Tree

Non linear data structure, composed of nodes.
Where nodes have a parent , child relationship.

What is a tree

Binary Tree

- Node, Root, Leaf, Parent, Child
- Sub tree
- Calculate number of nodes in a perfect binary tree
- Calculate min height, given number of nodes

In [ ]:

```
 
```

Binary and n-ary trees

Skew Tree

- Depth/height of skew tree
- Worst case complexity ?

Representation of a Tree

- Linked
- Array

Full binary tree: every node has either 0 or 2 children

```
Y
   1
  / \
 2   3

Y
      1
     / \
    2   3
   / \
  4   5

N
      1
     / \
    2   3
   /
  4

Y
      1
     / \
    2   3
       / \
      4   5

N
      1
     / \
    2   3
         \
          4
```

Complete binary tree: every level, except possibly the last is completely filled. Last level nodes are filled from the left to right. Can be represented using arrays. Ex: binary heap.

```
Y
  1
 / \
2   3

Y
      1
     / \
    2   3
   / \
  4   5
```

Max Number of nodes in a tree

**Linked representation of tree*

```
C++
struct Node {
    int data;
    Node * left;
    Node * right;
};


JAVA
class Node {
    public int data;
    public Node left;
    public Node right;

    public Node(int data, Node left=null, Node right=null) {
        this.data= data;
        this.left = left;
        this.righ = right;
    }
}


PYTHON

class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```
```

In [ ]:

````
```C++

struct Node {
    int data;
    Node *left;
    Node *right;
};

Node* newNode(int data, Node* left=NULL, Node* right=NULL) {
    Node *temp = new Node;

    temp->data = data;
    temp->left = left;
    temp->right = right;

    return temp;
}

// rLR - recursive
void printTree(Node *root) {
    if (root == NULL) return;

    cout << root->data << " ";
    printTree(root->left);
    printTree(root->right);
}

int main() {

    Node *n1 = newNode(20);
//    cout << n1->data << " l=" << n1->left << " r=" << n1->right;

    Node *n2 = newNode(30);
    Node *n3 = newNode(10, n1, n2);

         10
        /  \
       20  30


    printTree(n3);
}

```
````

````
**Pre-ORder traversal of tree**
```C++

struct Node {
````
````

```
    int data;
    Node *left;
    Node *right;
};

Node* newNode(int data, Node* left=NULL, Node* right=NULL) {
    Node *temp = new Node;

    temp->data = data;
    temp->left = left;
    temp->right = right;

    return temp;
}

// rLR - recursive
void preOrder(Node *root) {
    if (root == NULL) return;

    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}

// LrR - recursive
void inOrder(Node *root) {
    if (root == NULL) return;

    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}

// LRr - recursive
void postOrder(Node *root) {
    if (root == NULL) return;

    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << " ";
}

int main() {
// LRr
//     10
//    /  \
//   20  30
//  / \
// 40  50
//        \
//        60

    // inOrder
    // postOrder
    Node *root = newNode(10, newNode(20, newNode(40), newNode(50, NULL, newNode(60)))   , newNode(30) );
    preOrder(root);
```

```
}
/*
                                                   preOrder(10)
                                               /                  \
                           preOrder(20)
     preOrder(30)
                         /                \
        /       \
              preOrder(40)                  preOrder(50)
preOrder(NULL)   preOrder(NULL)
               /       \                     /       \
     preOrder(NULL)  preOrder(NULL)   preOrder(NULL)   preOrder(60)
                                                           /       \
                                                  preOrder(NULL)
preOrder(NULL)

   pre - 10 20 40 50 60 30
   in  - 40 20 50 60 10 30
   post- 40 60 50 20 30 10
*/
```
```

In [ ]:

```

```

**Types of traversal of a tree**

Depth order:

- rLR : pre order
- LrR : in order
- LRr : post order

TC: O(N) SC : O(depth of tree)

Level Order

- L->R (default)
- R->L

TC: O(N)
SC: ?

In [ ]:

```
DFS = Depth First Search -> Stack
BFS = Breadth First Search -> Queue
```

In [ ]:

```

```

# Height vs Depth

- Height: measured bottom up: Height of leaf node=0
- Depth: measured top to bottom: Depth of root node=0
```

Level=Depth (can start from 0/1)

Finding max depth of binary tree using recursion
https://leetcode.com/problems/maximum-depth-of-binary-tree/ (https://leetcode.com/problems/maximum-depth-of-binary-tree/)

DIY
https://leetcode.com/problems/minimum-depth-of-binary-tree/ (https://leetcode.com/problems/minimum-depth-of-binary-tree/)

In [ ]:
```
1
```

In [ ]:
```
1
```

Question
https://leetcode.com/problems/symmetric-tree/ (https://leetcode.com/problems/symmetric-tree/)

DIY
https://leetcode.com/problems/same-tree/ (https://leetcode.com/problems/same-tree/)

In [ ]:
```
1
```

Operations in a Tree

- Add
- Remove
- Traverse
- Search

In [ ]:
```
1
```

Traversal of a tree

Depth First Traversal, DFS = Stack

- rLR
- LrR
- LRr

Recursive implementation of DFS

In [ ]:
```
1
```

Breadth First / Level Order Traversal, BFS = Queue

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```