted at node child.

**Meeting Expectation with Faith:**

Now, how can we use the faith defined to meet our expectations?

* Since, we need to print all the children nodes corresponding to each node, we will create a string which will store root.data + “-> “ initially (in ***node pre area***).
* Now, we will run a loop over all the children nodes and add their values to the string, separated by commas.
* In the end of the string, we will also append the dot (.) character, as needed in the expected output. We can print the current string.
* Standing at any node, we can call for all of its children and have faith that they will display all the subtrees below them.
* We can call the display function for all of the node’s children by running a for loop over the children array.

Hence, the recursive relation can be:



Only thing remaining is handling the ***base case***.

* When there are no children of the current node, then we can simply return from the recursive function.
* However, this base case will also be handled by the for loop itself, as for 0 size of children array, it will not stop in the first iteration only.

***Pseudo Code/ Algorithm***

* Initialize a String *str = node.data + “-> “*.
* Run a for loop for all the children and append child.data and “, ” to the string.
* Finally, append “.” to the string and print the result.
* Now, recursively call the display function for all the children nodes.

***Implementation***

*Note*: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

import java.io.\*;

import java.util.\*;

public class Main {

private static class Node {

int data;

ArrayList<Node> children = new ArrayList<>();

}

public static void display(Node node) {

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

for (Node child : node.children) {

str += child.data + ", ";

}

str += ".";

System.out.println(str);

for (Node child : node.children) {

display(child);

}

}

public static Node construct(int[] arr) {

Node root = null;

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

for (int i = 0; i < arr.length; i++) {

if (arr[i] == -1) {

st.pop();

} else {

Node t = new Node();

t.data = arr[i];

if (st.size() > 0) {

st.peek().children.add(t);

} else {

root = t;

}

st.push(t);

}

}

return root;

}

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

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

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

int[] arr = new int[n];

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

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

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

}

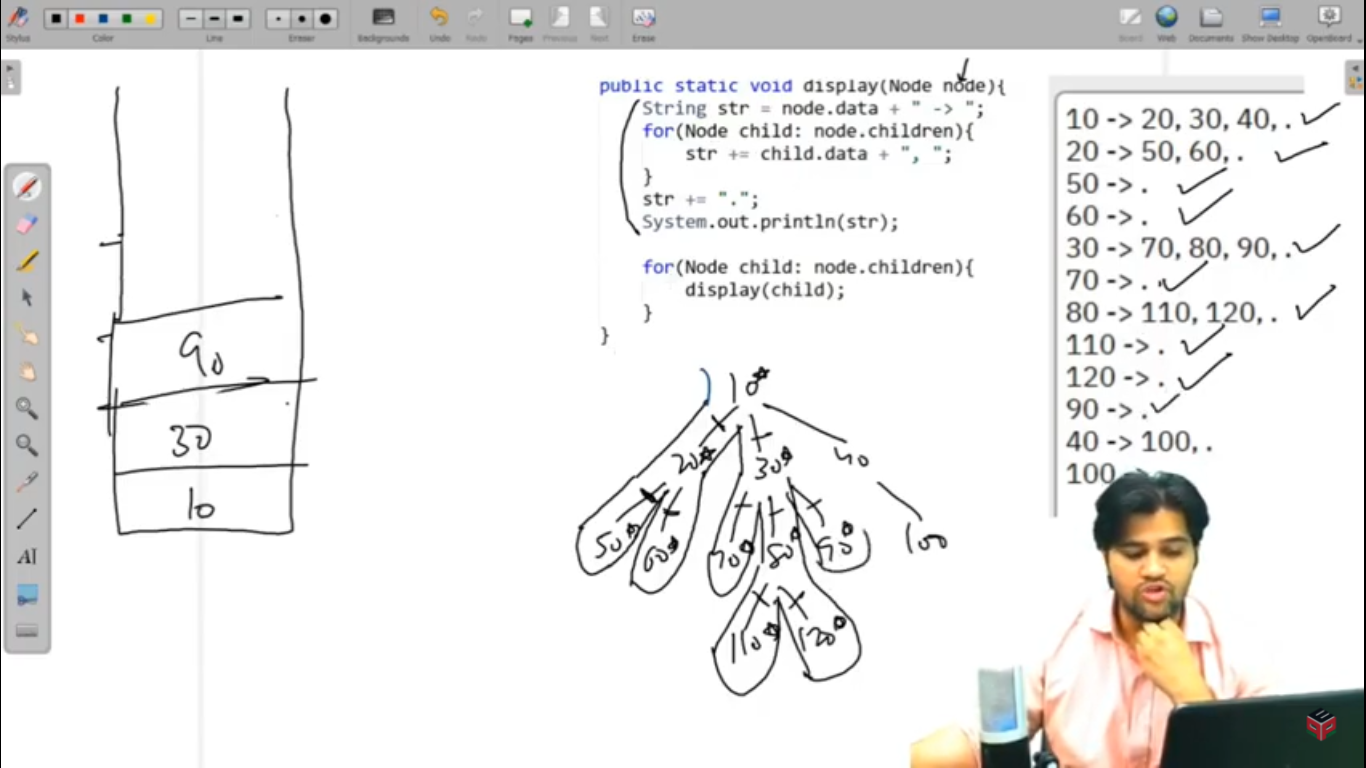
Node root = construct(arr);

display(root);

}

}

I suggest you draw an euler tree running your code line by line and see how recursion works. Drawing a recursion tree will help develop ***low-level thinking***. It is explained using an example in the [same video](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/display-generic-tree/video).



* What is the ***time complexity*** of the above code?

We are traversing the entire tree once, i.e. we are making a recursive call for display of each node, hence the total time complexity is ***O(n)*** where n = number of nodes in the generic tree.

* What is the ***space complexity*** of the above code?

We are not taking any auxiliary data structure, hence extra space used is O(1).

However, we are using recursion, which does take recursion call stack space. The recursion can grow to the maximum depth of the tree. Hence, the recursive stack will take ***O(d) space*** where d = depth of tree.

*Note*: In the worst case, the depth of the tree can be equal to the number of nodes in the tree, if all nodes are linearly arranged. Hence, the recursion call stack can take ***O(n) space*** in the ***worst*** case.

Hope that you liked the article on *Display A Generic Tree*.

Subscribe to Pepcoding’s youtube channel for more such amazing content on Data Structures & Algorithms and follow the resources available for all students in the resources section of Pepcoding’s website.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by: [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)