Rooted Binary Trees

Subhabrata Samajder

IIIT, Delhi

Summer Semester,

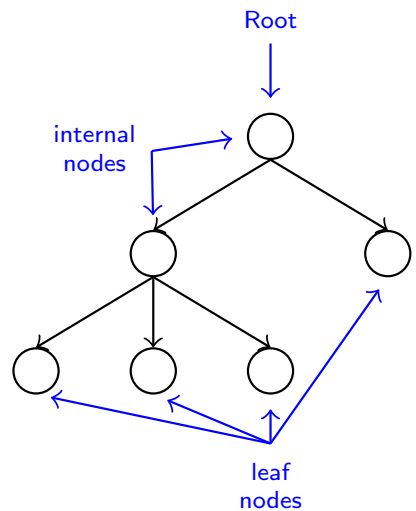
2nd June, 2022

Rooted Binary Trees

A General Tree

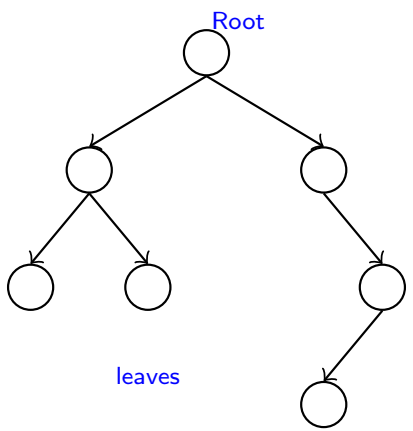
A (rooted) tree is an abstract data type

- one entry point, the *root*.
- Each node is either a *leaf* or an *internal node*.
- An internal node has 1 or more *children*.
- The internal node is *parent* of its child nodes.
- The *leaf nodes* have no children.



Properties of Trees

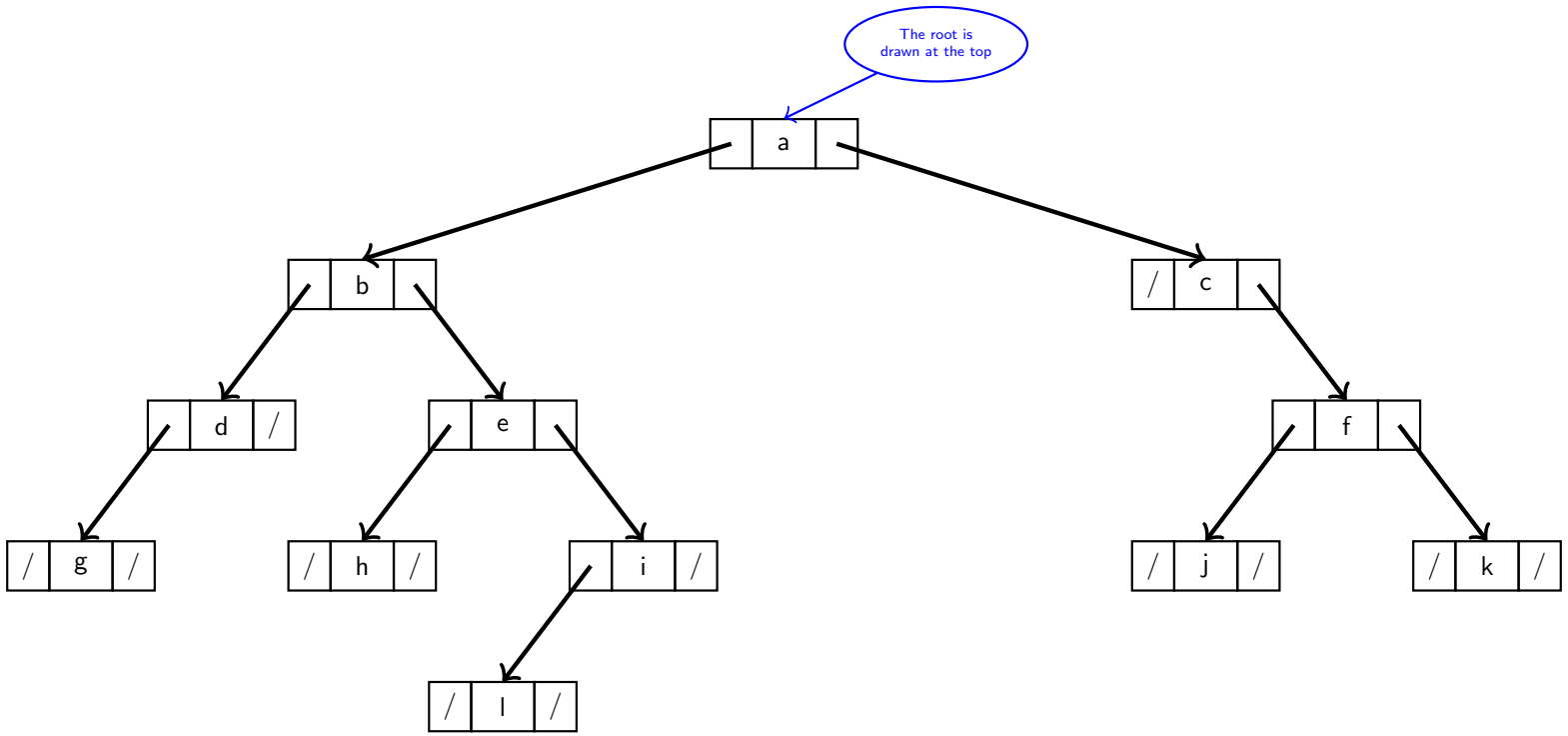
- Only access point is the root.
- All nodes, except the root, have one parent.



Binary Trees

- Array, linked lists, stack, or queue are all linear structures.
- A (rooted) tree has a hierarchical structure (non-linear).
- The (rooted) binary tree is a special case of the general tree, having maximum of two child nodes.
- It is either empty or consists of
  - an element called the root,
  - and two distinct binary trees, called the left subtree and right subtree.

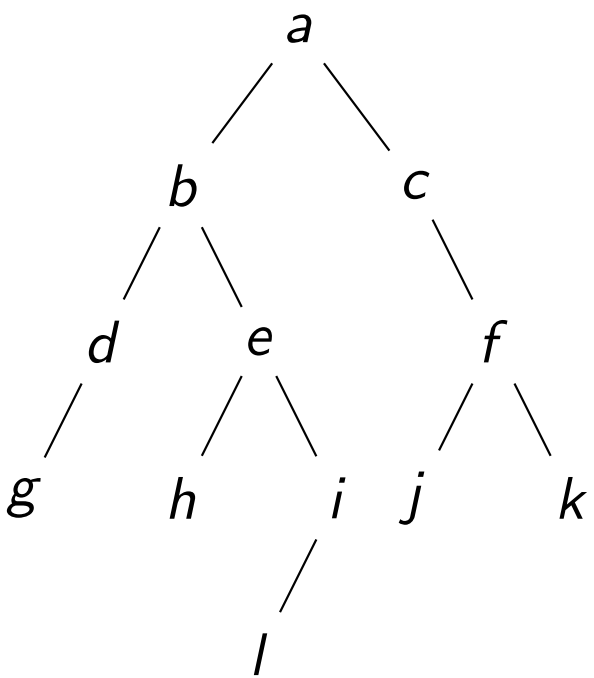
Picture of a Binary Tree



Binary Tree

- Each node consists of
  - Data value.
  - Left link: Points to the left child
  - Right link: Points to the right child
- Any node can have null value in its right link or in its left link.
- Leaf nodes have null values in left and right link.
- Children of a node are termed siblings

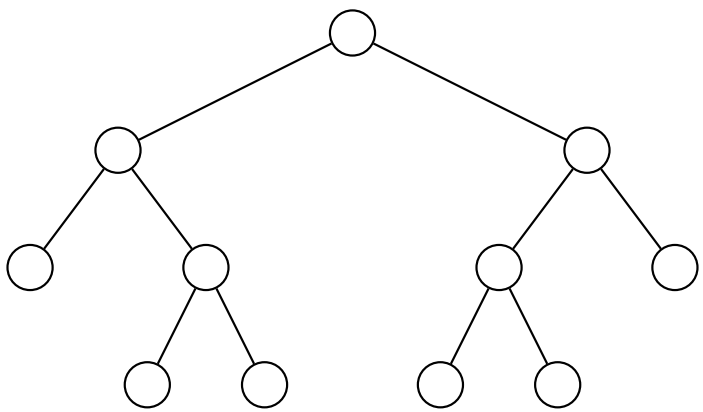
Size and depth



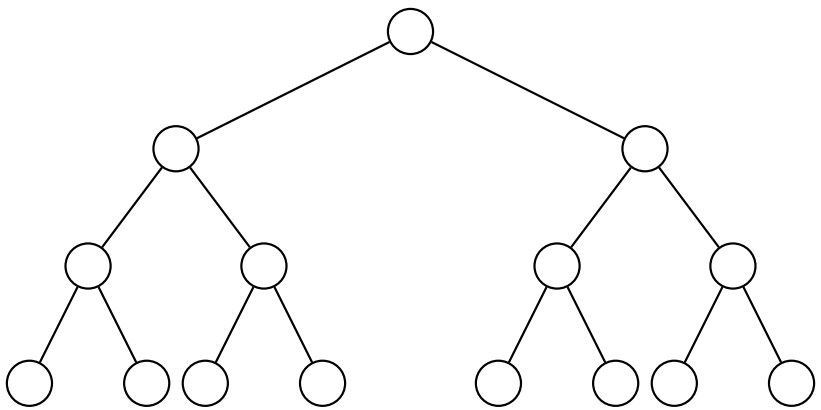
- The size of a binary tree is the number of nodes in it.
  - This tree has size 12.
- The depth of a node is its distance from the root.
  - a is at depth zero.
  - e is at depth 2.
- The depth of a binary tree is the depth of its deepest node.
  - This tree has depth 4.

Full Binary Tree

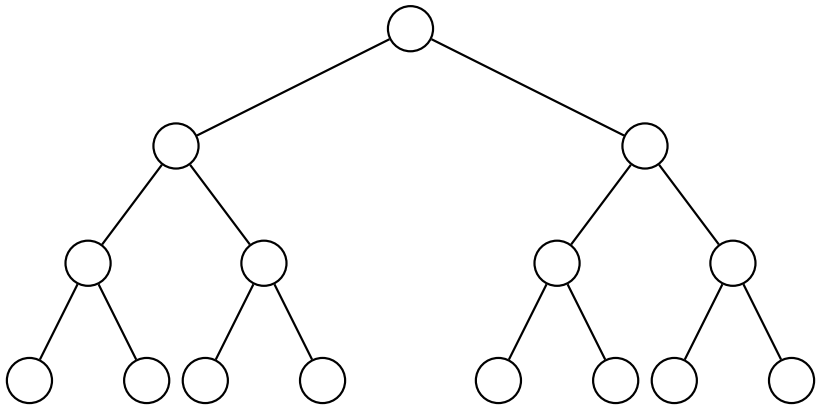
- Every node has zero or two children.



Height is $\mathcal{O}(\log n)$



Height is $\mathcal{O}(\log n)$



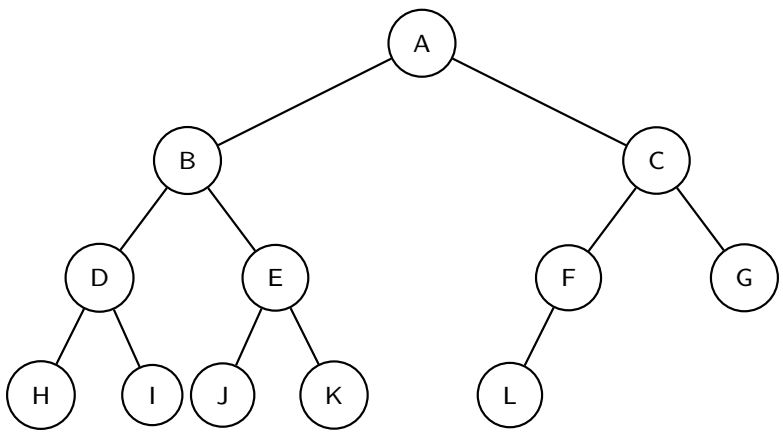
$$
\begin{aligned}
N &= 1 + 2 + 4 + 8 + 16 + \cdots \\
N &= 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{h-1} \\
N &= \frac{2^h - 1}{2 - 1} \\
N + 1 &= 2^h
\end{aligned}
$$

Taking log of both sides

$$h = \log_2(N + 1)$$

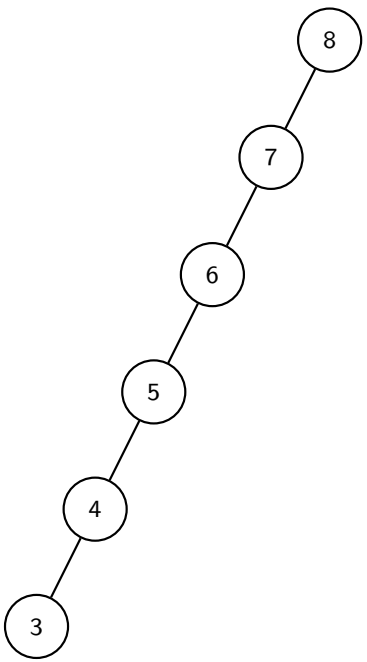
Complete Tree

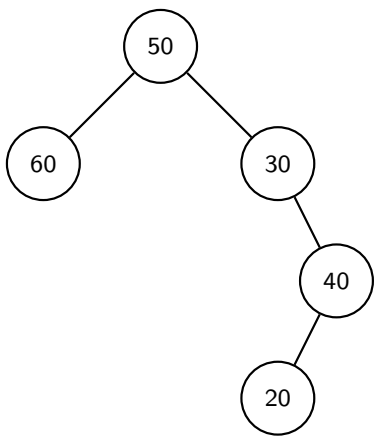
- Has all levels filled, except possibly the last level, where all nodes are as far left as possible.



Skewed Tree

- A skewed tree is one which is predominantly leaning to one side.





Binary Tree: Definition

- A **rooted binary tree** is defined recursively: it consists of
  - a root,
  - a left subtree, and
  - a right subtree
- A tree node can be constructed with or without any data.
- Array implementation of a tree is very messy if the tree is large and many of it's internal nodes are missing

Binary Tree Node in C

```
typedef struct BTNode {
   int nData;
   struct Node *pLeft;
   struct Node *pRight;
} BTNode;
```

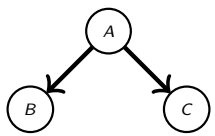
Traversing a Binary Tree

Tree Traversals

- Traverse (or walk): To visit each node in the binary tree exactly once.
- They are naturally recursive.
- Popular ways to traverse a binary tree:
  - **Pre-order traversal:** root, left, right.
  - **In-order traversal:** left, root, right.
  - **Post-order traversal:** left, right, root.

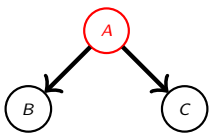
Pre-order Traversal

- root, left, right.



Pre-order Traversal

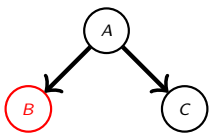
- root, left, right.



Output: A

Pre-order Traversal

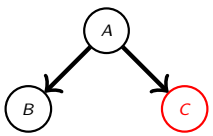
- root, left, right.



Output: *A*, *B*

Pre-order Traversal

- root, left, right.

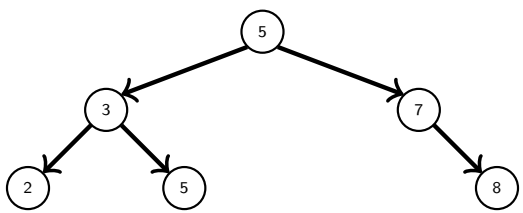


Output: A, B, C

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

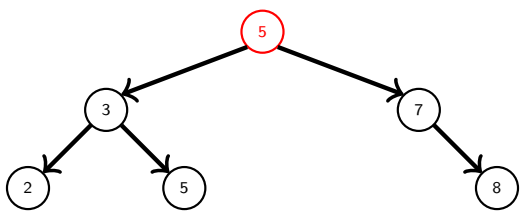


Output:

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

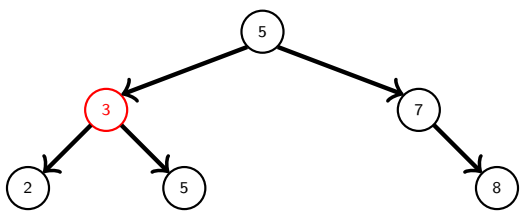


Output: 5

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

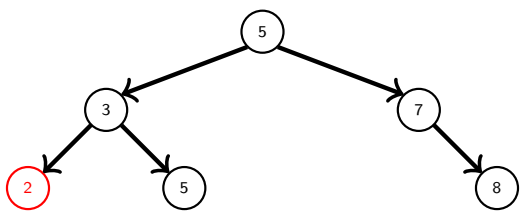


Output: 5, 3

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

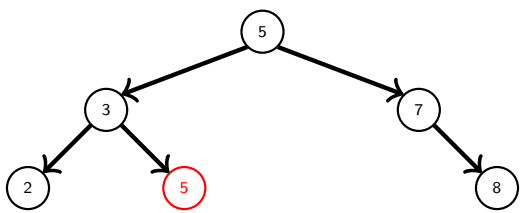


Output: 5, 3, 2

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

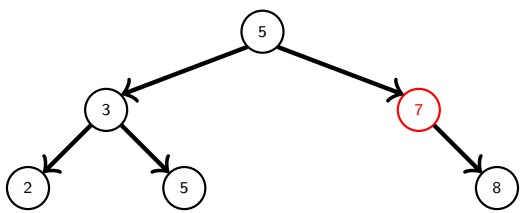


Output: 5, 3, 2, 5

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

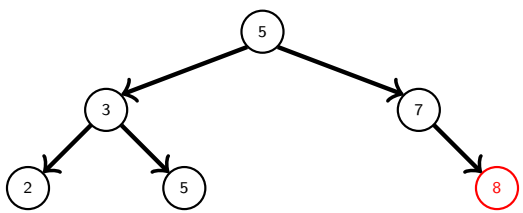


Output: 5, 3, 2, 5, 7

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

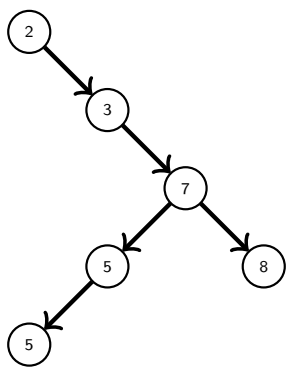


Output: 5, 3, 2, 5, 7, 8

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

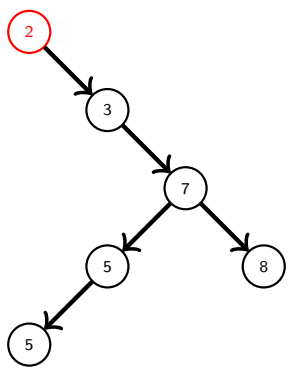


Output:

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

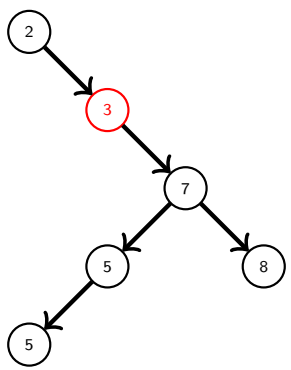


Output: 2

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

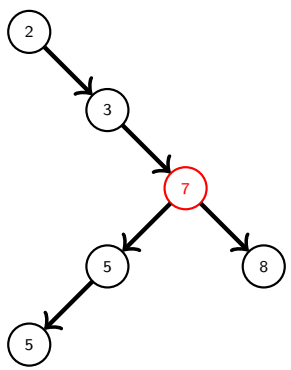


Output: 2, 3

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

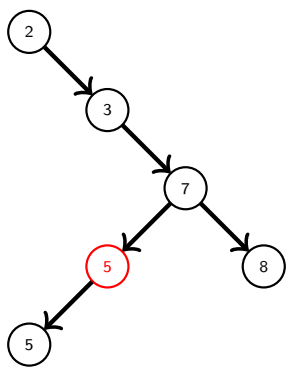


Output: 2, 3, 7

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

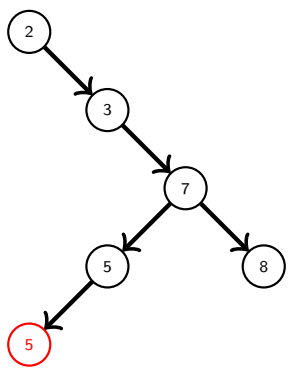


Output: 2, 3, 7, 5

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```

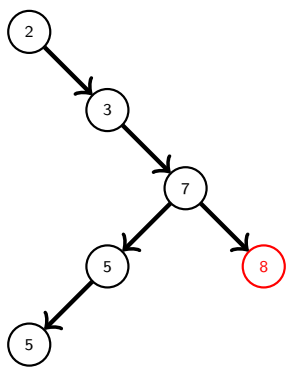


Output: 2, 3, 7, 5, 5

Pre-order Traversal

- root, left, right.
- The nodes are visited in root, left, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      printf ("%d, ", pRoot->nData);
      display (pRoot->pLeft);
      display (root->pRight);
   }
}
```



Output: 2, 3, 7, 5, 5, 8

Pre-Order-Tree-Walk($root[T]$)

I/P: The root of a binary tree T.

```
Begin
  if x ≠ nil then
    print Key[x];
    Pre-Order-Tree-Walk(left[x]);
    Pre-Order-Tree-Walk(right[T]);
  else
    return FLAG;
End
```

Complexity:

Pre-Order-Tree-Walk($root[T]$)

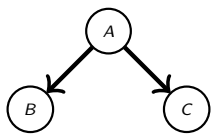
I/P: The root of a binary tree T.

```
Begin
  if x ≠ nil then
    print Key[x];
    Pre-Order-Tree-Walk(left[x]);
    Pre-Order-Tree-Walk(right[T]);
  else
    return FLAG;
End
```

Complexity: $\Theta(n)$, where $n = \#$ nodes.

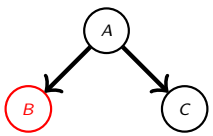
In-order Traversal

- left, root, right.



In-order Traversal

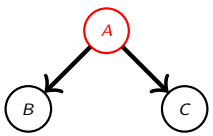
- left, root, right.



Output: B

In-order Traversal

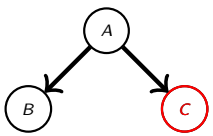
- left, root, right.



Output: B, A

In-order Traversal

- left, root, right.

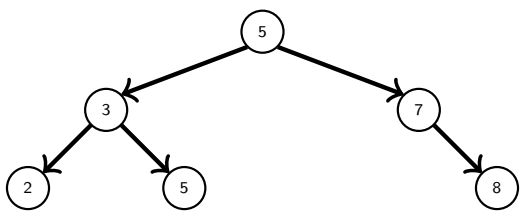


Output: B, A, C

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

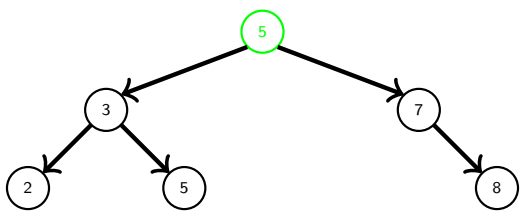


Output:

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

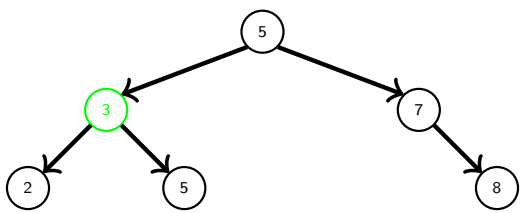


Output:

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

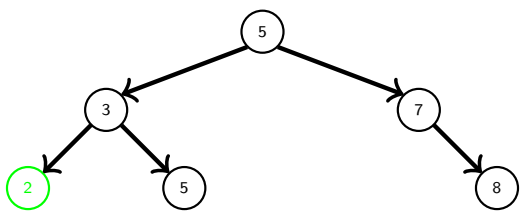


Output:

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

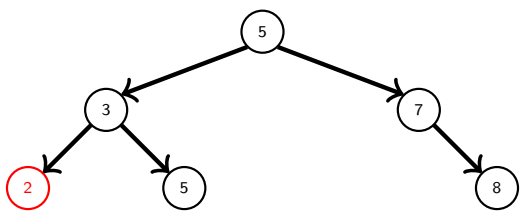


Output:

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

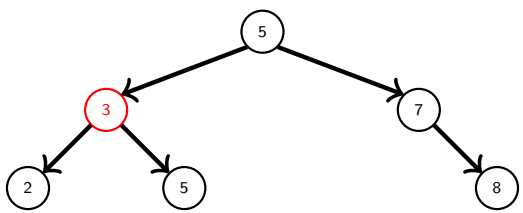


Output: 2

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

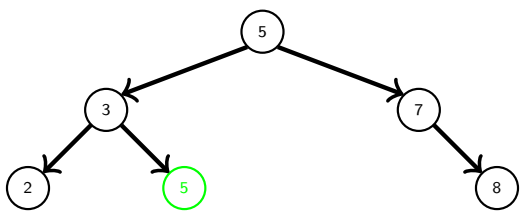


Output: 2, 3

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

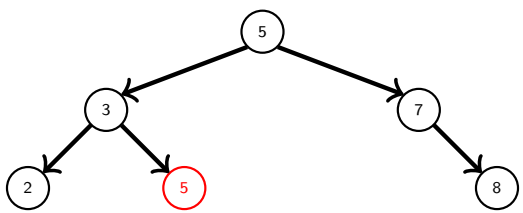


Output: 2, 3

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

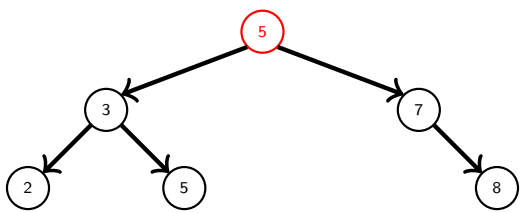


Output: 2, 3, 5

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

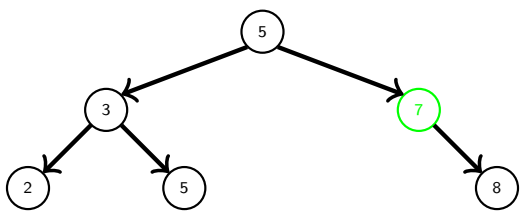


Output: 2, 3, 5, 5

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

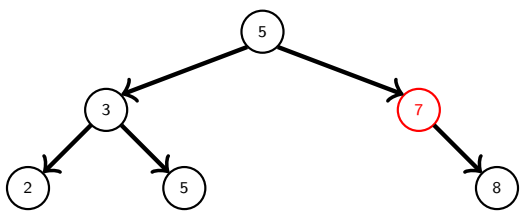


Output: 2, 3, 5, 5

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

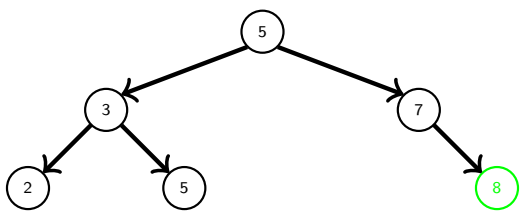


Output: 2, 3, 5, 5, 7

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

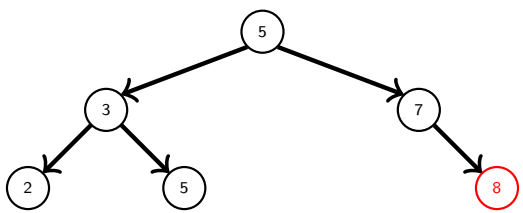


Output: 2, 3, 5, 5, 7

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

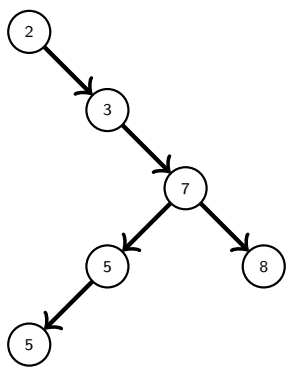


Output: 2, 3, 5, 5, 7, 8

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

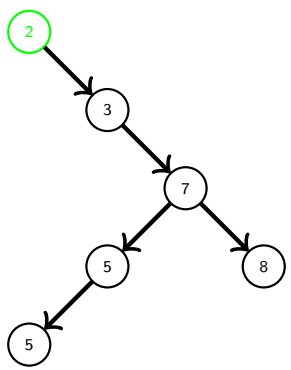


Output:

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

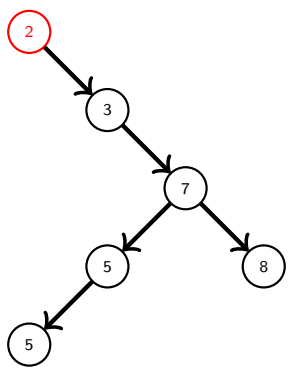


Output:

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

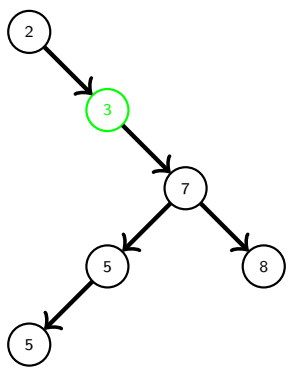


Output: 2

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

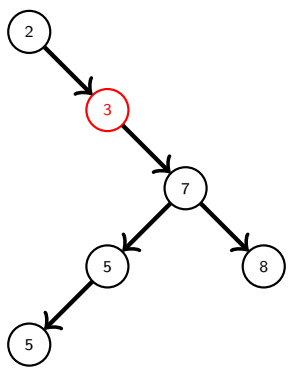


Output: 2

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

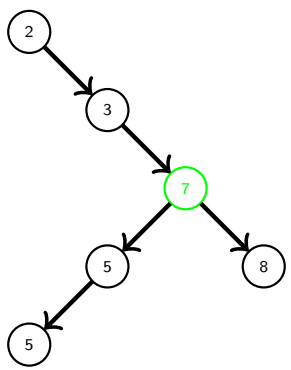


Output: 2, 3

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

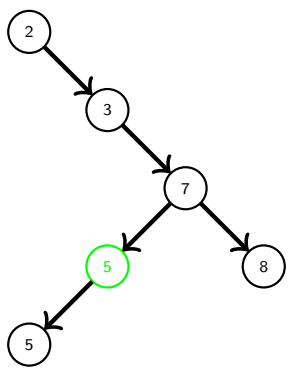


Output: 2, 3

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

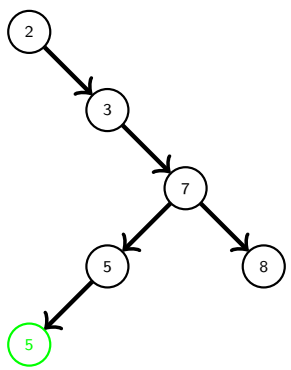


Output: 2, 3

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

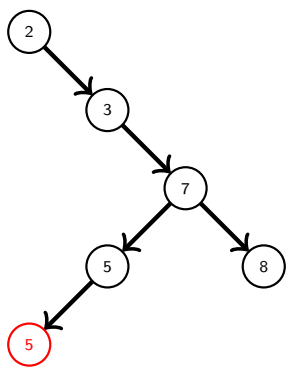


Output: 2, 3

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

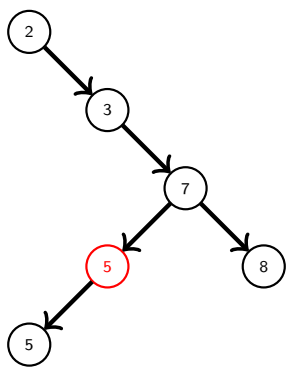


Output: 2, 3, 5

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

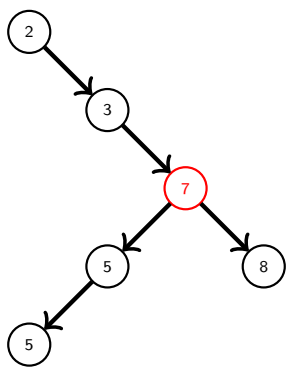


Output: 2, 3, 5, 5

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

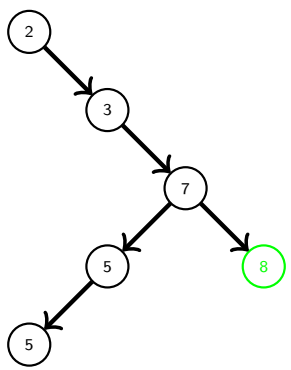


Output: 2, 3, 5, 5, 7

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```

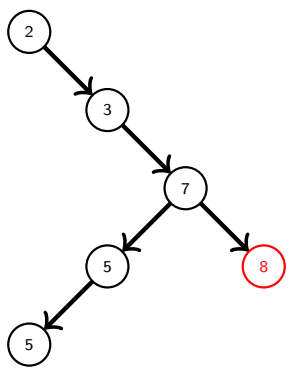


Output: 2, 3, 5, 5, 7

In-order Traversal

- left, root, right.
- The nodes are visited in left, root, right fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      printf ("%d, ", pRoot->nData);
      display (root->pRight);
   }
}
```



Output: 2, 3, 5, 5, 7, 8

In-Order-Tree-Walk($root[T]$)

I/P: The root of a binary tree T.

```
Begin
  if x ≠ nil then
    In-Order-Tree-Walk(left[x]);
    print Key[x];
    In-Order-Tree-Walk(right[T]);
  else
    return FLAG;
End
```

Complexity:

In-Order-Tree-Walk($root[T]$)

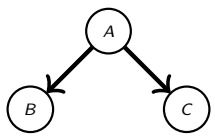
I/P: The root of a binary tree T.

```
Begin
  if x ≠ nil then
    In-Order-Tree-Walk(left[x]);
    print Key[x];
    In-Order-Tree-Walk(right[T]);
  else
    return FLAG;
End
```

Complexity: $\Theta(n)$, where $n = \#$ nodes.

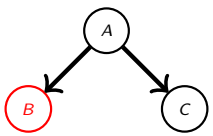
Post-order Traversal

- left, right, root.



Post-order Traversal

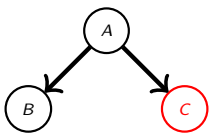
- left, right, root.



Output: B

Post-order Traversal

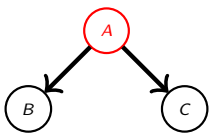
- left, right, root.



Output: B, C

Post-order Traversal

- left, right, root.

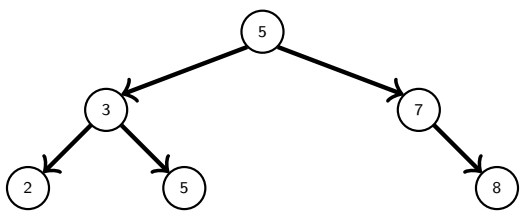


Output: B, C, A

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

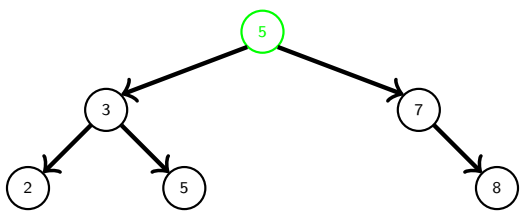


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

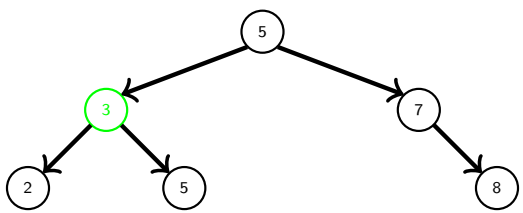


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

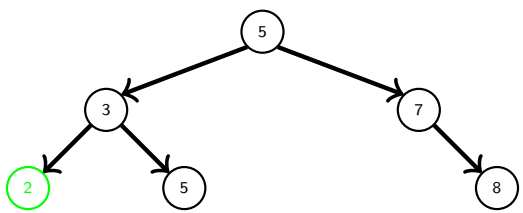


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

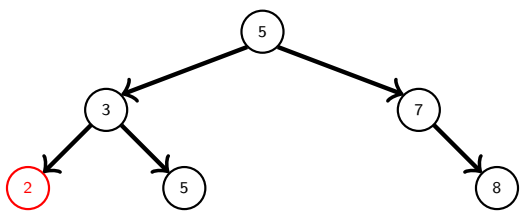


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

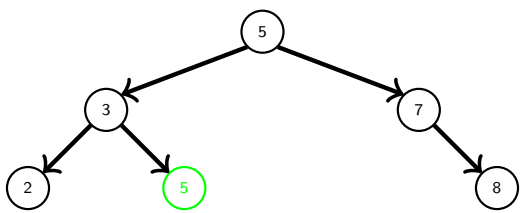


Output: 2

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

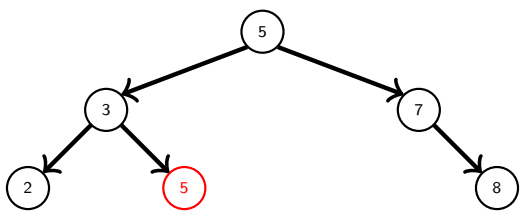


Output: 2

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

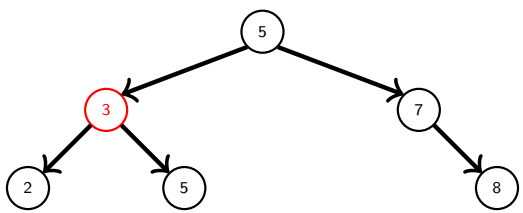


Output: 2, 5

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

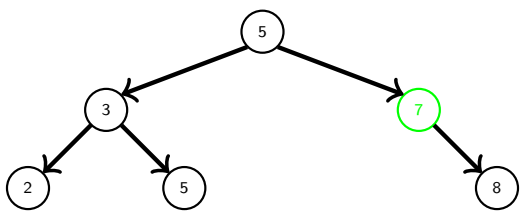


Output: 2, 5, 3

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

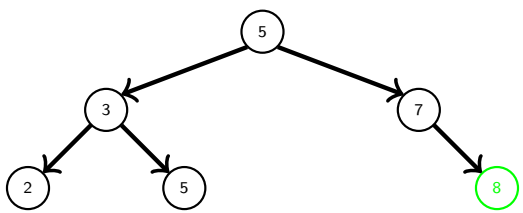


Output: 2, 5, 3

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

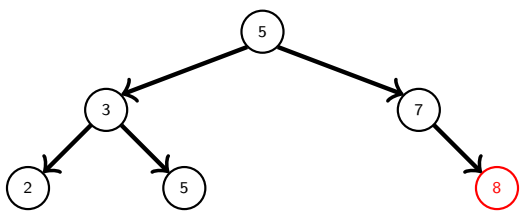


Output: 2, 5, 3

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

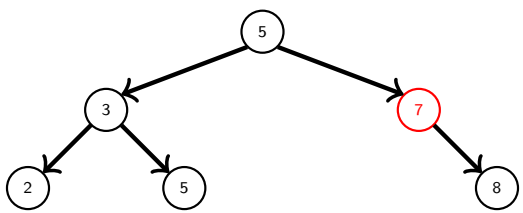


Output: 2, 5, 3, 8

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

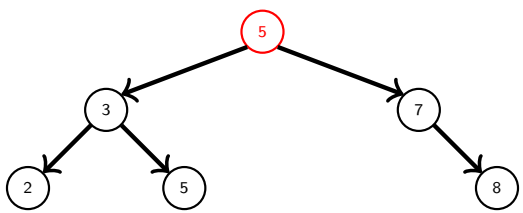


Output: 2, 5, 3, 8, 7

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

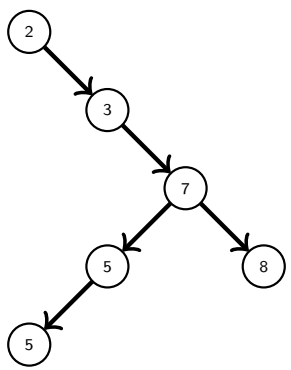


Output: 2, 5, 3, 8, 7, 5

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

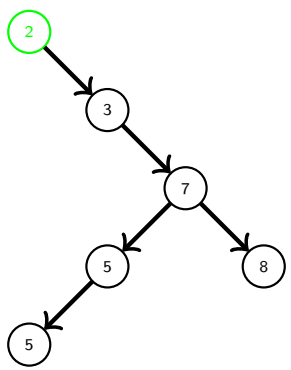


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

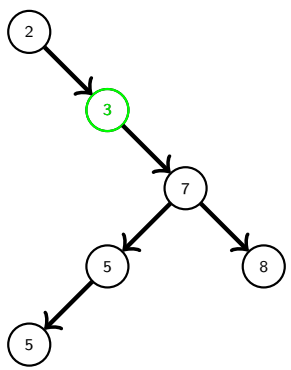


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

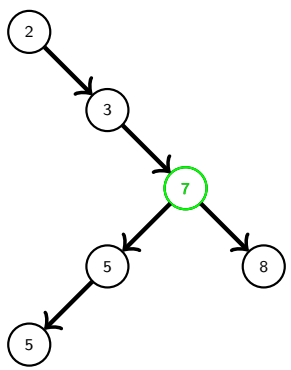


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

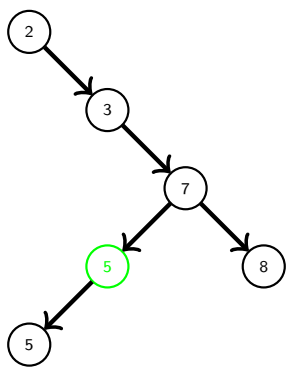


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

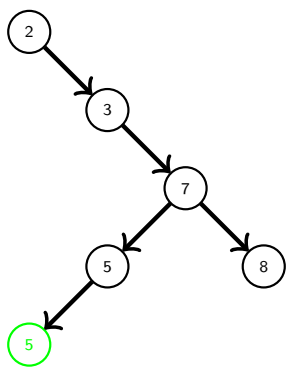


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

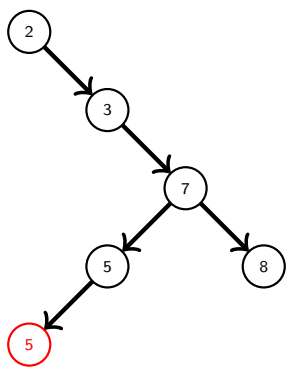


Output:

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

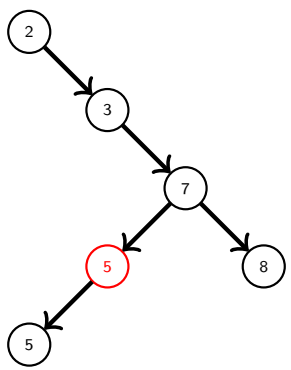


Output: 5

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

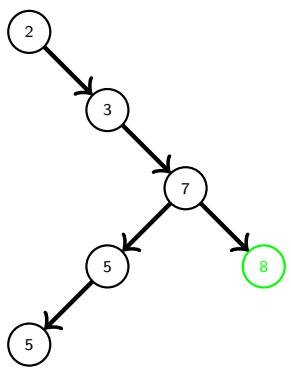


Output: 5, 5

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

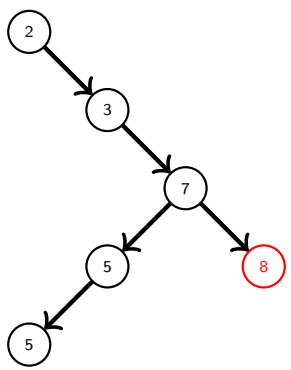


Output: 5, 5

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

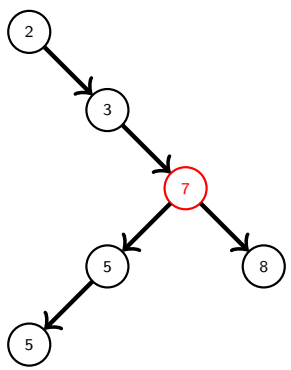


Output: 5, 5, 8

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

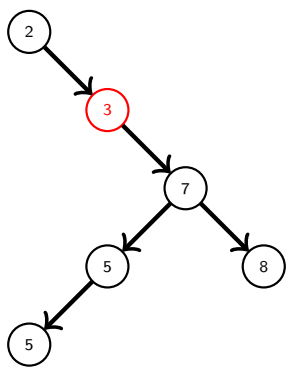


Output: 5, 5, 8, 7

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```

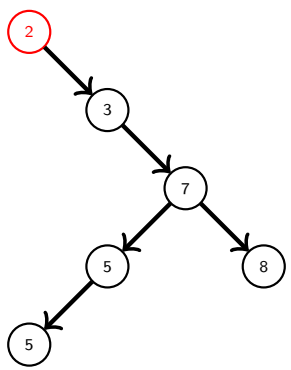


Output: 5, 5, 8, 7, 3

Post-order Traversal

- left, right, root.
- The nodes are visited in left, right, root fashion.

```
/* preorder display */
void display (Node *pRoot) {
   if (pRoot!=null) {
      display (pRoot->pLeft);
      display (root->pRight);
      printf ("%d, ", pRoot->nData);
   }
}
```



Output: 5, 5, 8, 7, 3, 2

Post-Order-Tree-Walk($root[T]$)

I/P: The root of a binary tree T.

Begin
  if $x \neq$ **nil** then
    In-Order-Tree-Walk($left[x]$);
    In-Order-Tree-Walk($right[T]$);
    print $Key[x]$;
  else
    return FLAG;
End

Complexity:

Post-Order-Tree-Walk($root[T]$)

I/P: The root of a binary tree T.

```
Begin
  if x ≠ nil then
    In-Order-Tree-Walk(left[x]);
    In-Order-Tree-Walk(right[T]);
    print Key[x];
  else
    return FLAG;
End
```

Complexity: $\Theta(n)$, where $n = \#$ nodes.

Tree Traversals

- Different trees may have same in-order traversal.
- Similarly, there can be many trees whose pre-order traversals and post-order traversals are same.
- A tree cannot be reconstruct from just one traversal sequence.
- But, given two traversals one can reconstruct the tree uniquely.

Tree Algorithms

Counting Nodes of a Binary Tree

Counting Nodes of a Binary Tree

- One needs to visit all the nodes.
- Count the current node, and
- recursively visit the left sub-tree and the right sub-tree.

```
int count (Node *Root) {
   if (pRoot==null)
      return 0;
   else
      return 1 + count(pRoot->pLeft) + count(pRoot->pRight);
}
```

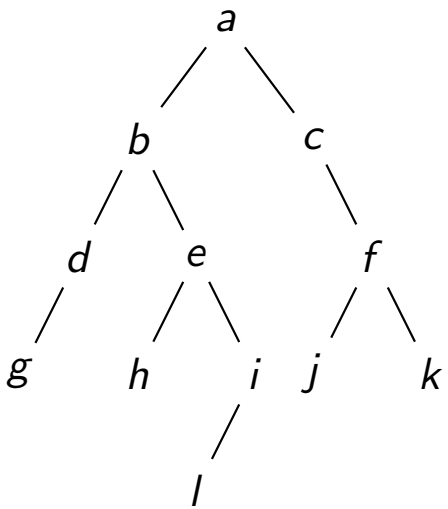
Counting Nodes of a Binary Tree

- One needs to visit all the nodes.
- Count the current node, and
- recursively visit the left sub-tree and the right sub-tree.

```
int count (Node *Root) {
   if (pRoot==null)
      return 0;
   else
      return 1 + count(pRoot->pLeft) + count(pRoot->pRight);
}
```

Exercise: Write an equivalent iterative code for counting nodes.

Height of a Tree



- The height of a binary tree is the number of levels of tree.
  - Tree height: 5.
- Height of left sub-tree: 4.
- Height of right sub-tree: 3.

Height of a Binary Tree

1. Get the height of left sub tree, say LeftHeight.
2. Get the height of right sub tree, say RightHeight.
3. Take $\max\{LeftHeight, RightHeight\}$ and add 1 for the root
4. Call recursively.

Height of a Binary Tree: C Code

```
/* height of binary tree */
int height (Node pRoot) {
   if (pRoot==null)
      return 0;
   else
      return 1 + max(height(root->pLeft), height (root->pRight));
}
```

Copying a Binary Tree

- Copy the current node.
- Recursively call the routine for left sub-tree and right sub-tree.

```
BTNode *Copy (BTNode pRoot) {
   if (root == null) {
      return null;
   }

   {BTNode *copy = NULL;
   copy = (BTNode *)malloc(sizeof(BTNode));
   copy->nData = pRoot->nData; }

   copy->pLeft = Copy(root->pLeft);
   copy->pRight = Copy(root->pRight);

   return copy;
}
```

Thank You for your kind attention!

Books Consulted

1. Chapter 4.3.3 of *Introduction to Algorithms: A Creative Approach* by Udi Manber.

2. Chapter 12 of *Introduction to Algorithms* by Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein.

Thank You for your kind attention!

Questions!!