Class 57

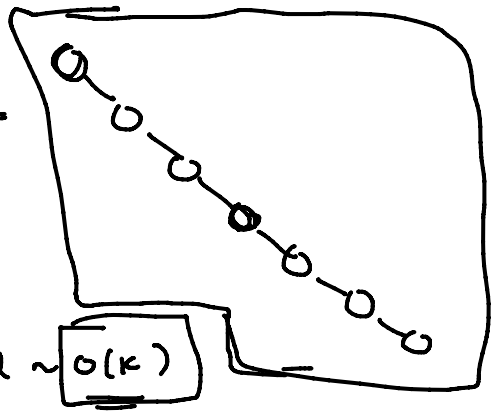
Building a Tree

Height a Tree

[Print level of a Tree (k^{th} level)] ⇒ ~O(k)

O(k*n) k = ?

(5) Skew Tree =



H = no of nodes = n

n = no of levels = H

6 5

(0.1) H = n

T.C to print k^{th} level ~ O(k)

1st level → O(1)

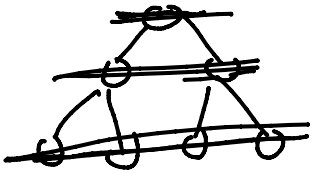
2 " → O(2)

⋮

n " → O(n)

→ O(1) + O(2) + O(3) - - - O(n)

⇒ O(n²)



* Print all the levels of Tree

```
int h = height(root);
For( i = 1; i <= h; i++)
{
    Print kth level (root, i, !w);
}
```

T.C = O(n²)

Trees → 2 types of Traversal

Depth First search

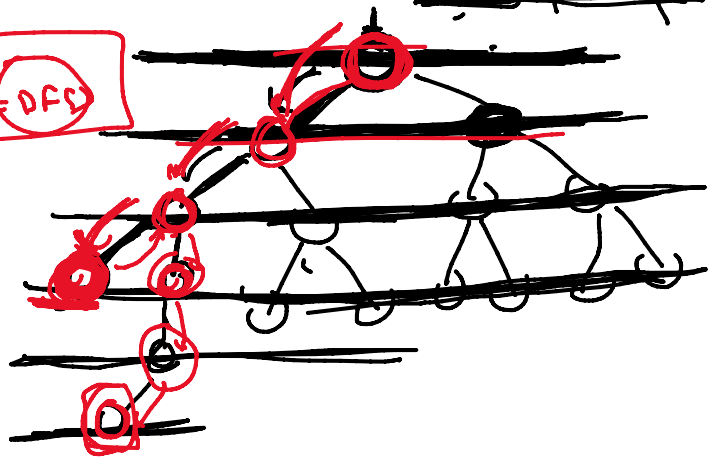
DFS

Preorder Postorder Inorder

BFS → level order Traversal

Preorder Post Inorder of

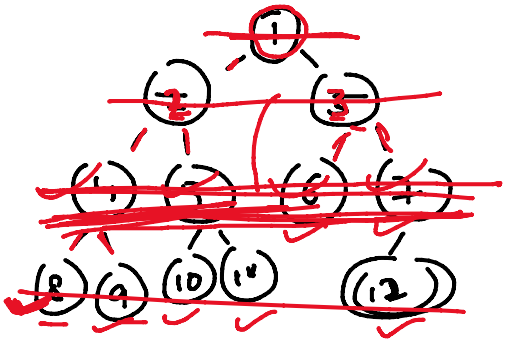
Recursion = DFS



along the depth

Breath First search

width , height



⇒

BFS output

1 2 3 4 5 6 7 8 9 10 11 12

Total time = O(2n) = O(n)

1 2 3 4 5 6 7 8 9 10 11 12

Queue Data Structure

```
queue <Node*> q;
q.push(root);
while(!q.empty())
{  node* f = q.front();
   cout << f->data << " ";
   q.pop();
   if(f->left)
   {  q.push(f->left);
   }
   if(f->right)
      q.push(f->right);
   }
```

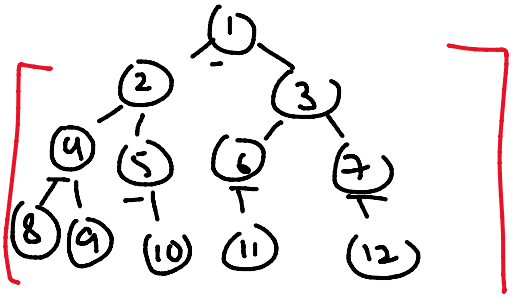
1

1 2 3

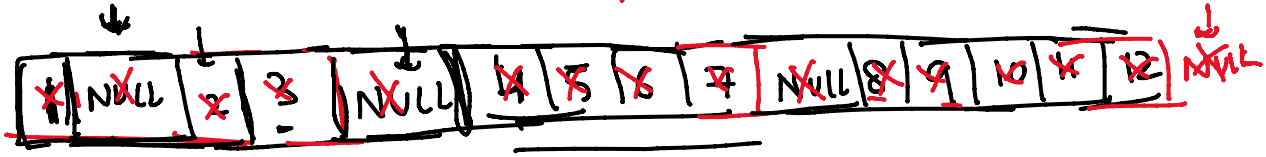
4 5 6

O(n)

q.push(t->right);



3
4 5
6 1



q.push(root);
q.push(NULL);

→ 1
→ 2 3
→ 4 5 6 7
→ 8 9 10 11 12

while(!q.empty())
{
  Node* f = q.front();
  if(f == NULL)
  {
    cout << endl; k--;
    q.pop();
    if(!q.empty())
      q.push(NULL);
  }
  else if(k == 0)
  {
    cout << f->data;
    q.pop();
    if(f->left)
      q.push(f->left)
    if(f->right)
      q.push(f->right)
  }
}

(k) = (5)

(4) (3) (2) (1) (i == 0)

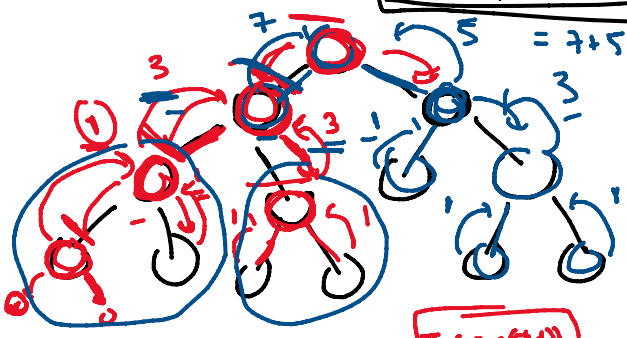
kth level

DFS

BFS

Binary Trees

count → No of Nodes in a Tree



= 7+5+1 = 13

T.C = O(H+1)

Skew Tree

O(H+1) = n

DFS

BFS

O(n)

$2^0 + 2^1 + 2^2 + 2^3$

log n

log n

n levels

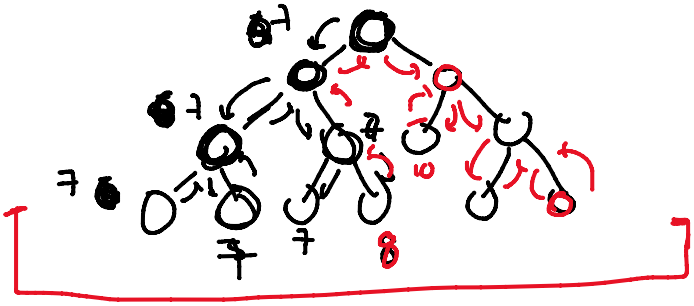
(n-1) levels

$2^0 + 2^1 + \cdots 2^{n-1}$

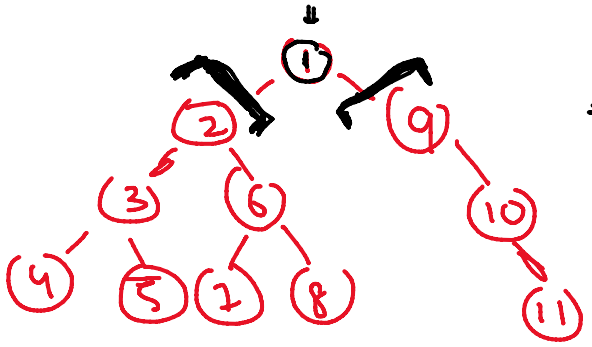
```
Void Preorder (Node* root, int & count)
{   if (root == NULL) return;
    count++;
    Preorder (root->left, count);
    Preorder (root->right, count);
}
Preorder (root, 0);
```

```
int countNodes (Node* root)  ⇒ No of Nodes in a tree start from root
{   if (root == NULL)
        return 0;
    int left = countNodes (root->left);
    int right = countNodes (root->right);
    return 1 + left + right;
}
```

count < count = 13



Count Sum of all Nodes



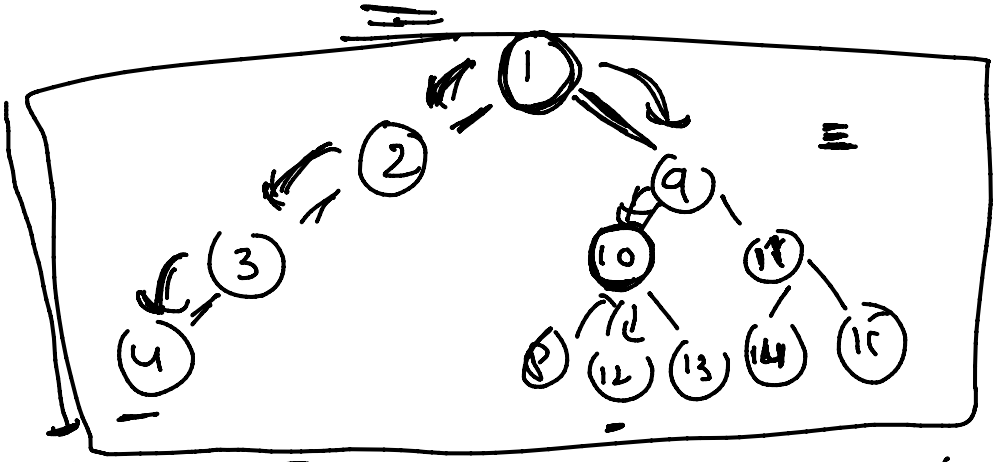
=>

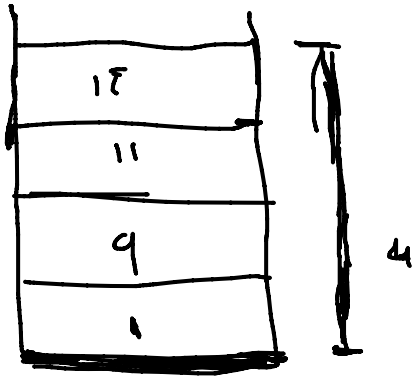
w/o reference

```
int sumNodes(Node* root)
{   if(root == NULL)
        return 0;

    int left = sumNodes(root->left);
    int right = sumNodes(root->right);

    return left + right + root->data;
}
```

=>

T.C = O(n) or O(H)





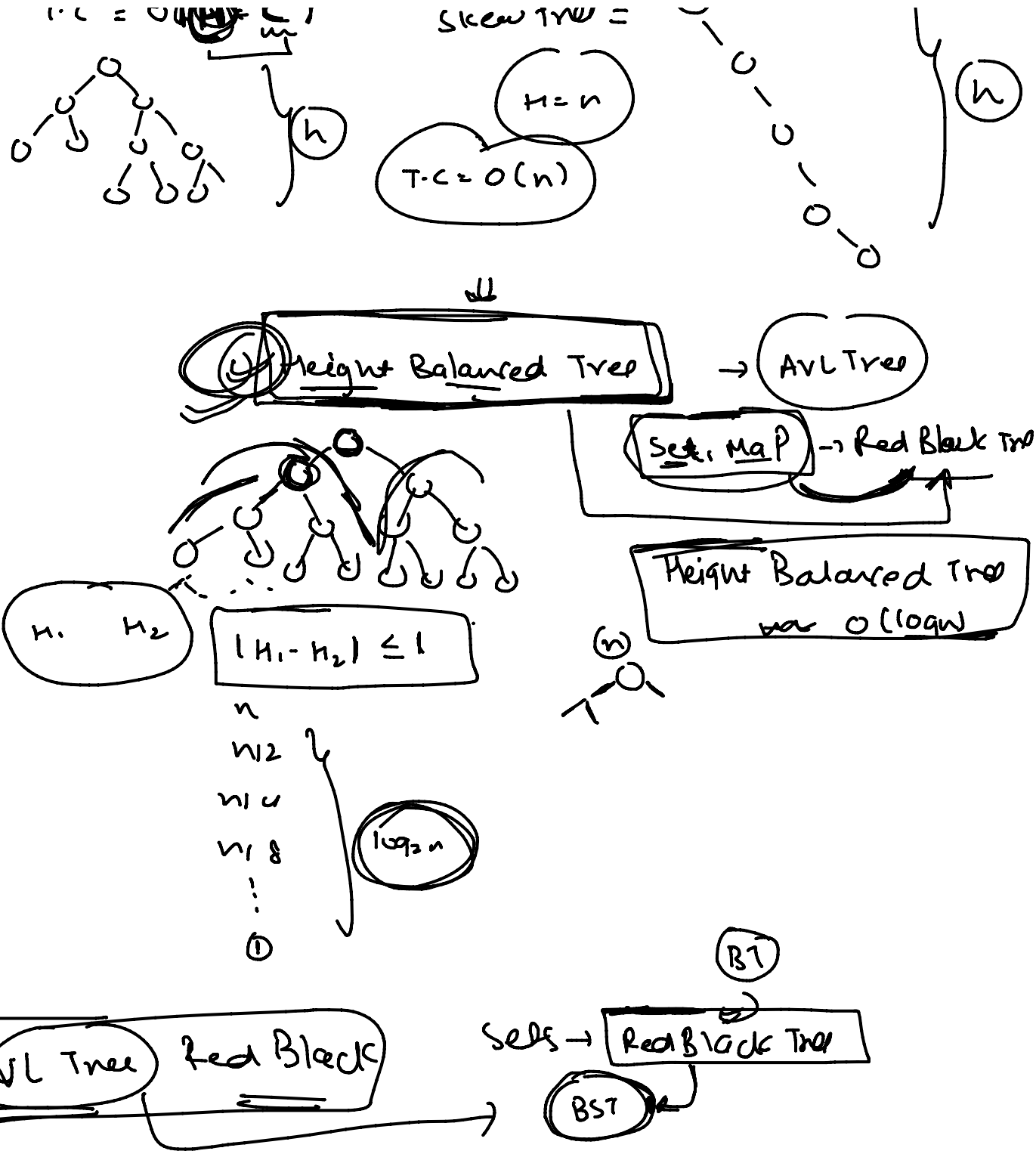
2 baar visit

T.C = O(H)

T.C = O(N) ⇒ O(K·H) ⇒ O(H·H) ⇒ O(H)

T.C = O(H + C)

Skew tree =


Skew Tree =
h
H = n
T.C = O(n)
Height Balanced Tree
AVL Tree
Set, Map
Red Black Tree
Height Balanced Tree
T.C = O(log n)
H1 H2
|H1 - H2| ≤ 1
n
n/2
n/4
n/8
1
log2 n
AVL Tree
Red Black
BT
Sets
Red Black Tree
BST