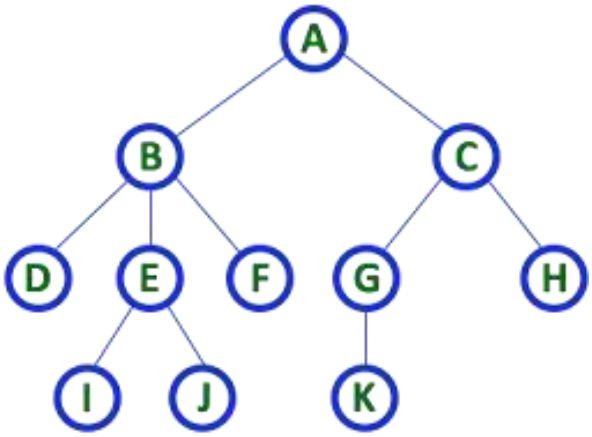
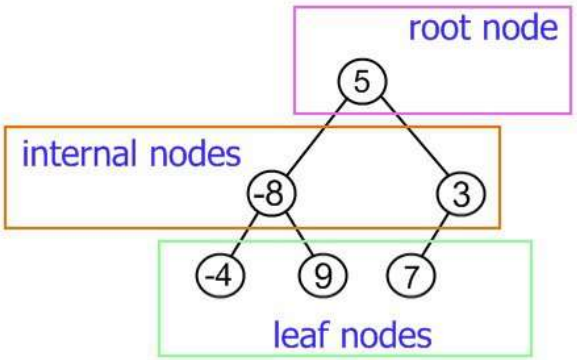
Tree Concept

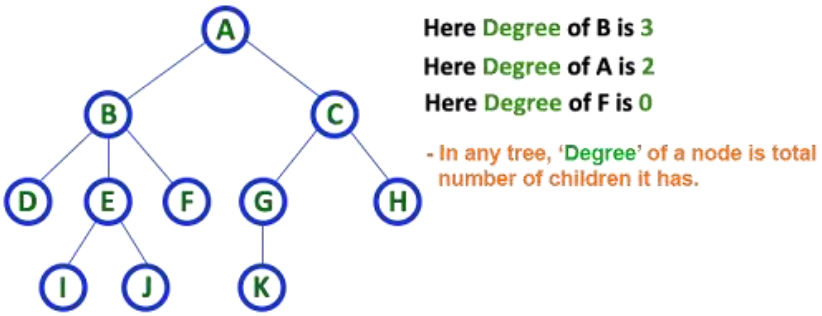
- Tree is a non-linear data structure that consists of nodes and is connected by **edges**.
- Trees allows easier and quicker access to the data.
- Tree terminology:



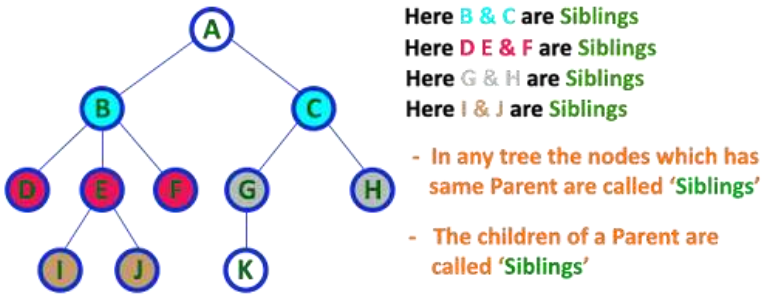
1. **Nodes**, just like linked list, contain data and pointer
2. **Edges**, connecting lines between nodes
3. **Root**, the first node of a tree
4. **Parent node**, a node's ancestor
5. **Child node**, a node's descendant
6. **Leaf node**, the node that doesn't have any children
7. **Internal node**, any node that at least has 1 child can be define as an internal node



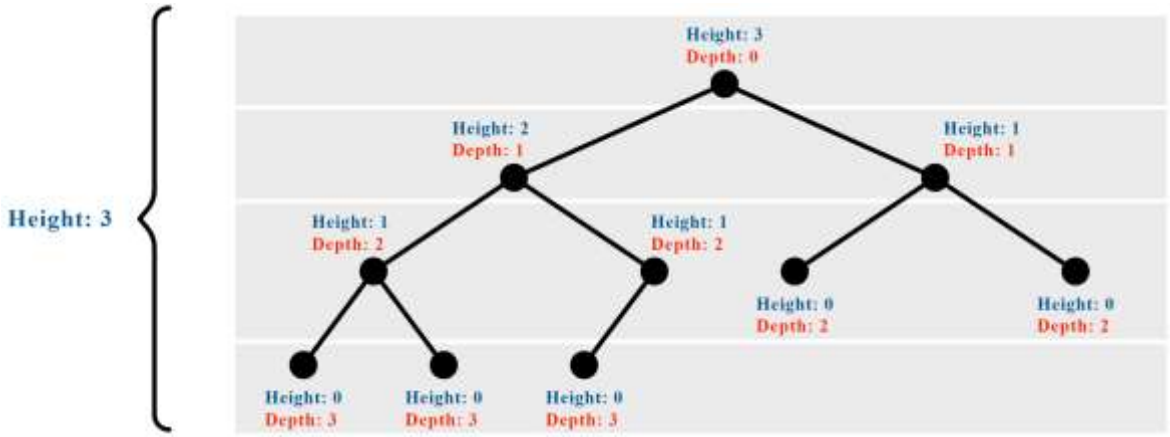
8. **Degree**, total child that any node has



9. **Siblings**, every node that has the same parent

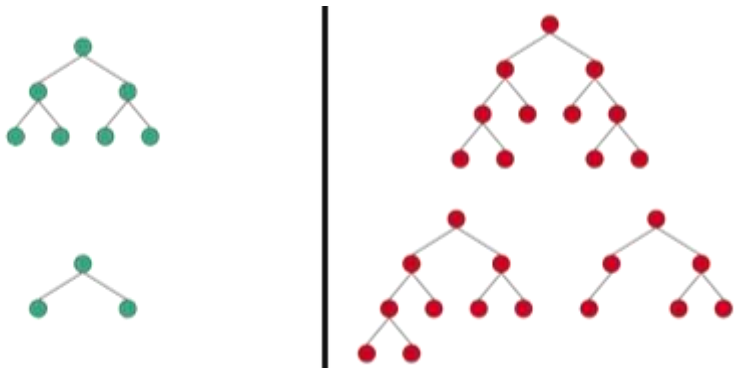


10. **Height/Depth**, the concept of measuring the distance from a node in a tree data structure to the root node or the farthest leaf node, respectively.

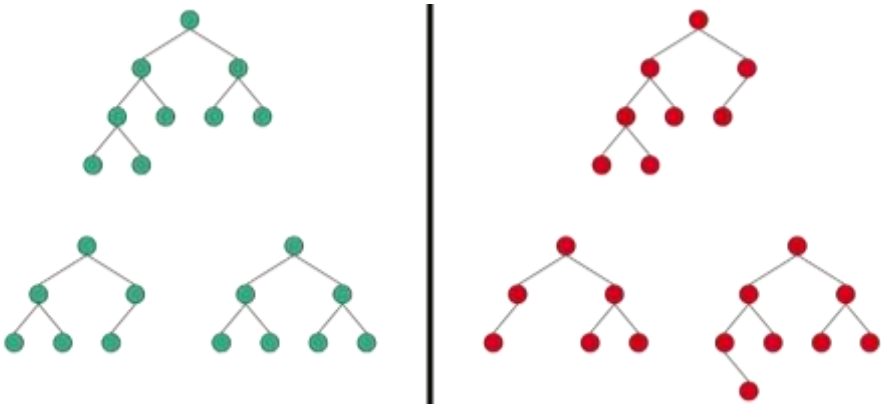


- Binary Tree:
  1. A tree that has a maximum of 2 children.
  2. Those 2 children usually called **left** and **right** child.
  3. It is possible that a binary tree only has 1 child.
  4. Example of a binary tree is right above (10.**Height/Depth**).

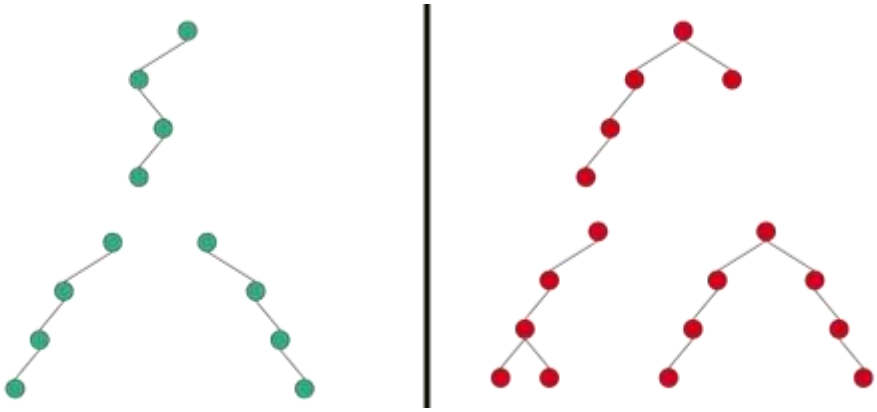
- Type of Binary Tree:
  1. **PERFECT**, all internal nodes have 2 children and all the leaf nodes are at the same depth or same level.



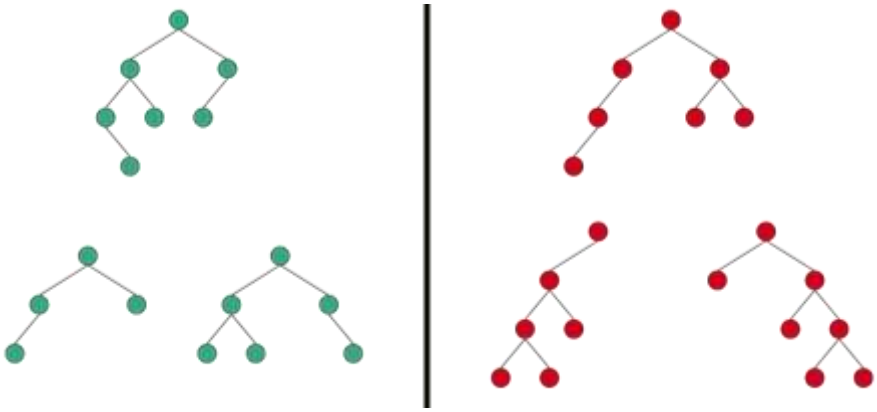
2. **COMPLETE**, all levels completely filled with nodes except the last level and in the last level, all the nodes are as left side as possible.



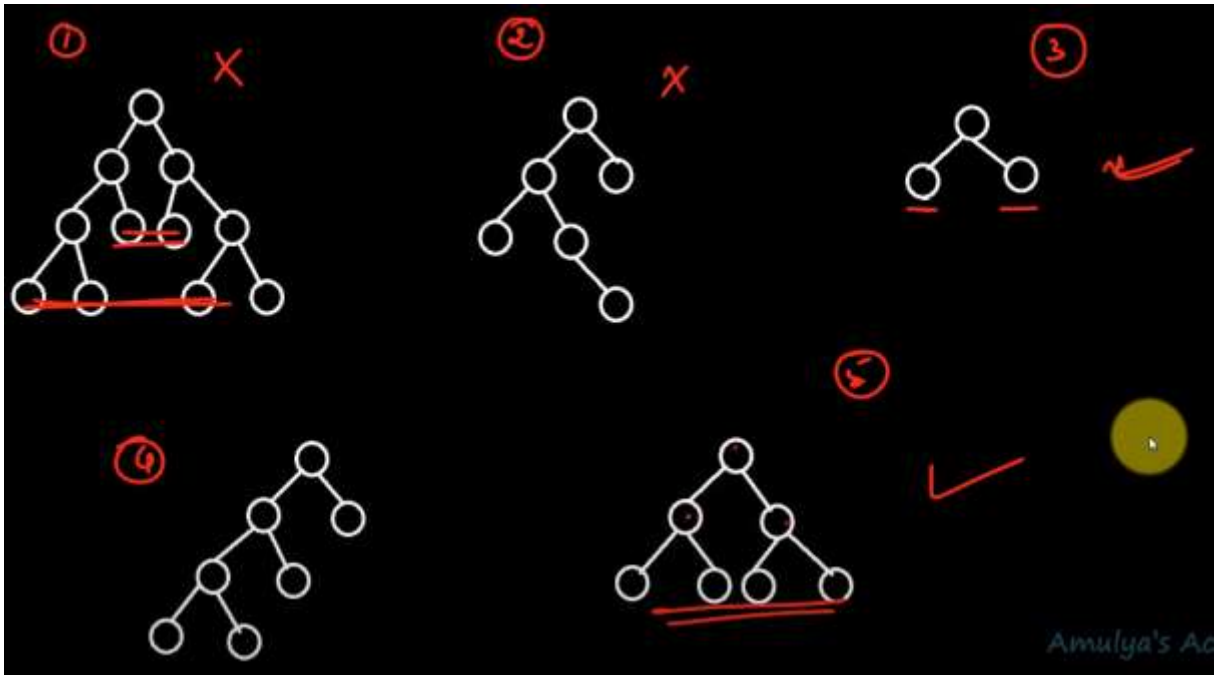
3. **SKEWED**, every parent node has only one child node.



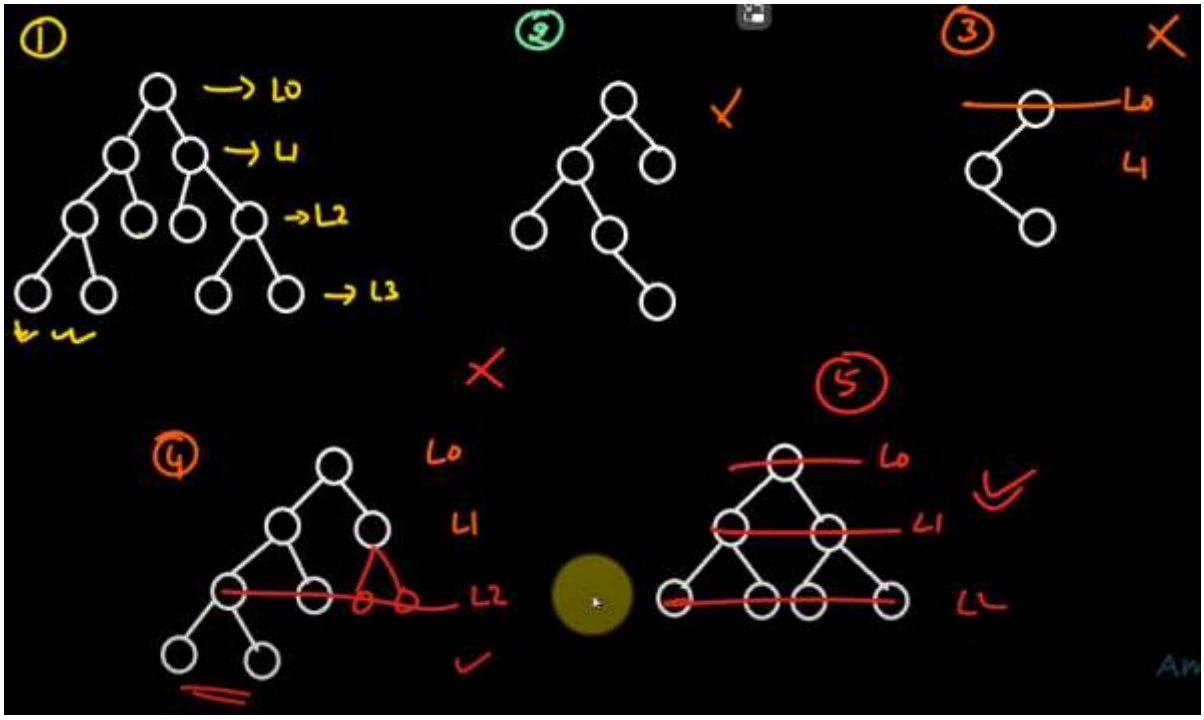
4. **BALANCED**, a Binary tree in which height*(or level) of the left and the right sub-trees of every node may differ by at most 1.



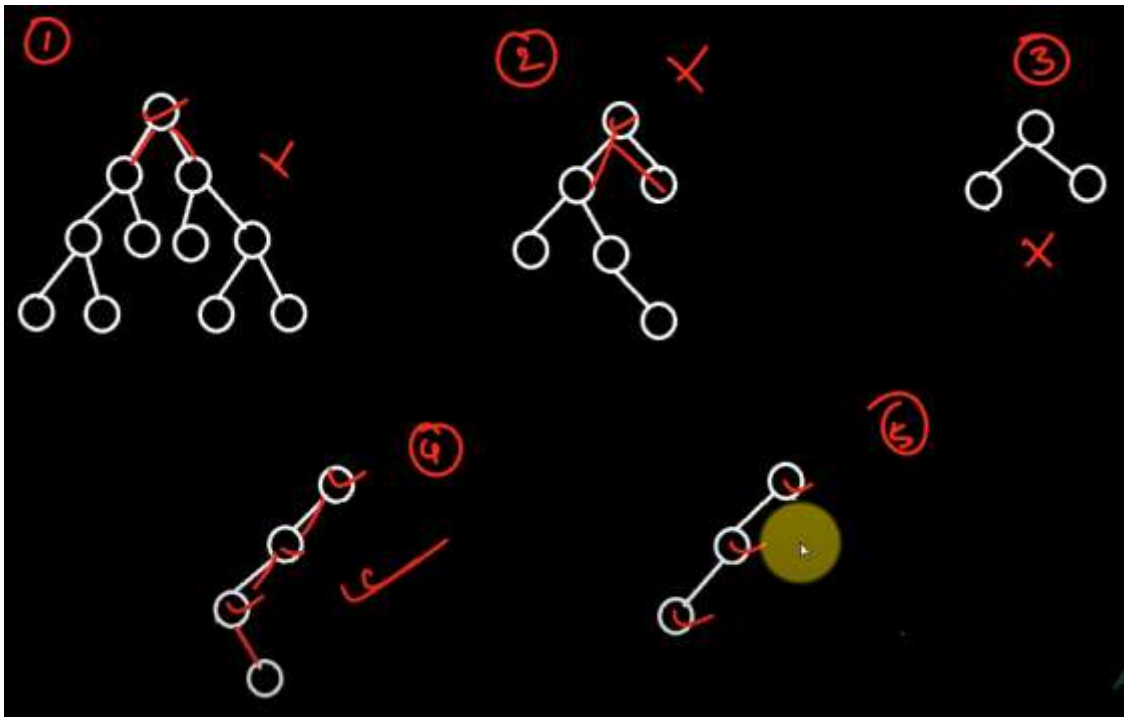
5. **EXERCISE**:
   - PERFECT:



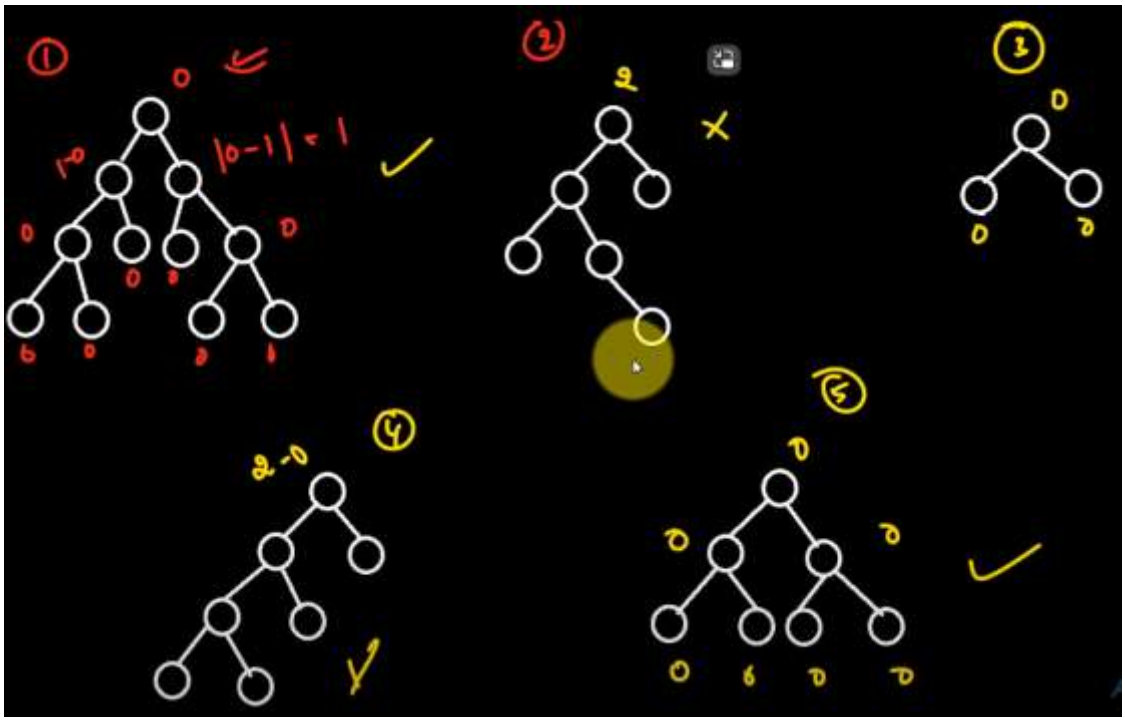
   - COMPLETE:



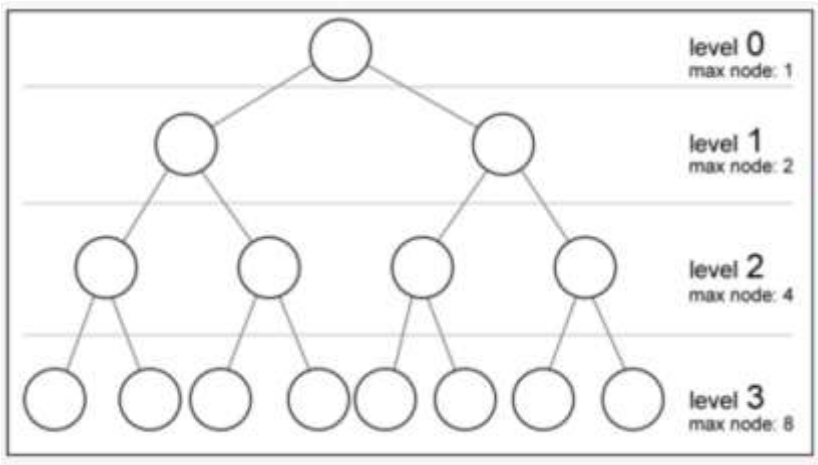
- SKEWED:



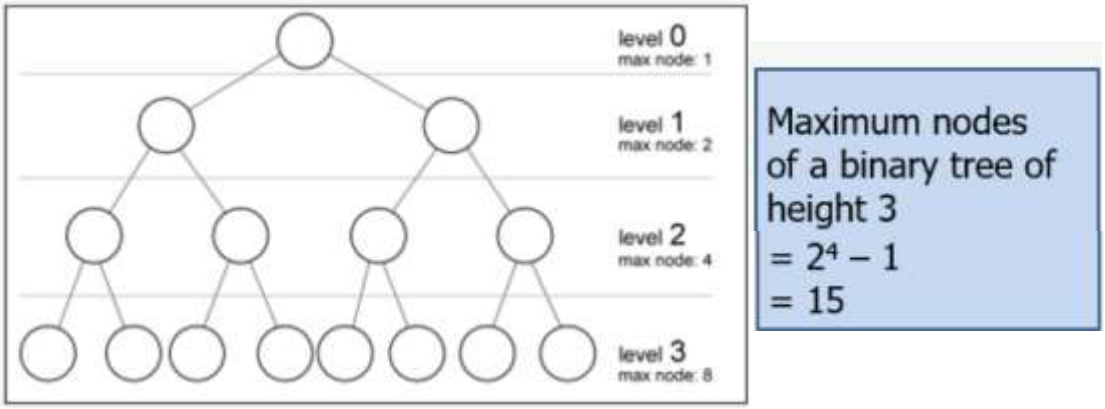
- BALANCED:



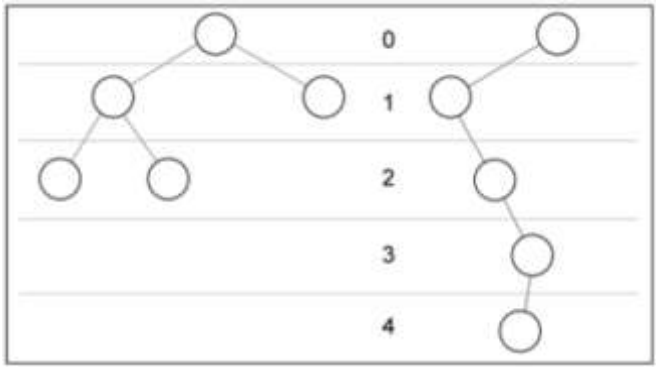
- Property of Binary Tree:
  1. To see the maximum nodes that can be handled by a binary tree at **each level** is 2^k, while *k* = level.



  2. To see the maximum nodes that can be handled by a binary tree is $2^{k+1} - 1$, while *k* means “level”.

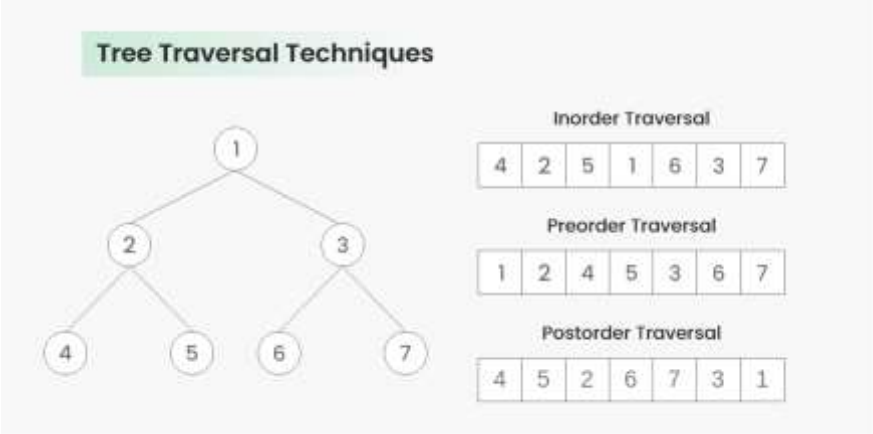


  3. To see the **minimum** level of a binary tree when you have *n* nodes is $\log_2(n)$ dan dibulatkan ke bawah.
  4. To see the **maximum** level of a binary tree when you have *n* nodes is $n - 1$.



-> Example of 5 nodes -> $\log_2(5) = 2,3$

- Tree Traversal Techniques: **Inorder**, **Preorder**, and **Postorder**.



1. Inorder algorithm:

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)*

2. Preorder algorithm:

Algorithm Preorder(tree)

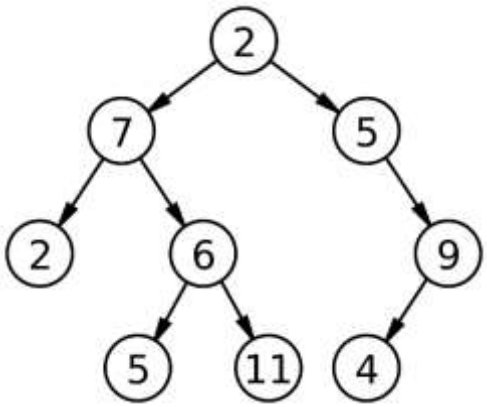
1. *Visit the root.*
2. *Traverse the left subtree, i.e., call Preorder(left->subtree)*
3. *Traverse the right subtree, i.e., call Preorder(right->subtree)*

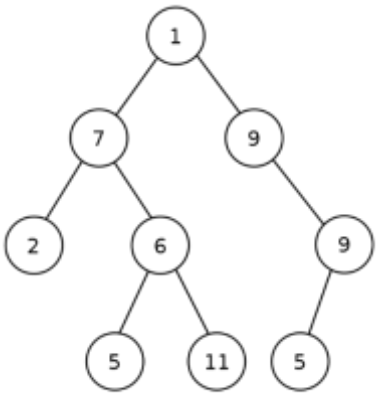
3. Postorder algorithm:

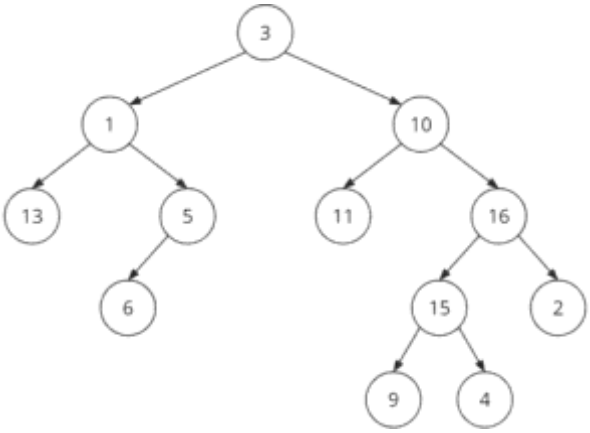
Algorithm Postorder(tree)

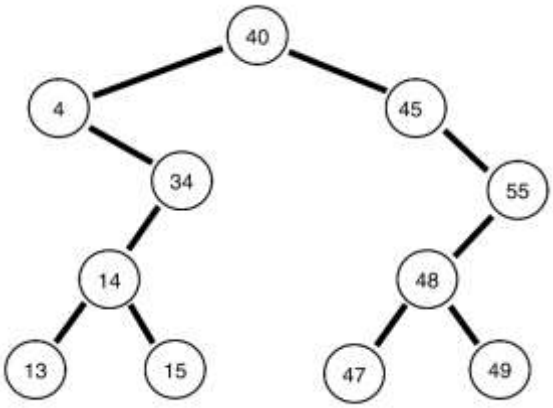
1. *Traverse the left subtree, i.e., call Postorder(left->subtree)*
2. *Traverse the right subtree, i.e., call Postorder(right->subtree)*
3. *Visit the root*

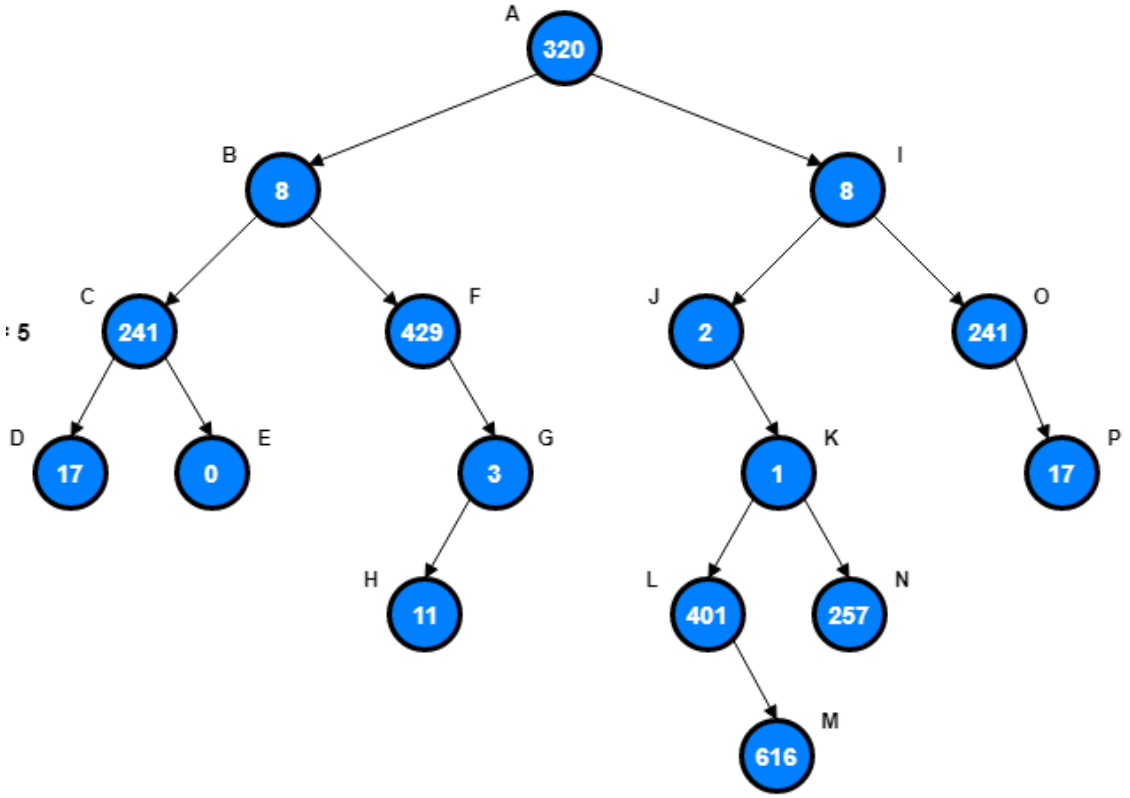
4. **EXERCISE** (Buatkan inorder, preorder, dan postorder):



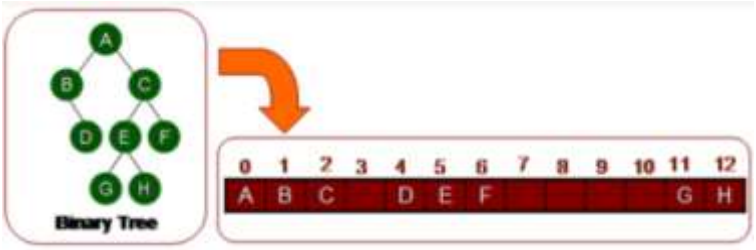








- Representation of Binary Tree
  1. Using array

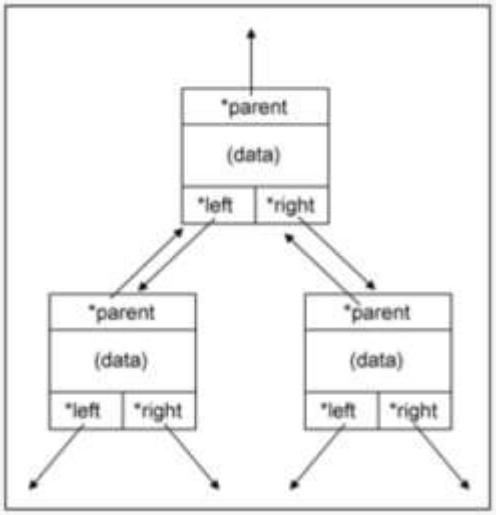


  - Index on array represents node number.
  - In this case, there are supposed to have 13 node, so we need to reserve 13 index.
  - Index 0 is Root node
  - Index Left Child: **2*p* + 1**
  - Index Right Child: **2*p* + 2**
  - *p* = parent
  - And to find the parent: **(*i*-1)/2**, lalu bulatkan ke bawah
  - *i* = corresponding index
  - Dari **EXERCISE** sebelumnya, coba buatkan representasi array nya

  2. Using linked list

  Dibawah ini merupakan contoh memakai double linked list:

```
struct node {
      int data;
      struct node *left;
      struct node *right;
      struct node *parent;
};

struct node *root = NULL;
```



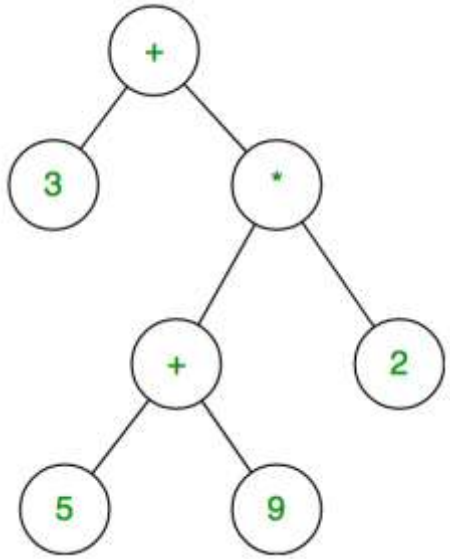
  The code below is using single linked list:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
  int item;
  struct node* left;
  struct node* right;
}node;
```

```
// Create a new Node
node* create(int value) {
  node* newNode = malloc(sizeof(node));
  newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}
```

```c
// Inorder traversal
void inorderTraversal(node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ", root->item);
  inorderTraversal(root->right);
}

// Preorder traversal
void preorderTraversal(node* root) {
  if (root == NULL) return;
  printf("%d ", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}

// Postorder traversal
void postorderTraversal(node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ", root->item);
}

int main() {
  node* node1 = create(1);
  node* node2 = create(2);
  node* node3 = create(3);
  node* node4 = create(4);
  node* node5 = create(5);
  node* node6 = create(6);
  node* node7 = create(7);

  node1->left = node2;
  node1->right = node3;
  node2->left = node4;
  node2->right = node5;
  node3->left = node6;
  node3->right = node7;

  printf("Traversal of the inserted binary tree \n");
  printf("Inorder traversal \n");
  inorderTraversal(node1);

  printf("\nPreorder traversal \n");
  preorderTraversal(node1);


  printf("\nPostorder traversal \n");
  postorderTraversal(node1);

}
```

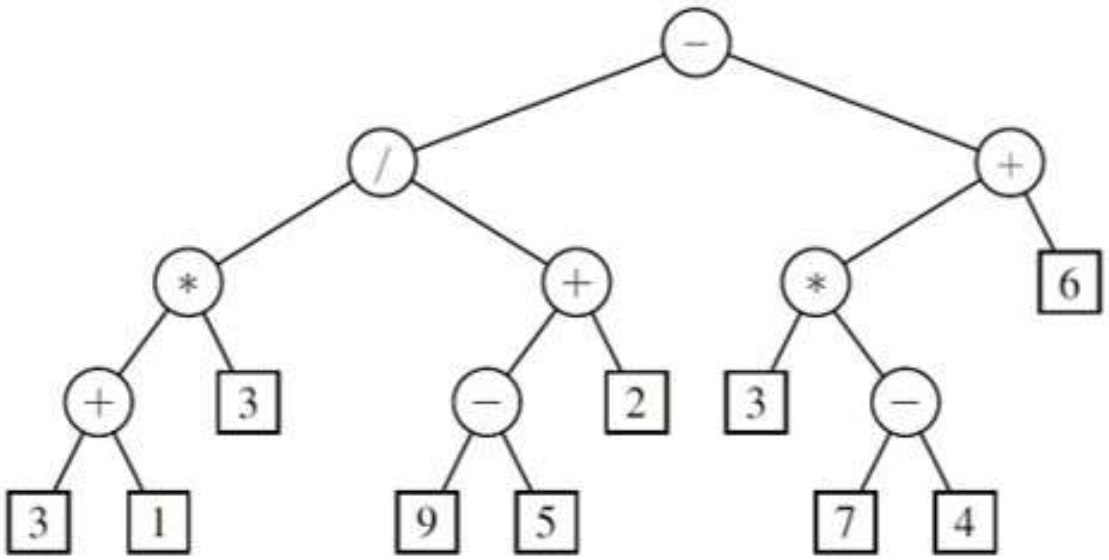
- Expression Tree Concept
  1. A binary tree in which each **internal node** corresponds to the **operator**
  2. Each **leaf node** corresponds to the **operand**
  3. **Example** in **3 + ((5+9)*2)**:



Remember the priority among the operators:
^ = highest
* / = second highest
+ - = Lowest

  4. **EXERCISE** (Buatkan bentuk expression tree nya):
     - 3+(5+9)*2
     - (4*(6-3))+(7/9)
     - 2*3/(2-1)+5*(4-1)

  5. **EXERCISE** (Buatkan inorder, preorder, dan postorder):

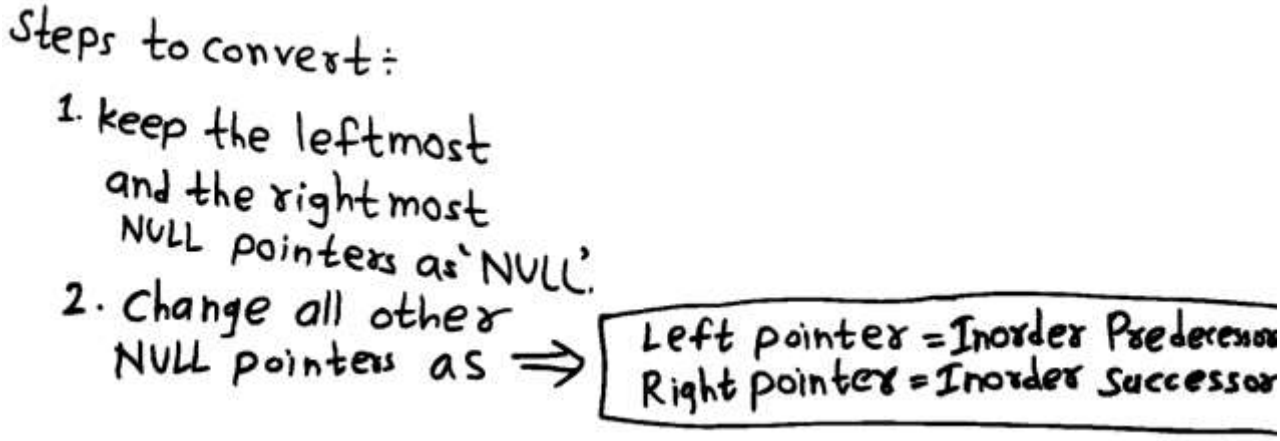


6. **Contoh Kode** (biarkan mhs kerjakan – input postfix, output infix dan prefix):

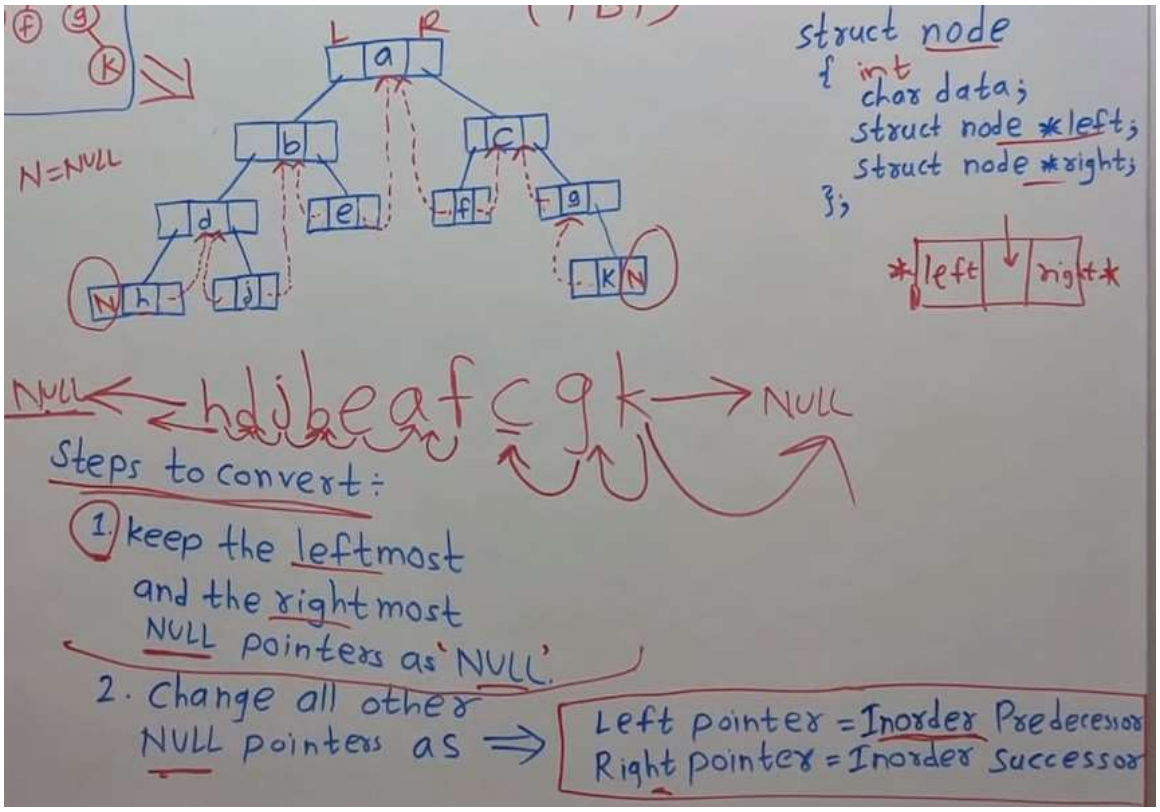
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    char item;
    struct node* left;
    struct node* right;
}node;

node* create(char value) {
    node* newNode = malloc(sizeof(node));
    newNode->item = value;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

void inorderTraversal(node* root) {
    if (root == NULL) return;

    if(is_operator(root->item)) putchar('(');
    inorderTraversal(root->left);
    printf("%c", root->item);
    inorderTraversal(root->right);
    if(is_operator(root->item)) putchar(')');
}

void preorderTraversal(node* root) {
    if (root == NULL) return;
    printf("%c", root->item);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

void postorderTraversal(node* root) {
    if (root == NULL) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%c", root->item);
}
```

```
node* construct_tree(char postfix[]) {
    struct node* stack[100];
    int top = -1;
    int i = 0;
    while (postfix[i] != '\0') {
        char ch = postfix[i];
        if (ch >= 'A' && ch <= 'Z') {
            node* newNode = create(ch);
            stack[++top] = newNode;
        } else {
            node* newNode = create(ch);
            newNode->right = stack[top--];
            newNode->left = stack[top--];
            stack[++top] = newNode;
        }
        i++;
    }
    return stack[top--];
}

int is_operator(char data){
    switch (data) {
        case '+': return 1;
        case '-': return 1;
        case '*': return 1;
        case '/': return 1;
        case '^': return 1;
        default : return 0;
    }
}


int main() {
    char postfix[] = "ABC*+D/";
    node* root = construct_tree(postfix);

    printf("Inorder traversal of expression tree:\n");
    inorderTraversal(root);

    printf("\n\nPreorder traversal of expression tree:\n");
    preorderTraversal(root);

    printf("\n\nPostorder traversal of expression tree:\n");
    postorderTraversal(root);
    return 0;
}
```

- Threaded Binary Tree Concept
  1. **Step to convert to two-way threaded**
     - Ubah ke **inorder** dulu
     - Lakukan yang ada dibawah berikut



Contoh:



  2. **Step to convert to one-way threaded**
     - Caranya sama tapi **left pointer =NULL**

(Selanjutnya lihat ppt)