



Binary Trees

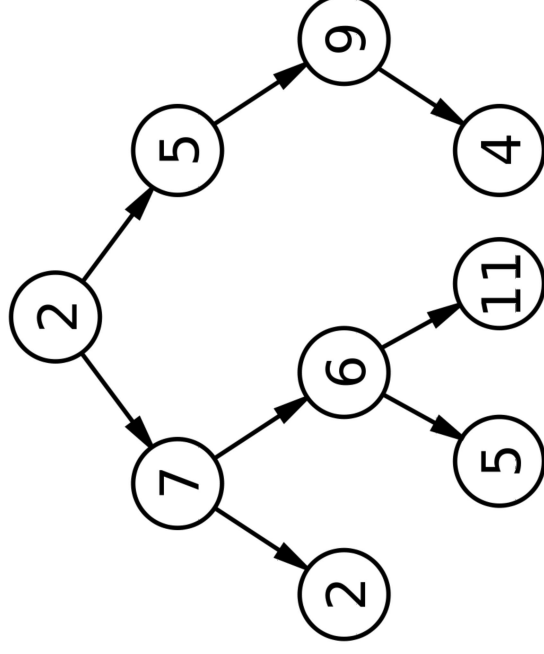


Introduction



Binary Tree

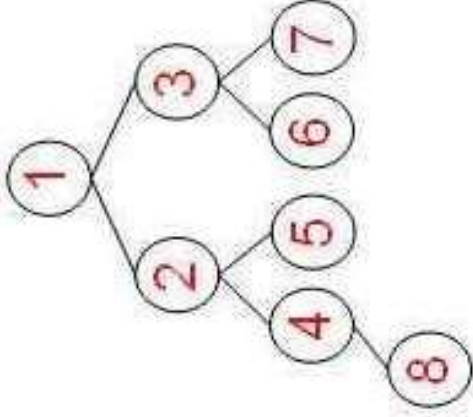
A **binary tree** is a tree data structure in which each node has at most two children, which are referred to as the *left child* and the *right child*.



Types of Binary Trees

Complete Binary Tree:

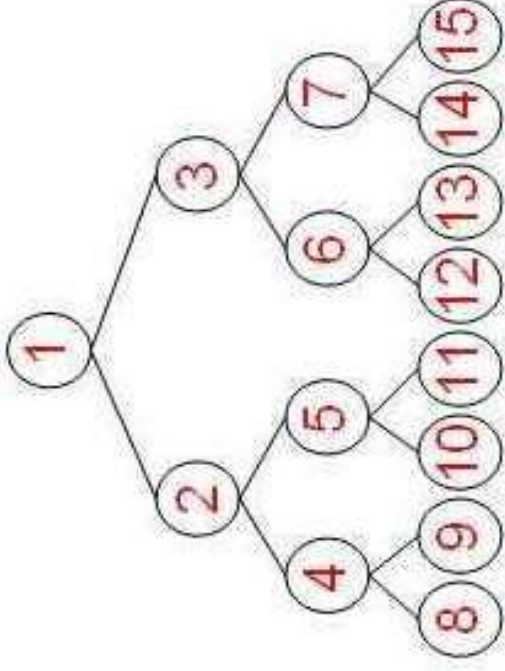
In a **complete** binary tree every level, except possibly the *last*, is completely filled, and all nodes in the last level are as far left as possible.



(a). Complete Tree

Full Binary Tree:

A **full** binary tree (sometimes referred to as a **proper** or **plane** binary tree) is a tree in which every node has either 0 or 2 children.



(b). Full Tree

Binary Tree Traversals

Tree Traversal: Process of visiting (checking and/or updating) each node in a tree data structure, exactly once.



Binary Tree Traversals

Tree Traversal: Process of visiting (checking and/or updating) each node in a tree data structure, exactly once.

Unlike linked lists, one-dimensional arrays etc., which are canonically traversed in linear order, trees may be traversed in multiple ways. They may be traversed in depth-first or breadth-first order.

Binary Tree Traversals

Tree Traversal: Process of visiting (checking and/or updating) each node in a tree data structure, exactly once.

Unlike linked lists, one-dimensional arrays etc., which are canonically traversed in linear order, trees may be traversed in multiple ways. They may be traversed in depth-first or breadth-first order.

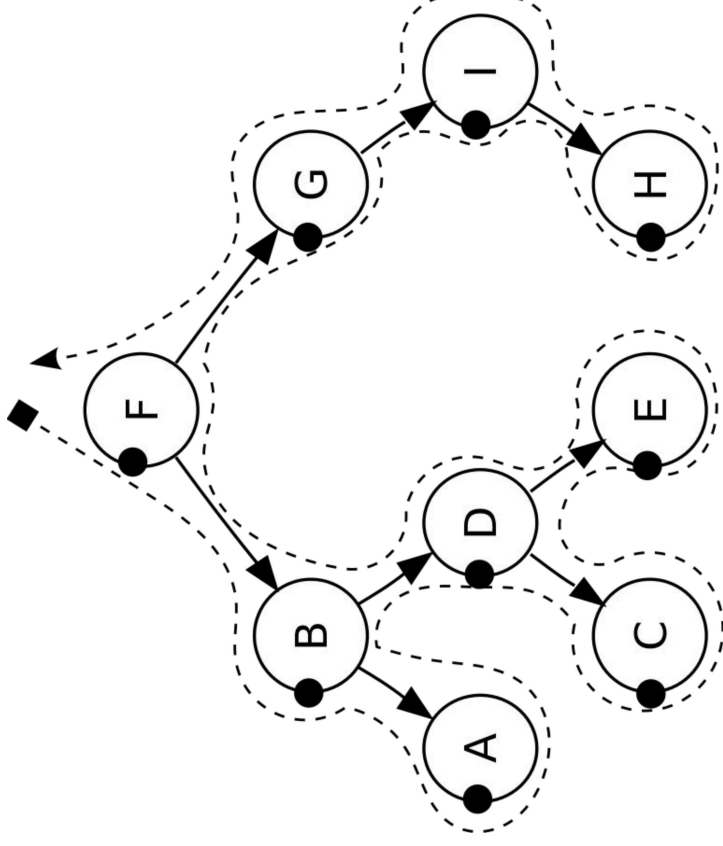
There are three common ways to traverse them in depth-first order: in-order, pre-order and post-order.



Pre-order Traversal

1. Check if the current node is empty / null.
2. Display the data part of the root (or current node).
3. Traverse the left subtree by recursively calling the pre-order function.
4. Traverse the right subtree by recursively calling the pre-order function.

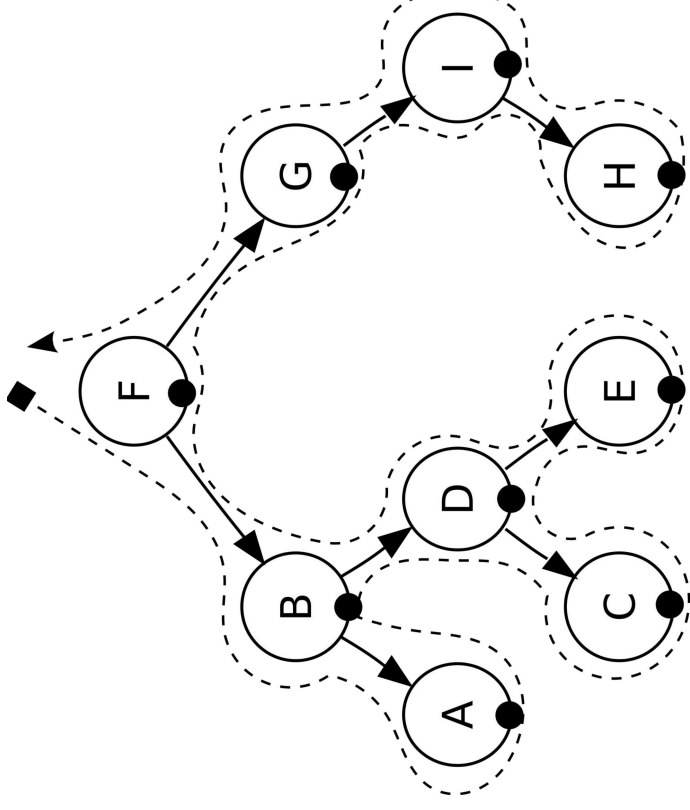
Pre-order: F, B, A, D, C, E, G, I, H.



In-order Traversal

1. Check if the current node is empty / null.
2. Traverse the left subtree by recursively calling the in-order function.
3. Display the data part of the root (or current node).
4. Traverse the right subtree by recursively calling the in-order function.

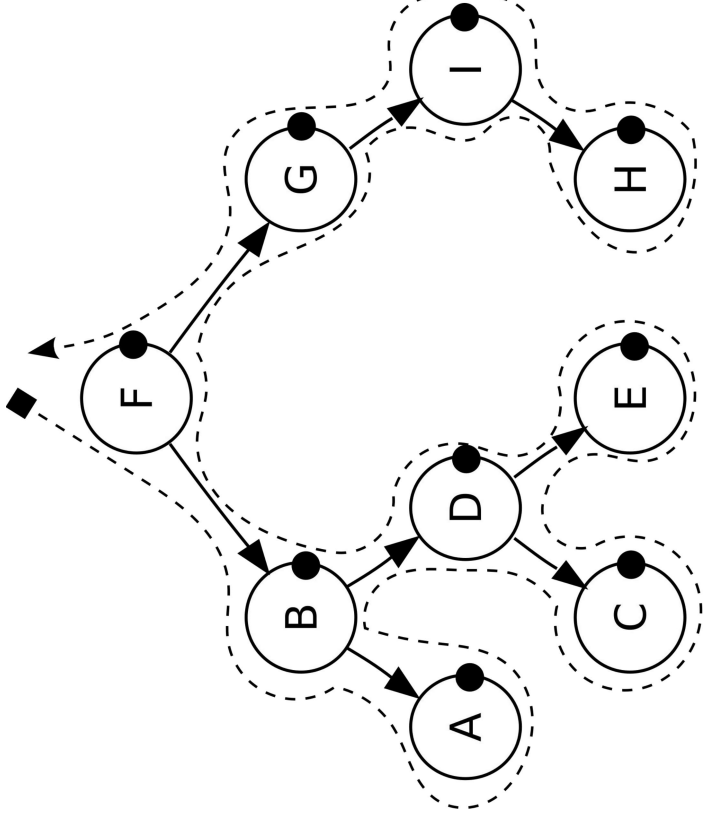
In-order: A, B, C, D, E, F, G, H, I.



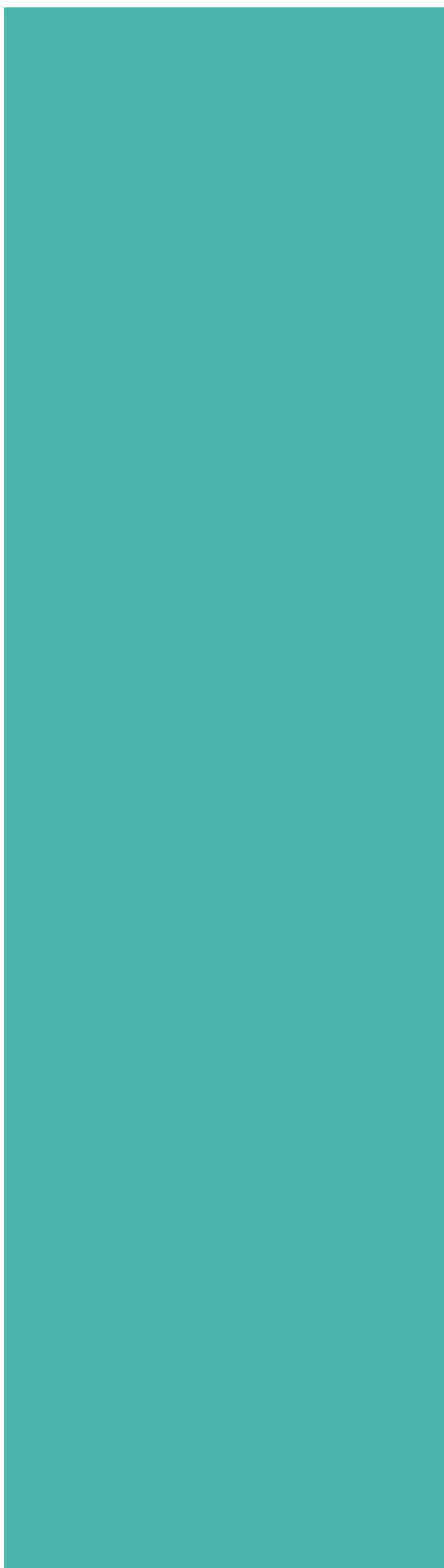
Post-order Traversal

1. Check if the current node is empty / null.
2. Traverse the left subtree by recursively calling the post-order function.
3. Traverse the right subtree by recursively calling the post-order function.
4. Display the data part of the root (or current node).

Post-order: A, C, E, D, B, H, I, G, F.

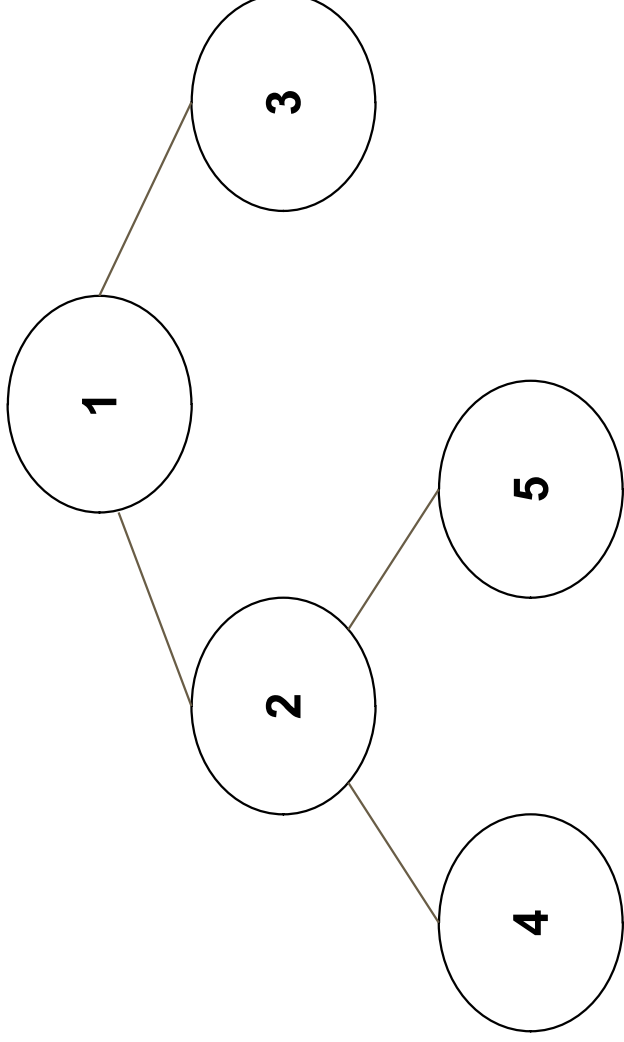


Level-order Traversal



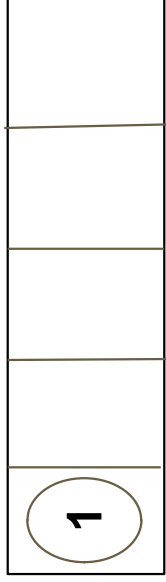
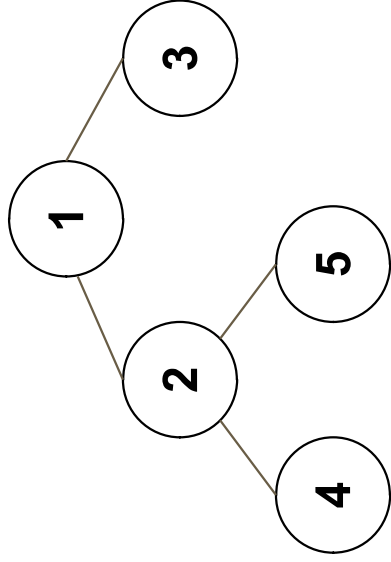
Level-order traversal

Level-order traversal:
1,2,3,4,5

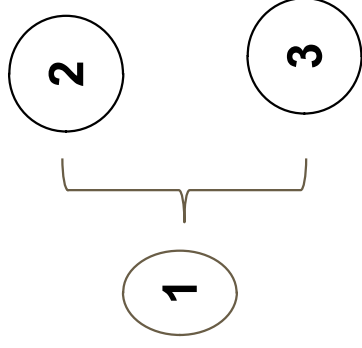
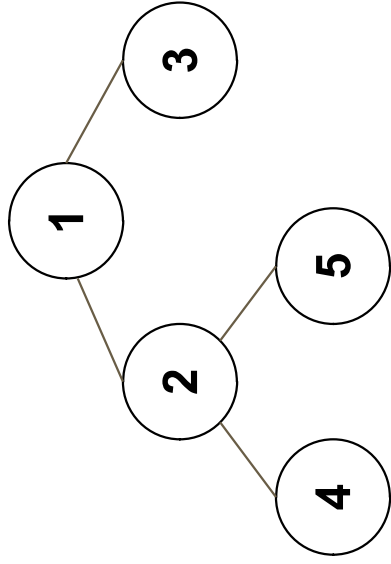


Level-order traversal

Level-order traversal:
1,



Level-order traversal

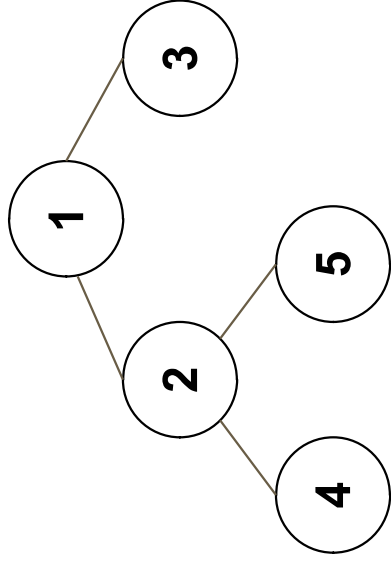


--	--	--	--	--



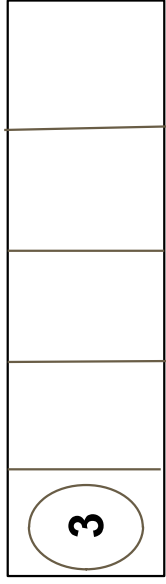
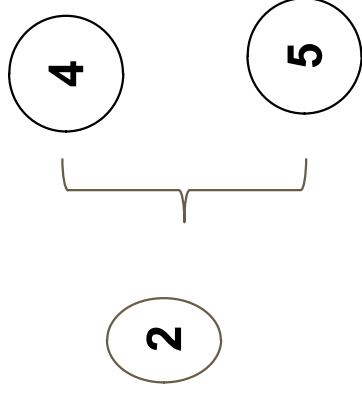
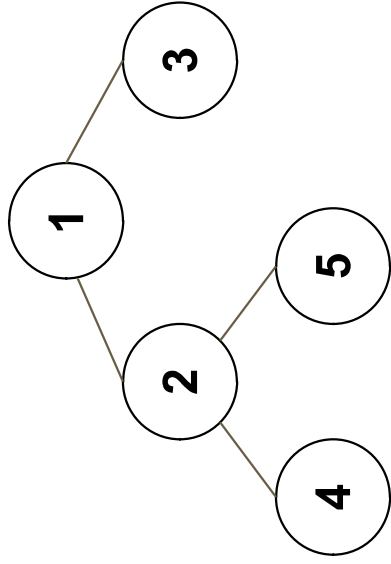
Level-order traversal

Level-order traversal:
1,2



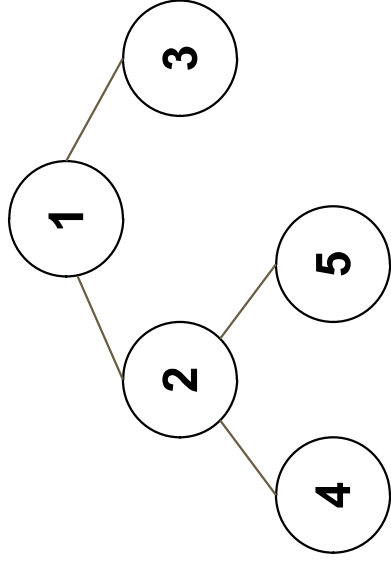
2	3			
---	---	--	--	--

Level-order traversal



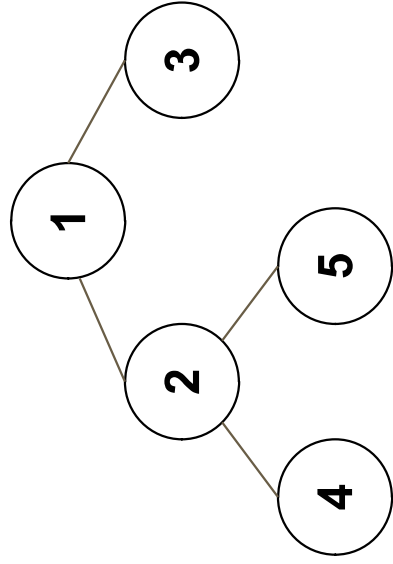
Level-order traversal

Level-order traversal:
1,2,3



3	4	5	
---	---	---	--

Level-order traversal



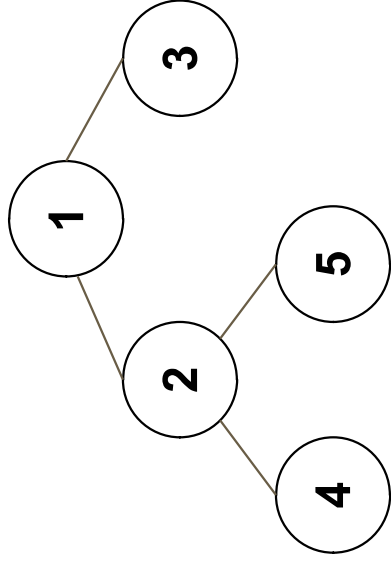
Level-order traversal:
1,2,3,4

3

4	5			
---	---	--	--	--

Level-order traversal

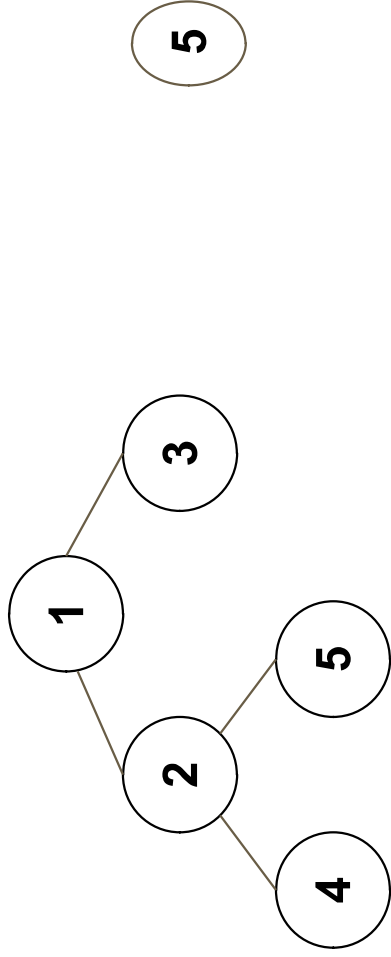
Level-order traversal:
1,2,3,4,5



4

5				
---	--	--	--	--

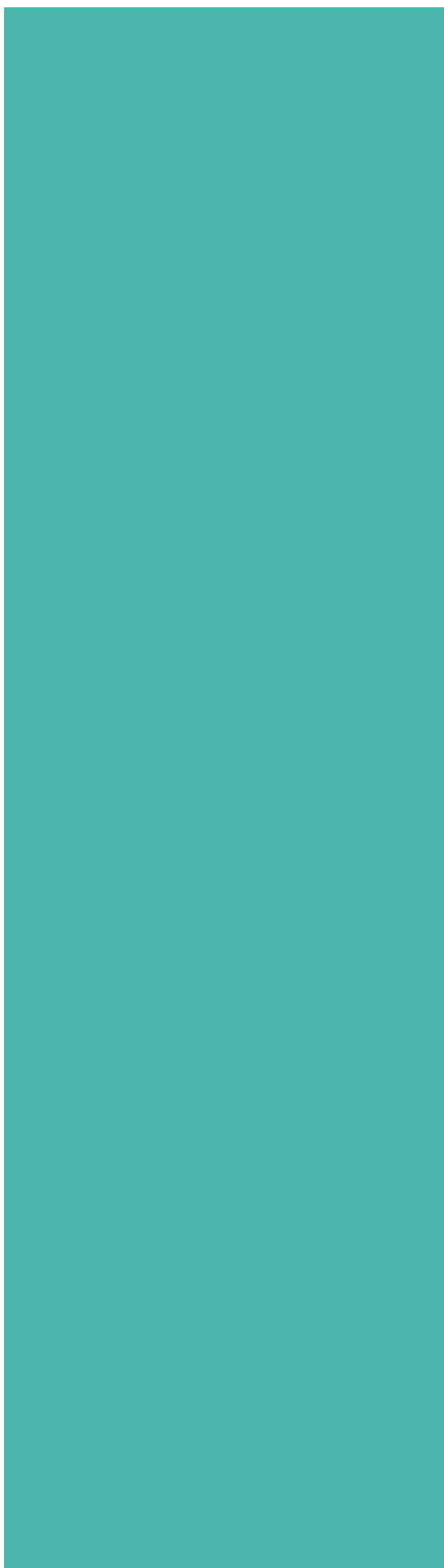
Level-order traversal



--	--	--	--	--

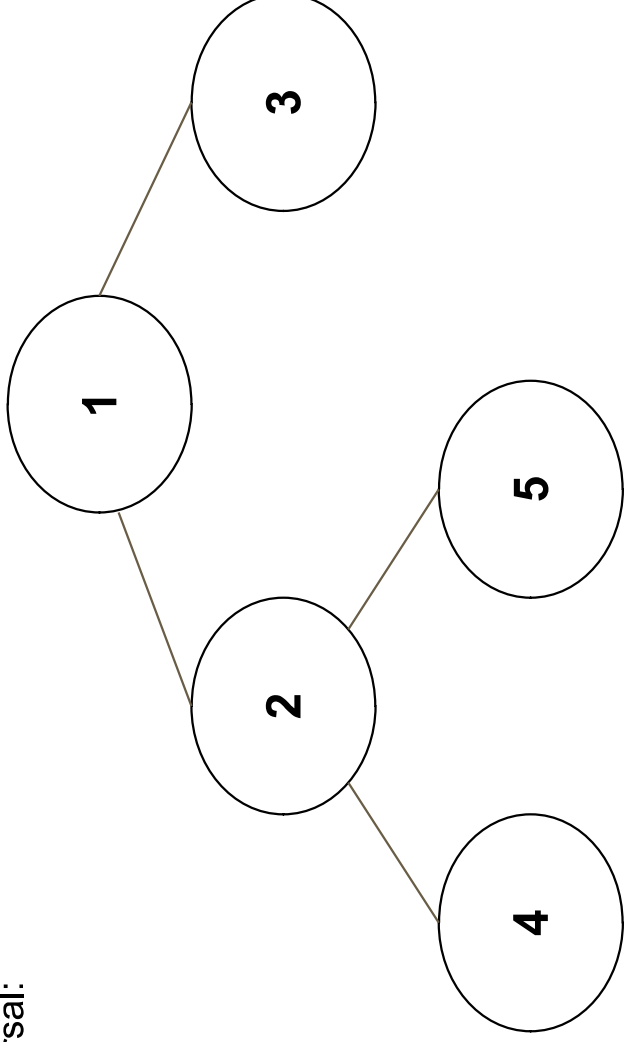


Reverse Level-order Traversal

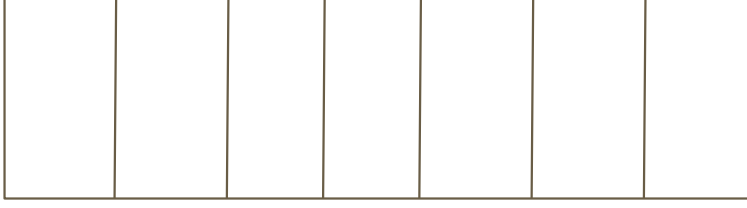
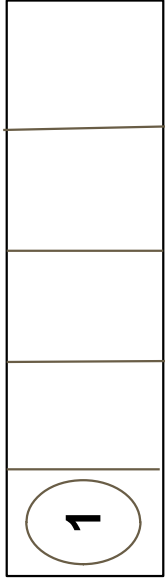
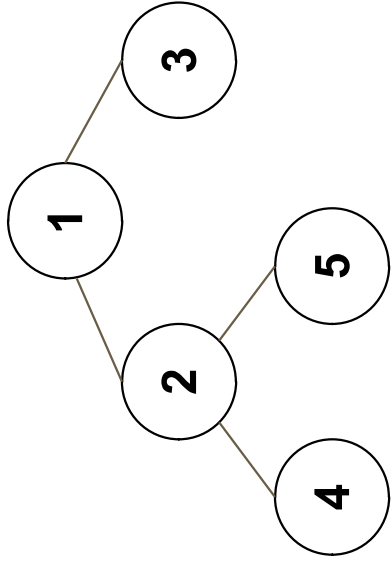


Reverse Level-order traversal

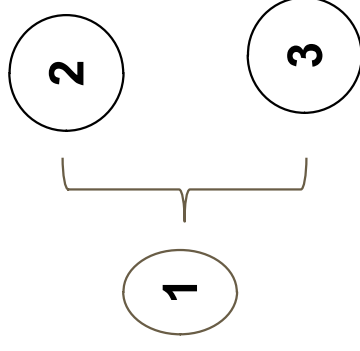
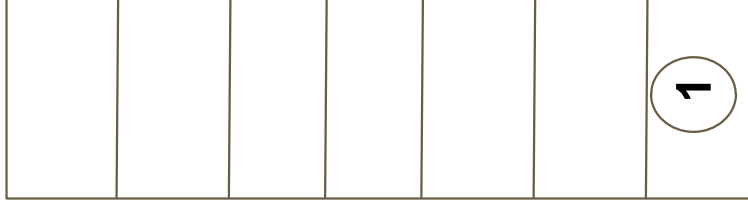
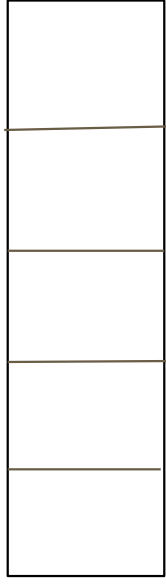
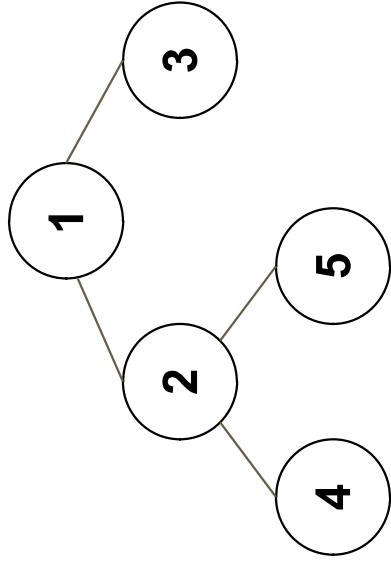
Reverse Level-order traversal:
4,5,2,3,1



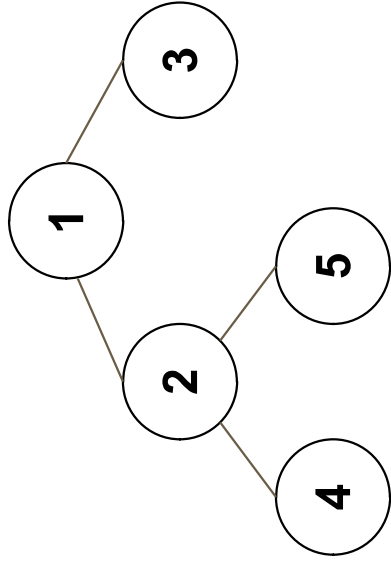
Reverse Level-order traversal



Reverse Level-order traversal



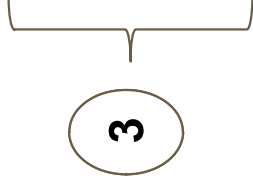
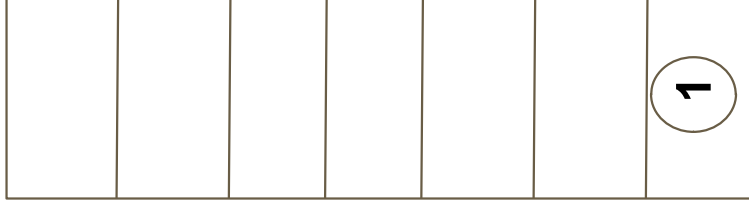
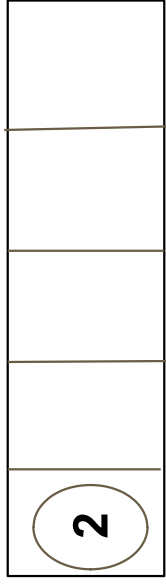
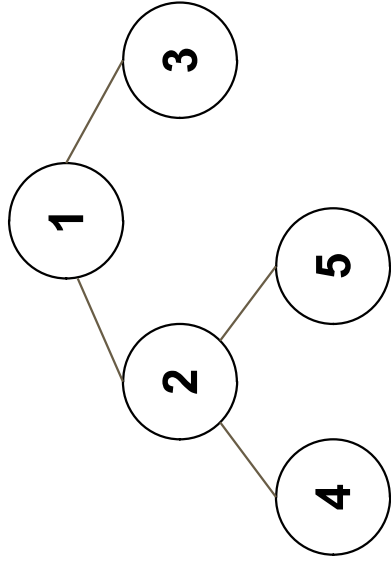
Reverse Level-order traversal



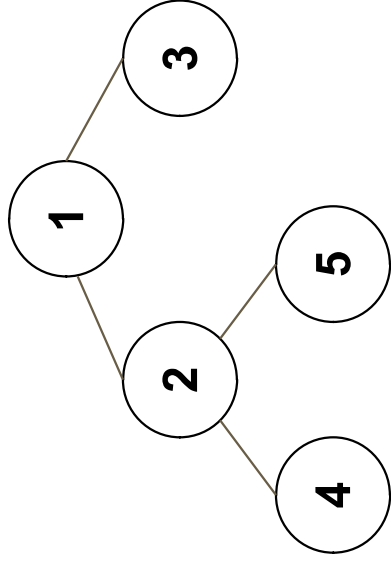
3	2				
---	---	--	--	--	--

						1
--	--	--	--	--	--	---

Reverse Level-order traversal



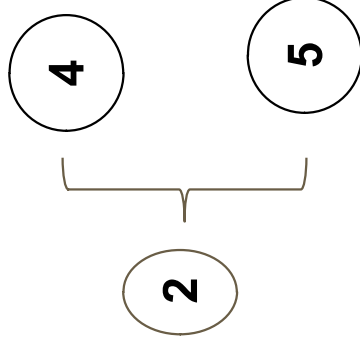
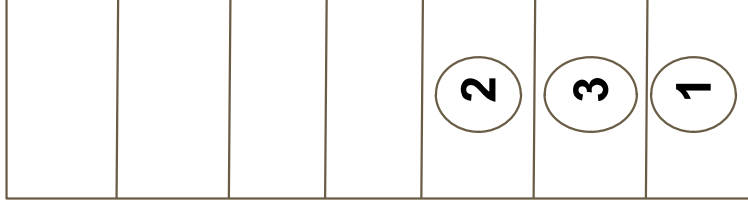
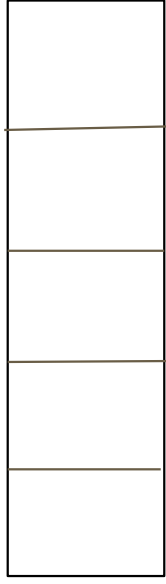
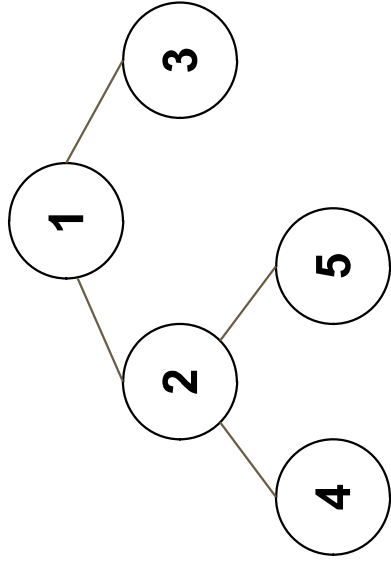
Reverse Level-order traversal



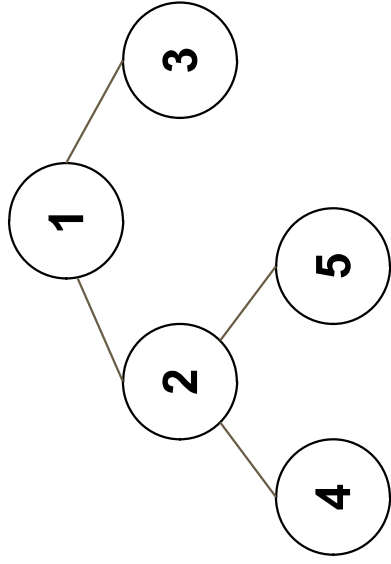
2				
---	--	--	--	--

						3	1
--	--	--	--	--	--	---	---

Reverse Level-order traversal



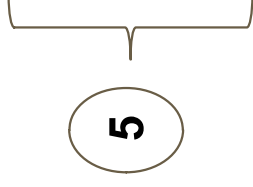
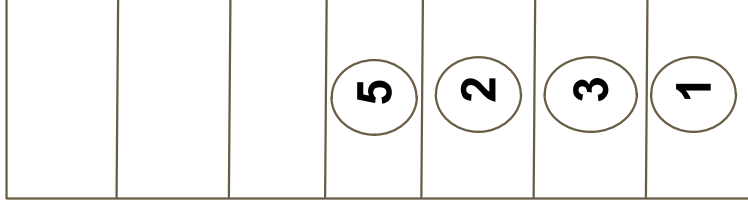
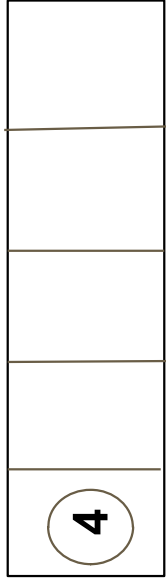
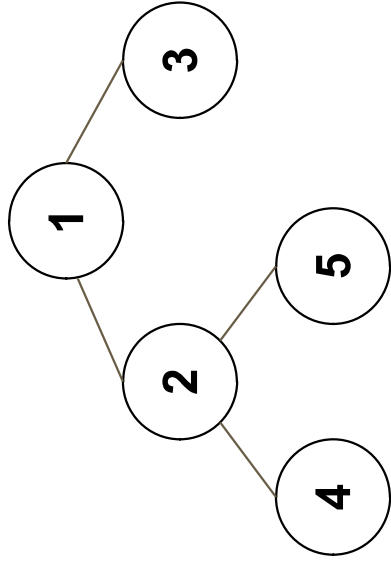
Reverse Level-order traversal



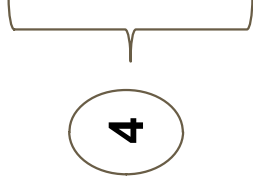
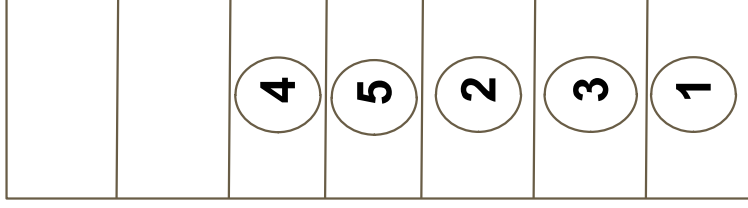
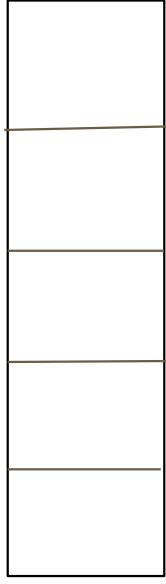
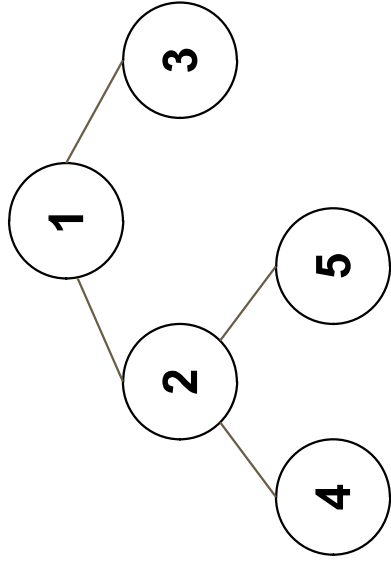
5	4				
---	---	--	--	--	--

	2	3	1		
--	---	---	---	--	--

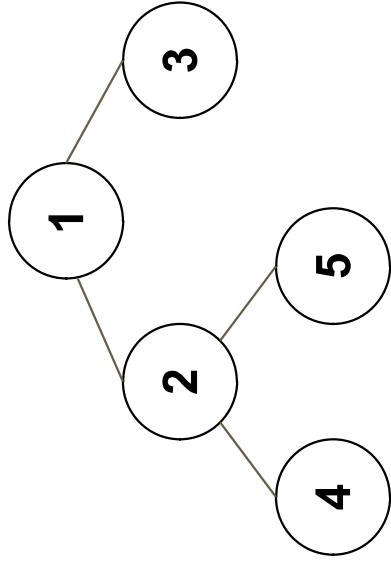
Reverse Level-order traversal



Reverse Level-order traversal



Reverse Level-order traversal



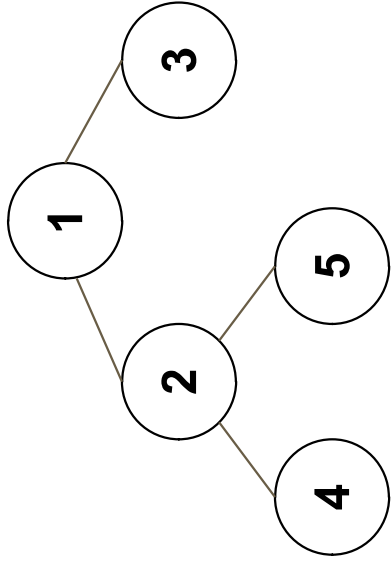
--	--	--	--	--

			5	2	3	1

4

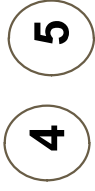


Reverse Level-order traversal

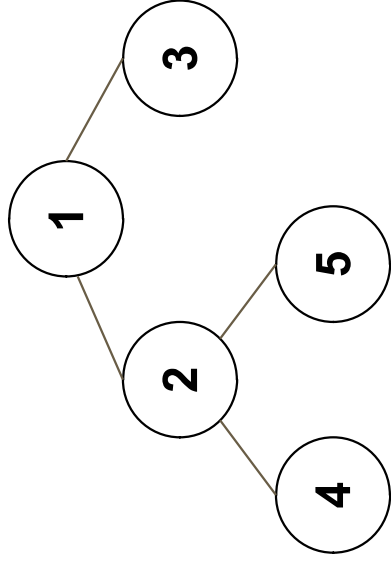


--	--	--	--	--

				2	3	1



Reverse Level-order traversal



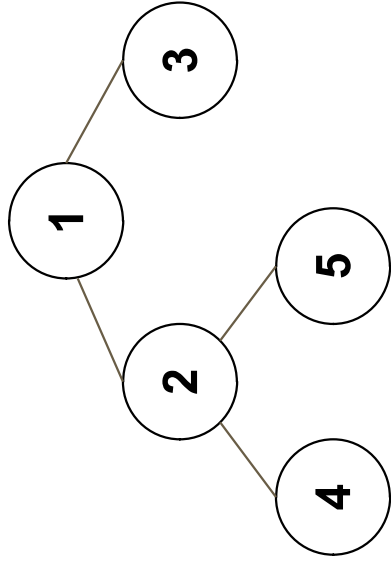
					3	1
--	--	--	--	--	---	---

2
5
4

5

2

Reverse Level-order traversal



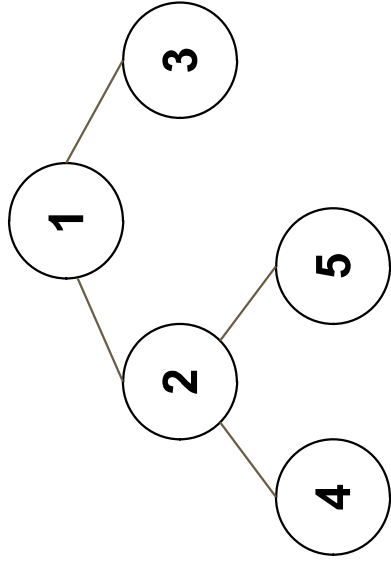
--	--	--	--	--	--

						1

- 4 5 2 3



Reverse Level-order traversal

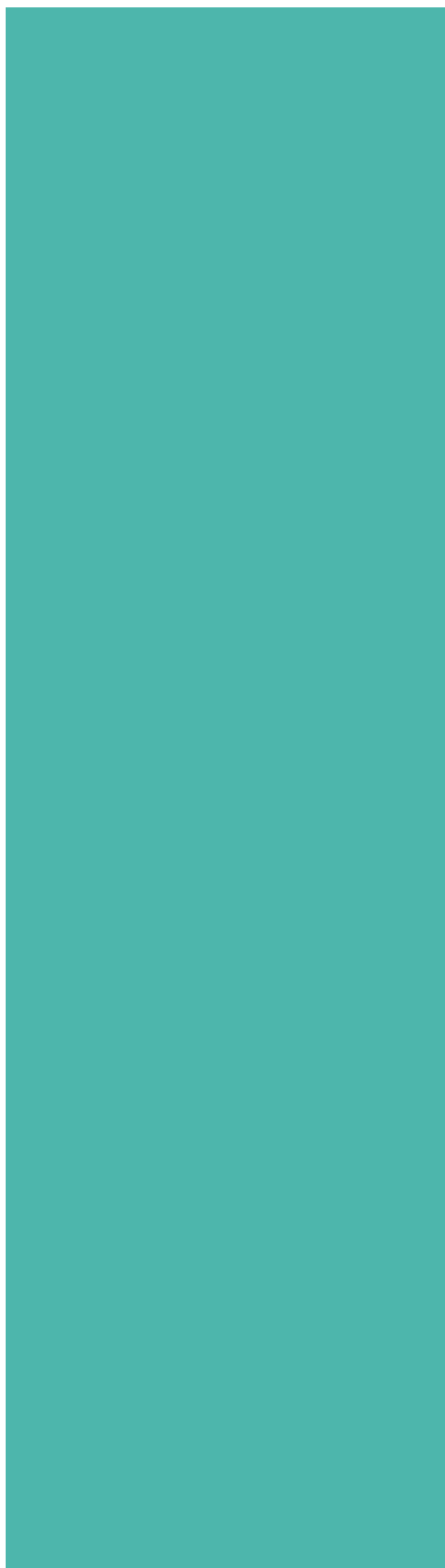


--	--	--	--	--	--

--	--	--	--	--	--	--



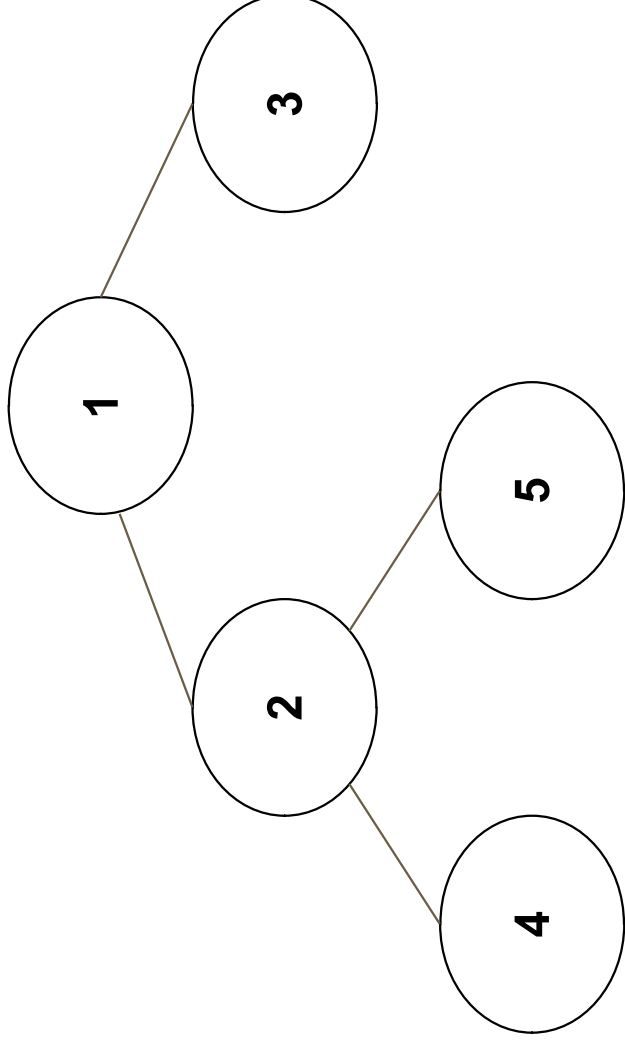
Size of Binary Tree



Calculate Size of Binary Tree

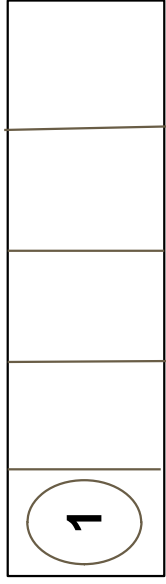
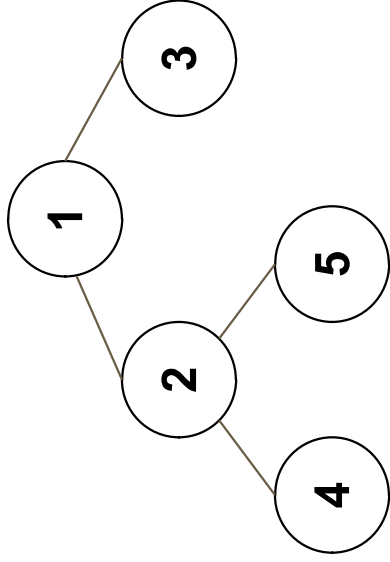
Size of Tree:

The total number of nodes in the tree.

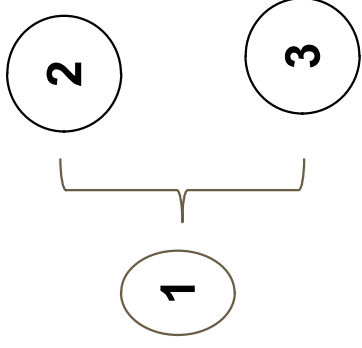
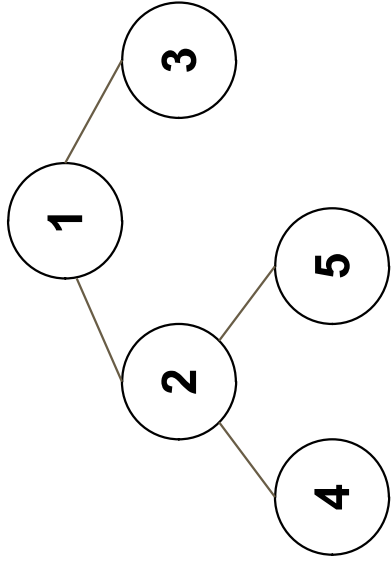


Size of Binary Tree

Size: 1



Size of Binary Tree

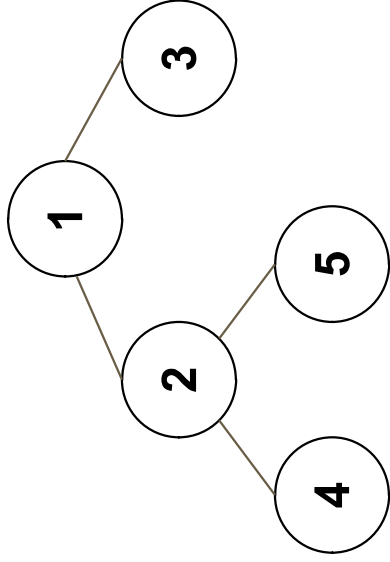


--	--	--	--	--



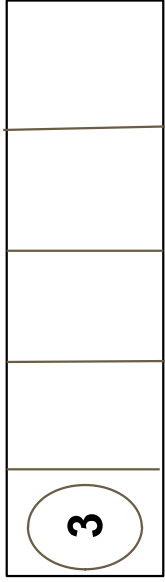
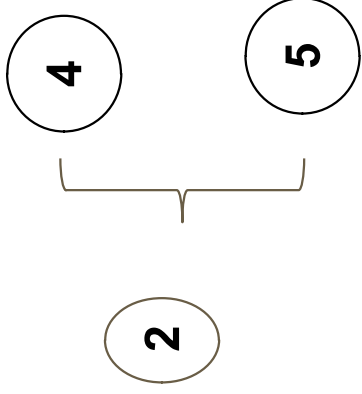
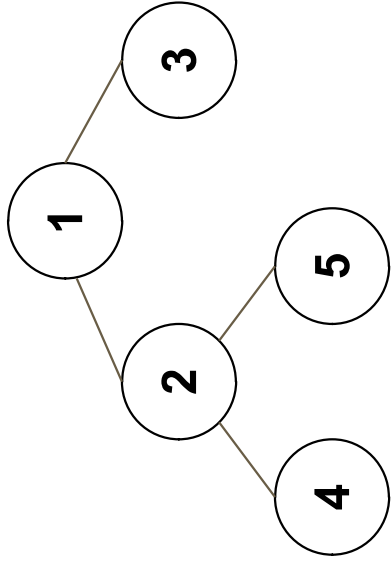
Size of Binary Tree

Size: 3



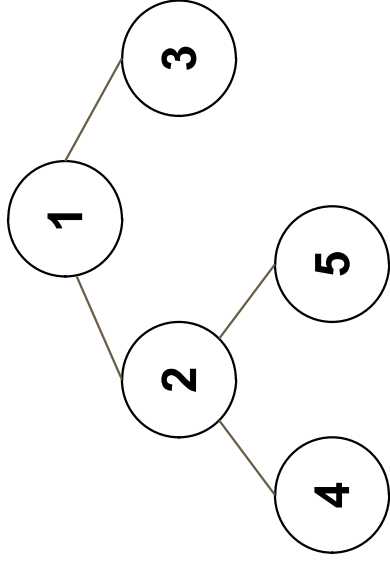
2	3			
---	---	--	--	--

Size of Binary Tree



Size of Binary Tree

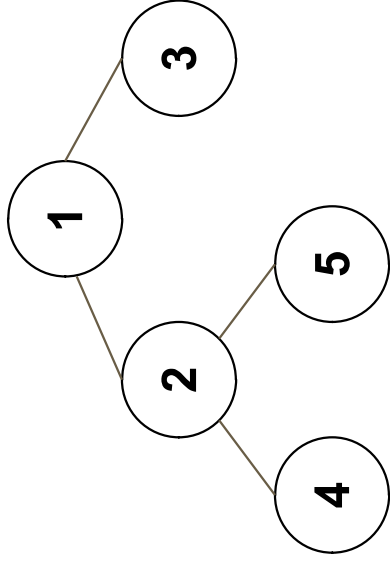
Size: 5



3	4	5		
---	---	---	--	--

Size of Binary Tree

Size: 5

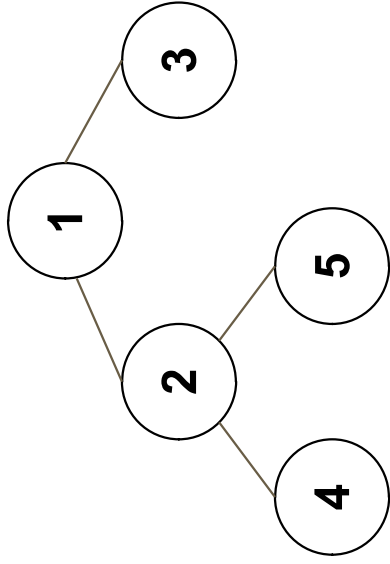


3

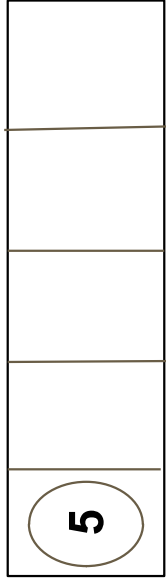
4	5			
---	---	--	--	--

Size of Binary Tree

Size: 5

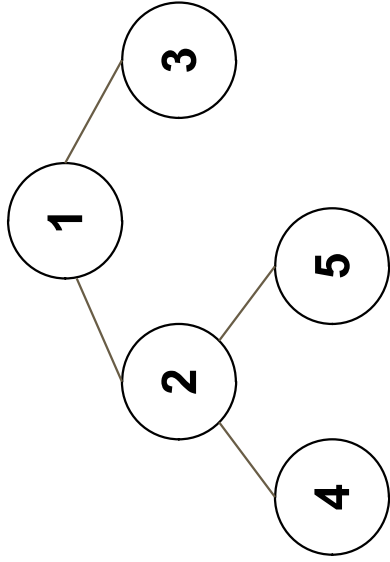


4



Size of Binary Tree

Size: 5

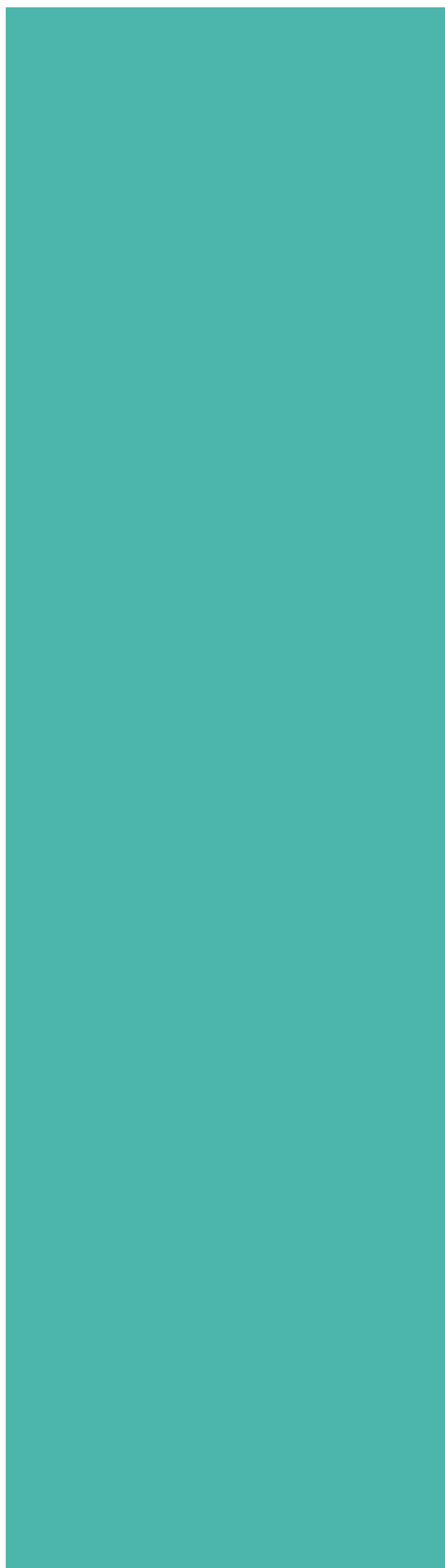


5

--	--	--	--	--



Height of Binary Tree



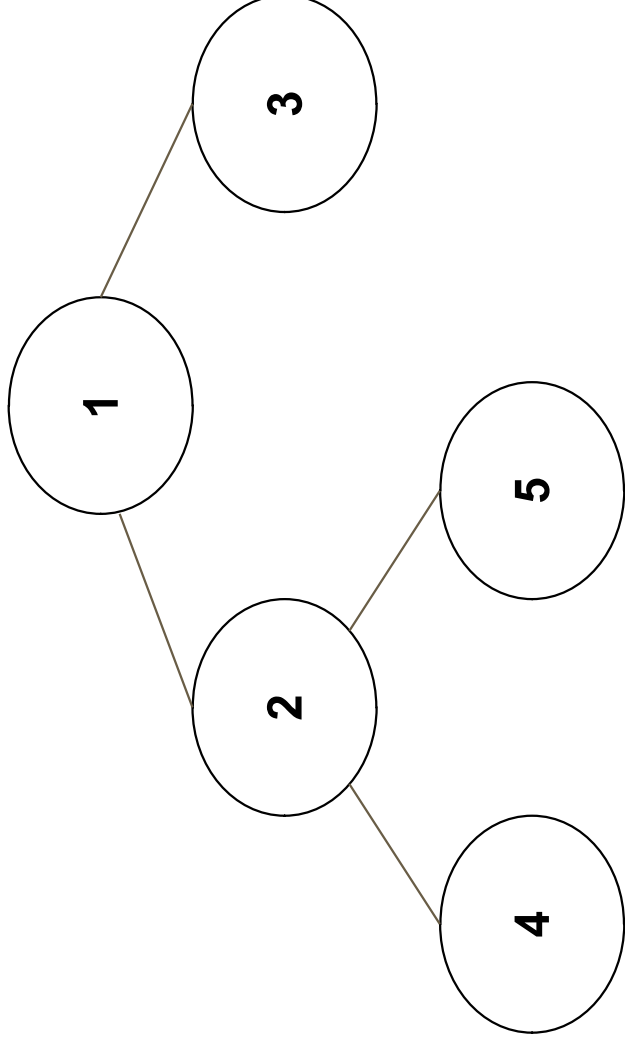
Calculate Height of Binary Tree

Height of Tree:

The height of a tree is the height of its root node.

Height of Node:

The height of a node is the number of edges on the longest path between that node and a leaf.



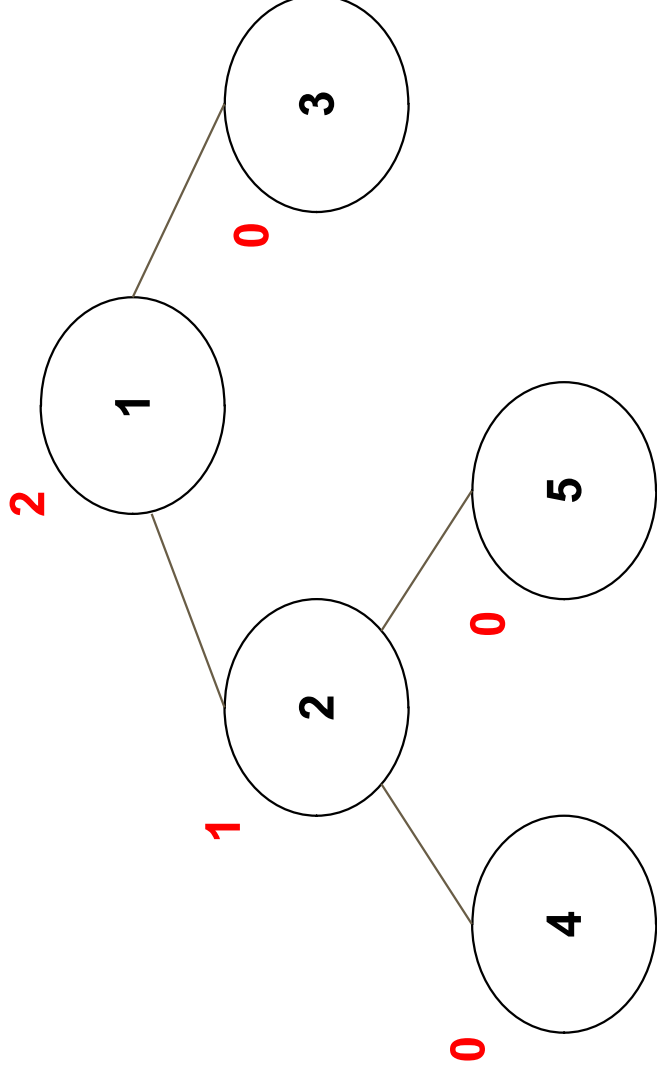
Calculate Height of Binary Tree

Height of Tree:

The height of a tree is the height of its root node.

Height of Node:

The height of a node is the number of edges on the longest path between that node and a leaf.



def height(node):

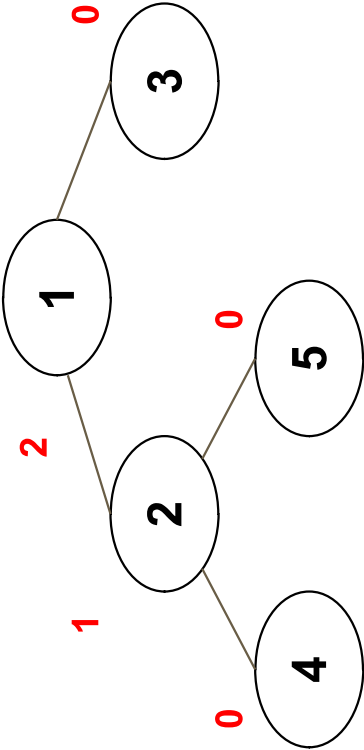
if node is None:

return -1

left_height = height(node.left)

right_height = height(node.right)

return 1 + max(left_height, right_height)



node:	left_height	right_height

def height(node):

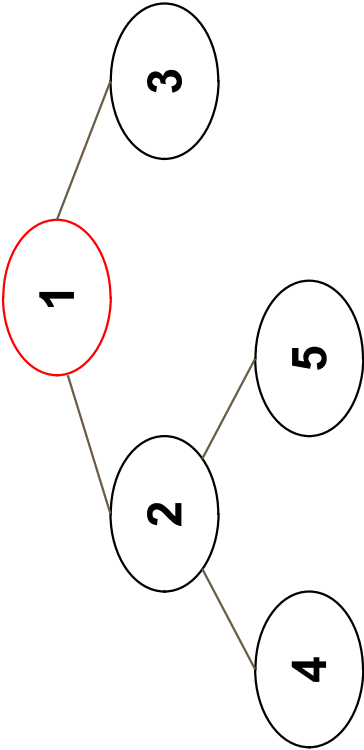
if node is None:

return -1

left_height = height(node.left)

right_height = height(node.right)

return 1 + max(left_height, right_height)



node:	left_height	right_height
1		



def height(node):

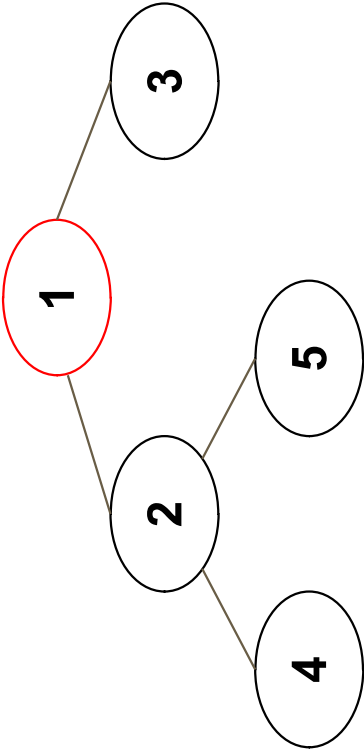
if node is None:

return -1

left_height = height(node.left)

right_height = height(node.right)

return 1 + max(left_height, right_height)



node:	left_height	right_height
1		



```
def height(node):
```

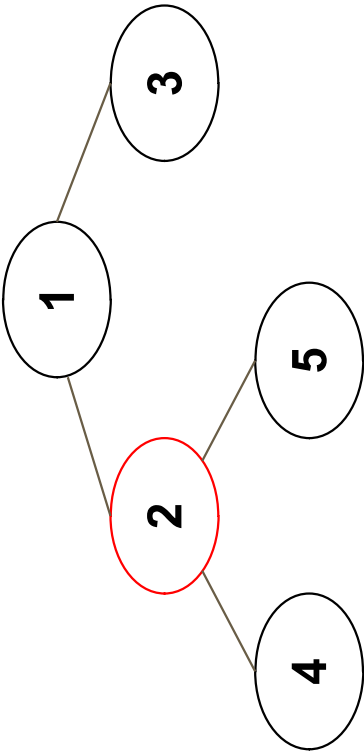
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		

```
def height(node):
```

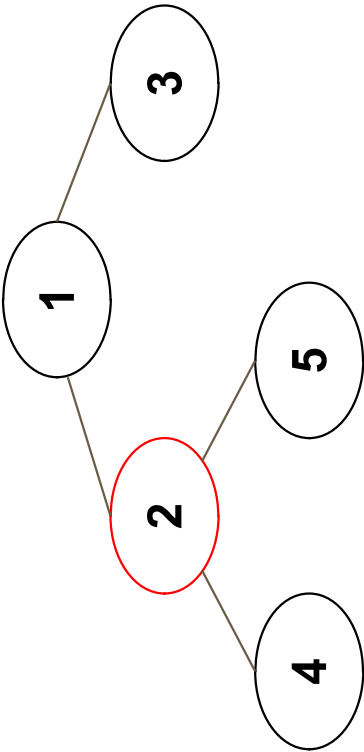
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		

```
def height(node):
```

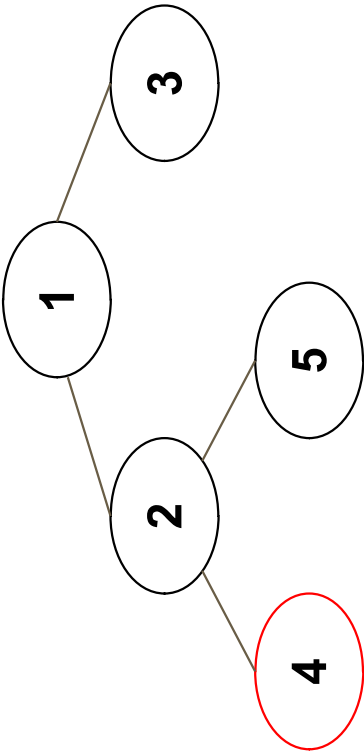
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
4		

```
def height(node):
```

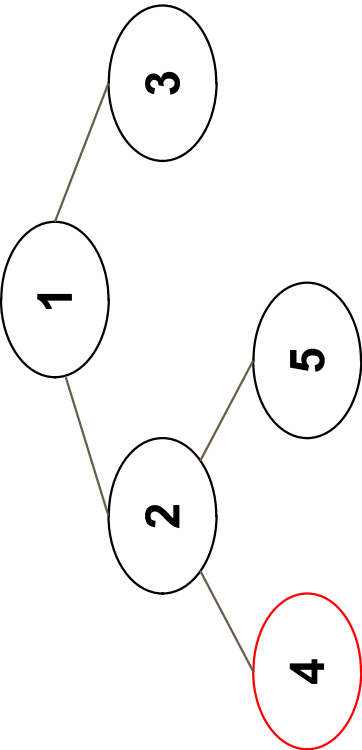
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
4		

```
def height(node):
```

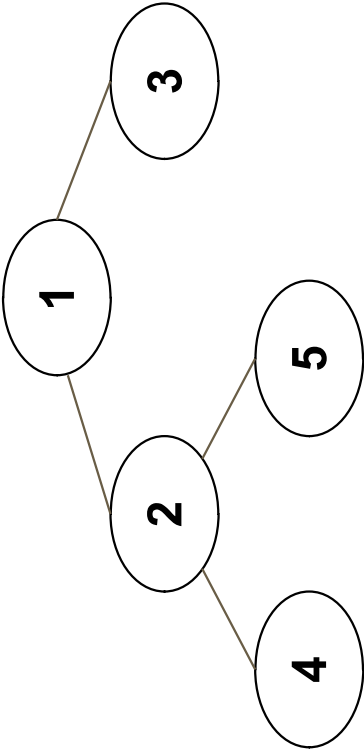
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
4		
None	-1	


```
def height(node):
```

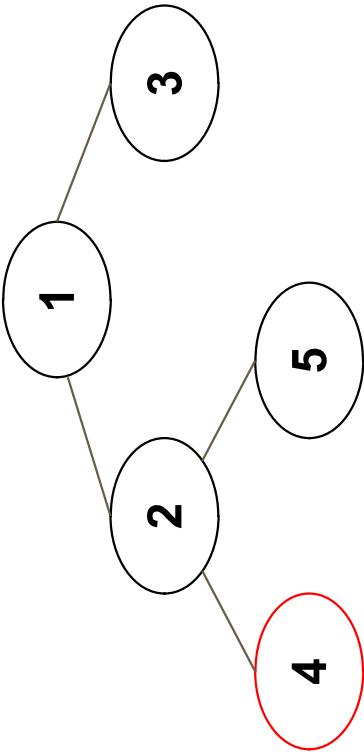
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
4		
None	-1	

```
def height(node):
```

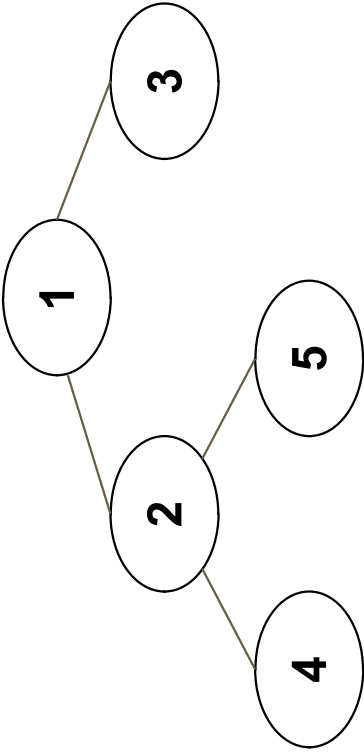
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
4		
None	-1	-1

```
def height(node):
```

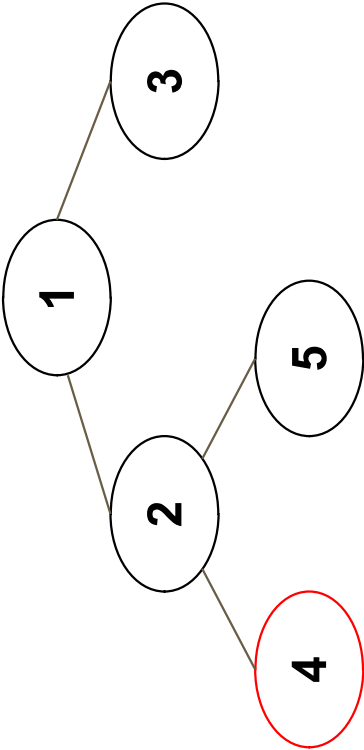
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
4		
None	-1	-1

```
def height(node):
```

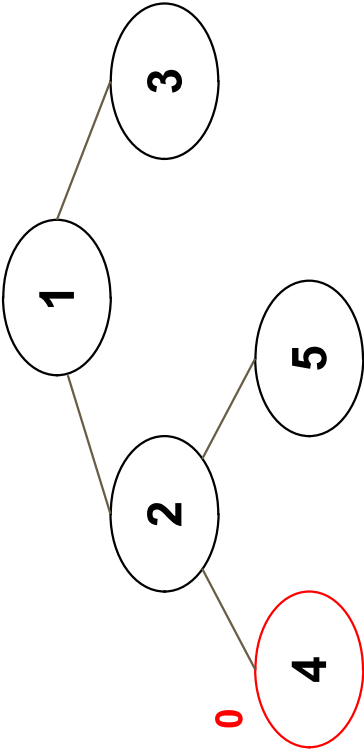
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
4		

```
def height(node):
```

```
    if node is None:
```

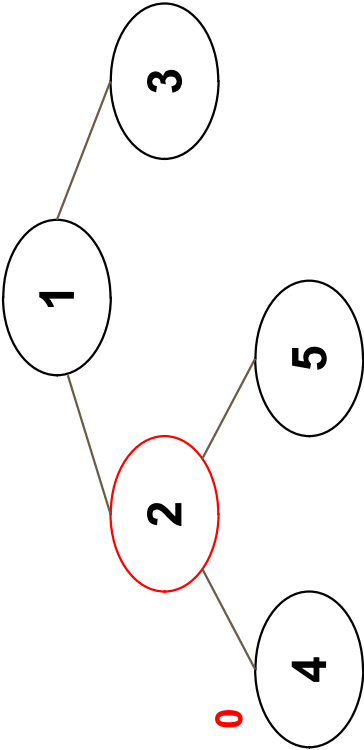
```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```

0



node:	left_height	right_height
1		
2		

```
def height(node):
```

```
    if node is None:
```

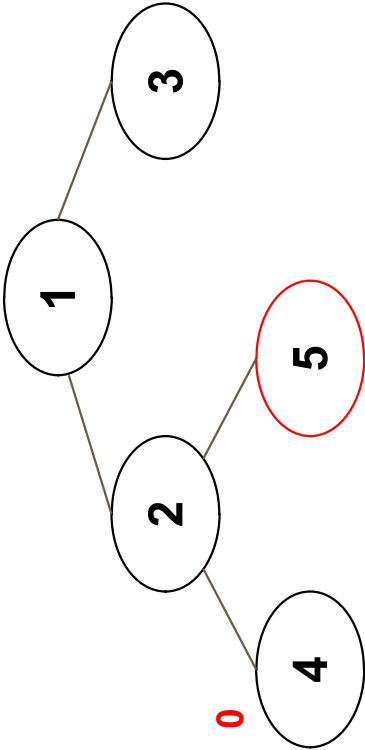
```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```

0



node:	left_height	right_height
1		
2		
5		

```
def height(node):
```

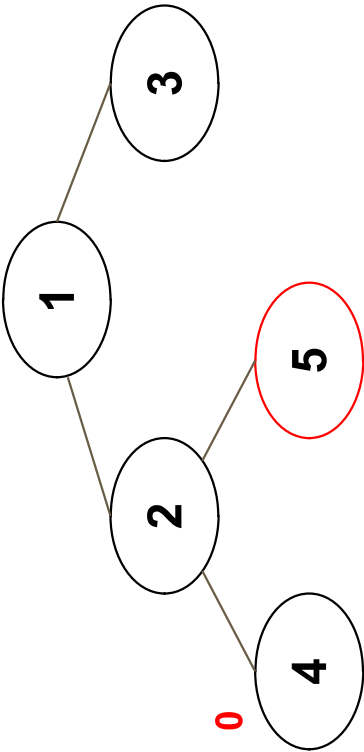
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
5		



```
def height(node):
```

```
    if node is None:
```

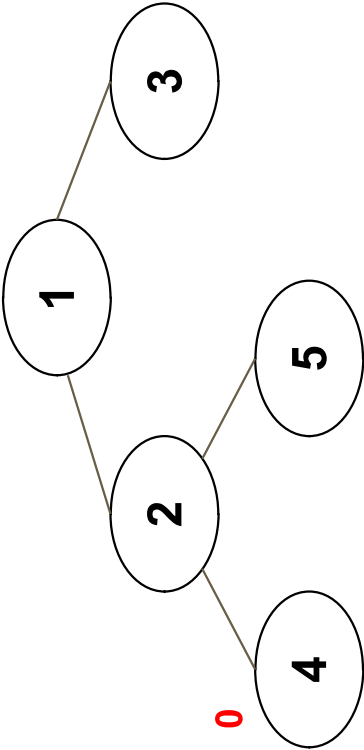
```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```

0



node:	left_height	right_height
1		
2		
5		
None	-1	


```
def height(node):
```

```
    if node is None:
```

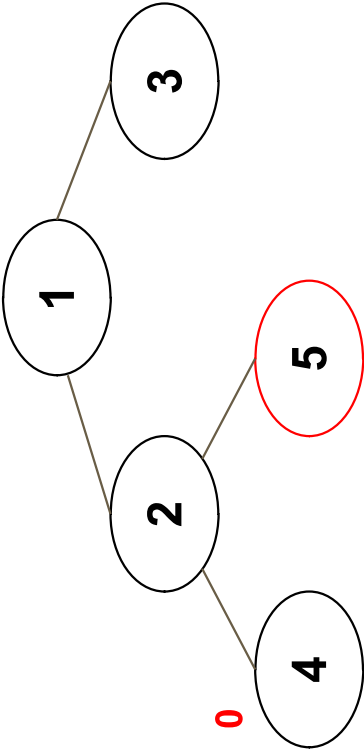
```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```

0



node:	left_height	right_height
1		
2		
5		
None	-1	

```
def height(node):
```

```
    if node is None:
```

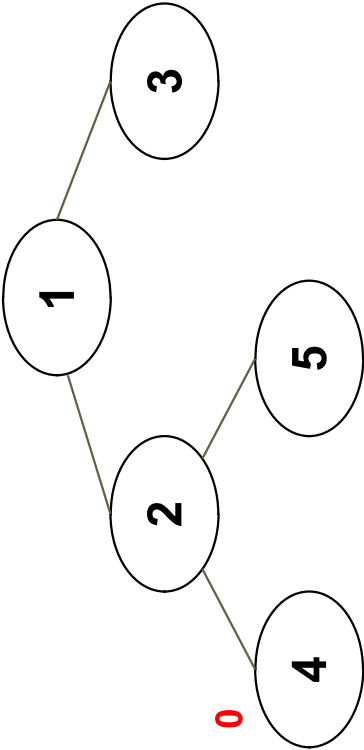
```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```

0



node:	left_height	right_height
1		
2		
5		
None	-1	-1

```
def height(node):
```

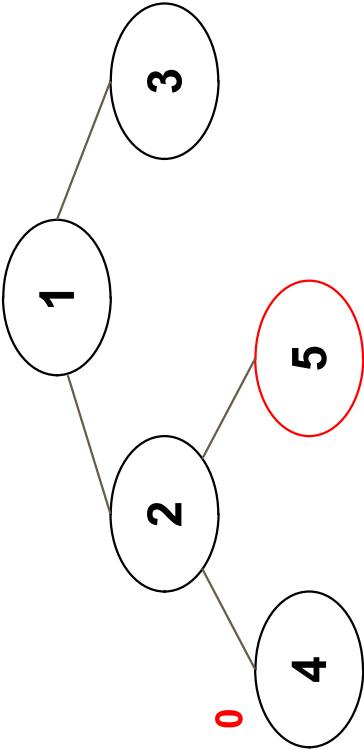
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
5		
None	-1	-1

```
def height(node):
```

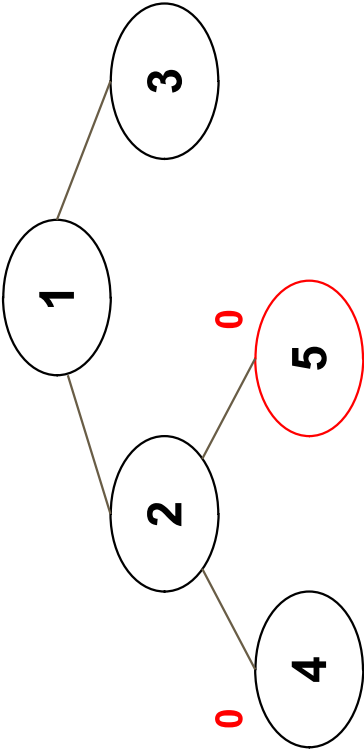
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		
5		

```
def height(node):
```

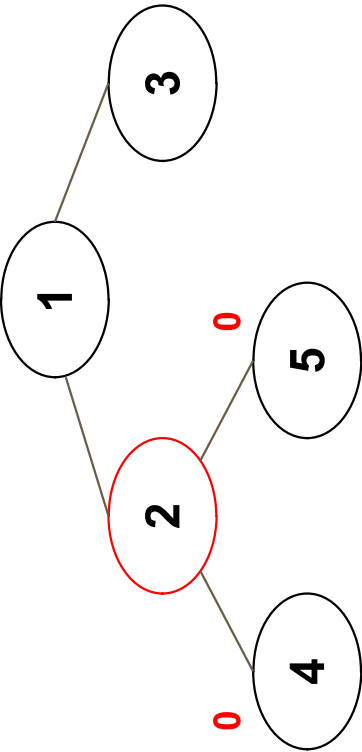
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		

```
def height(node):
```

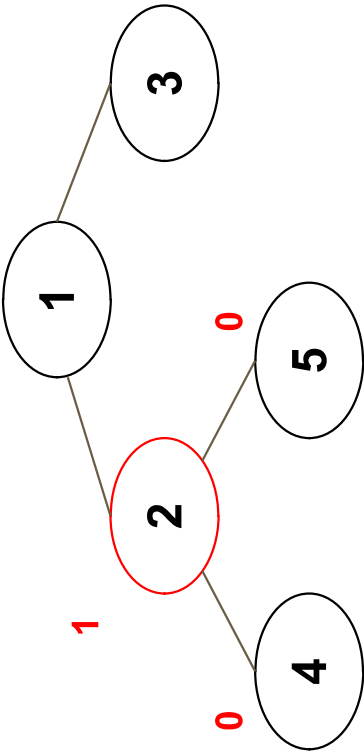
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1		
2		

def height(node):

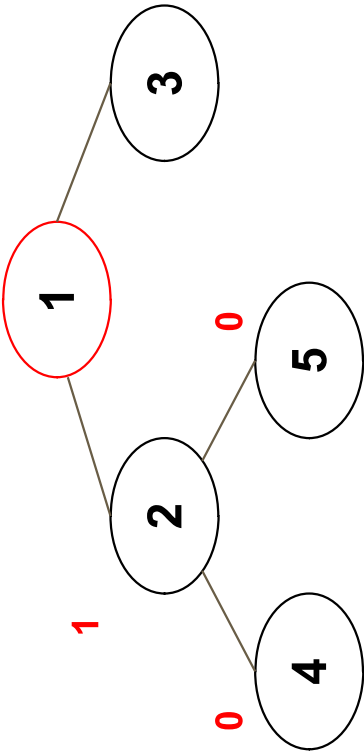
if node is None:

return -1

left_height = height(node.left)

right_height = height(node.right)

return 1 + max(left_height, right_height)



node:	left_height	right_height
1	1	

```
def height(node):
```

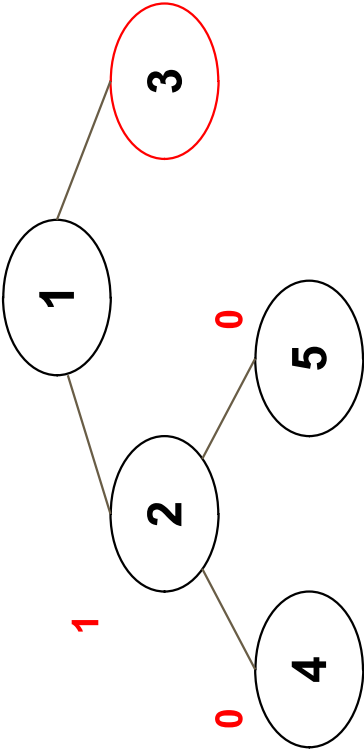
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1	1	
3		


```
def height(node):
```

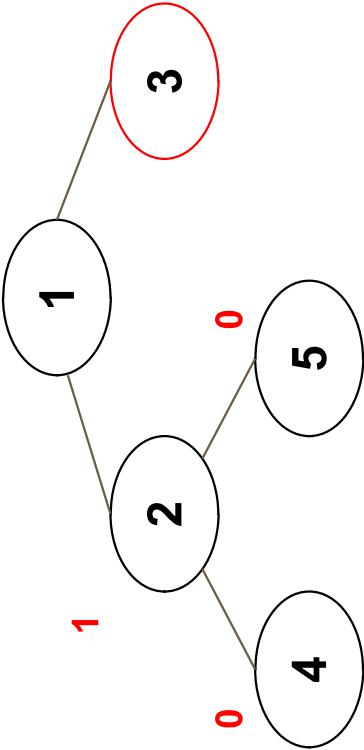
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



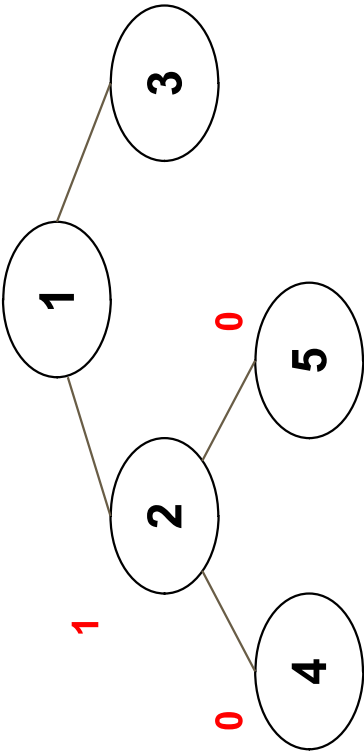
node:	left_height	right_height
1	1	
3		
None		

```
def height(node):
```

```
    if node is None:  
        return -1
```

```
    left_height = height(node.left)  
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1	1	
3		
None	-1	



```
def height(node):
```

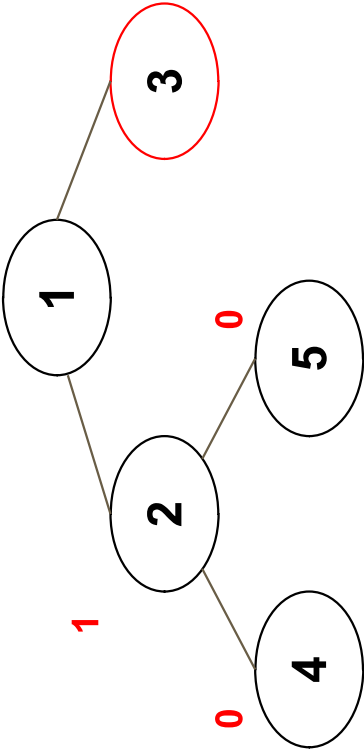
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



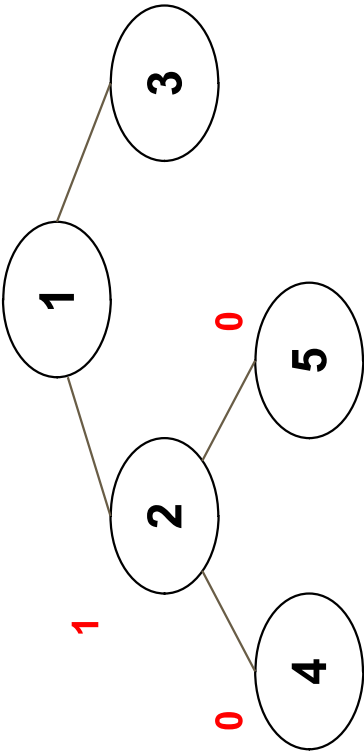
node:	left_height	right_height
1	1	
3		
None	-1	

def height(node):

if node is None:
return -1

left_height = height(node.left)
right_height = height(node.right)

return 1 + max(left_height, right_height)



node:	left_height	right_height
1	1	
3		
None	-1	-1

```
def height(node):
```

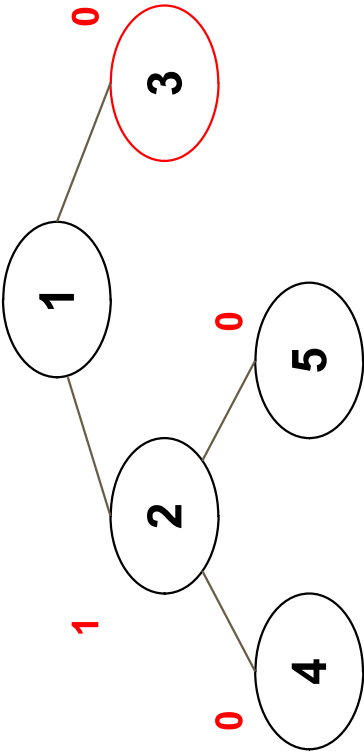
```
    if node is None:
```

```
        return -1
```

```
    left_height = height(node.left)
```

```
    right_height = height(node.right)
```

```
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height
1	1	
3		

def height(node):

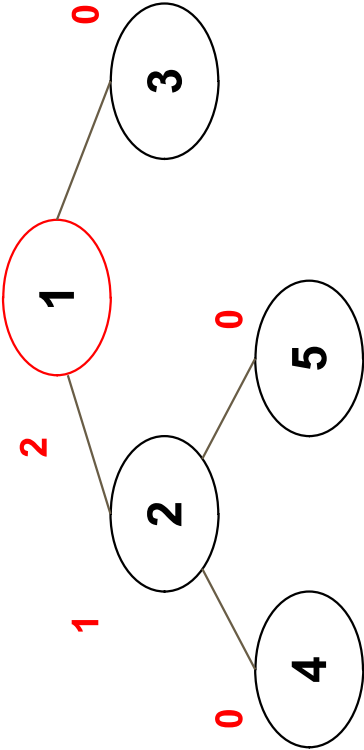
if node is None:

return -1

left_height = height(node.left)

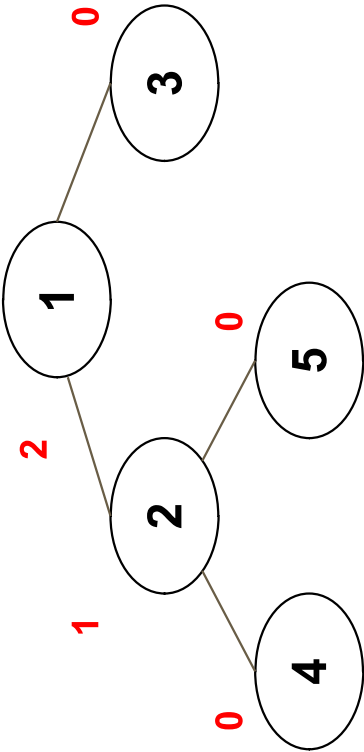
right_height = height(node.right)

return 1 + max(left_height, right_height)



node:	left_height	right_height
1	1	0

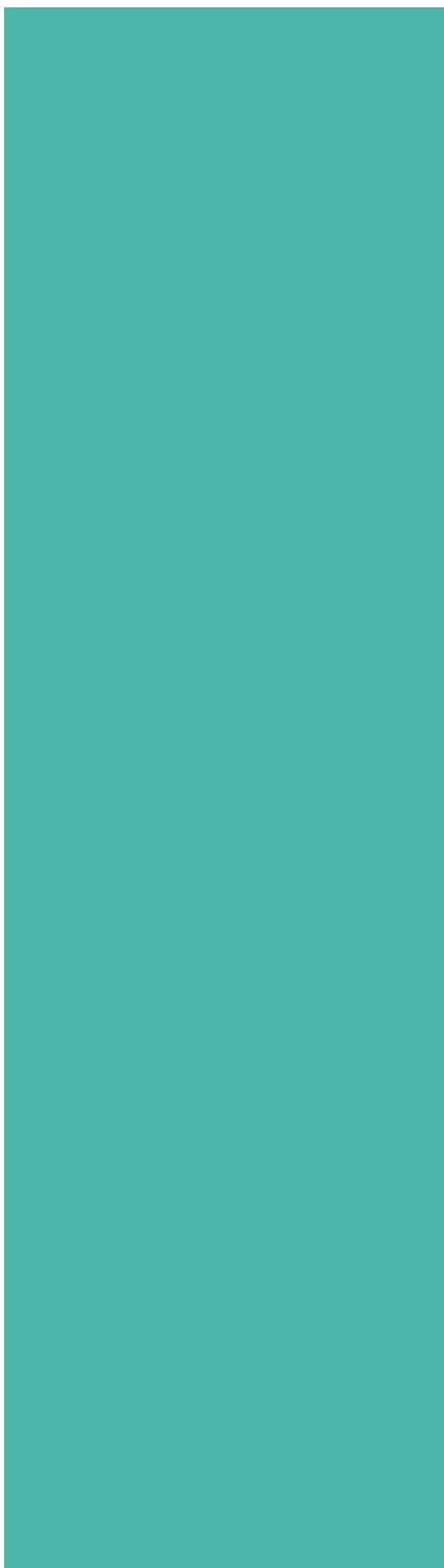
```
def height(node):  
    if node is None:  
        return -1  
    left_height = height(node.left)  
    right_height = height(node.right)  
    return 1 + max(left_height, right_height)
```



node:	left_height	right_height

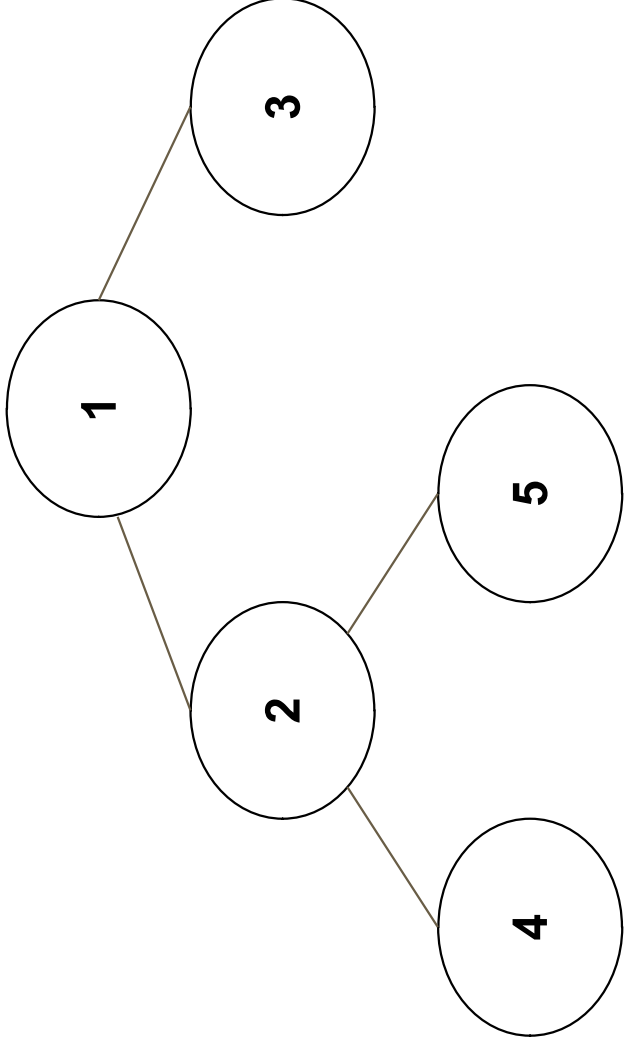


Pre-order Iterative



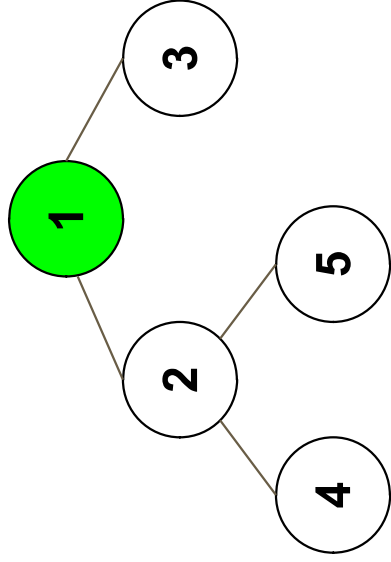
Pre-order Iterative

**Pre-order
output:
[1,2,4,5,3]**



Pre-order traversal

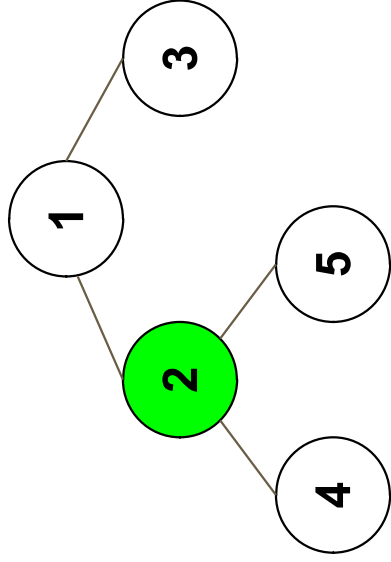
Pre-order traversal:
1,



1				
---	--	--	--	--

Pre-order traversal

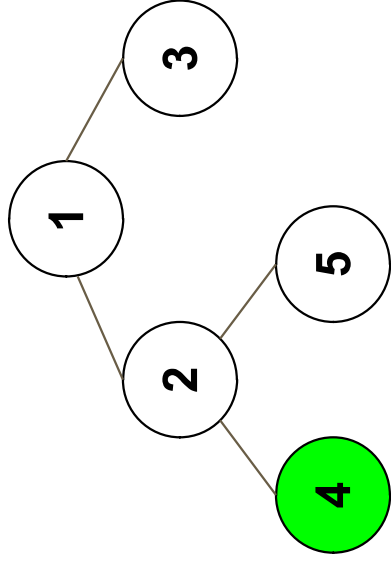
Pre-order traversal:
1,2



1	2			
---	---	--	--	--

Pre-order traversal

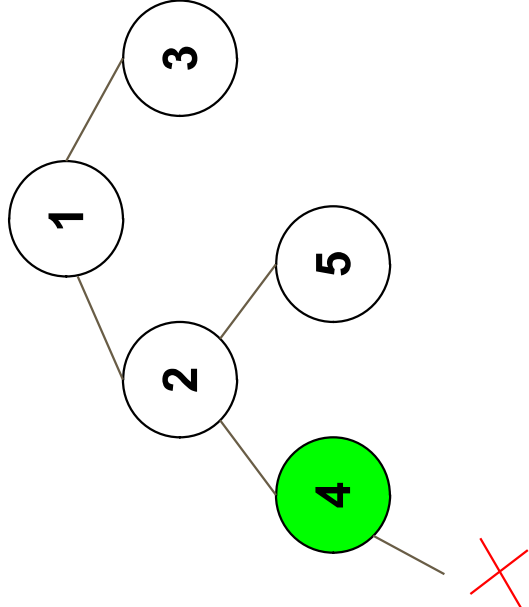
Pre-order traversal:
1,2,4



1	2	4	
---	---	---	--

Pre-order traversal

Pre-order traversal:
1,2,4

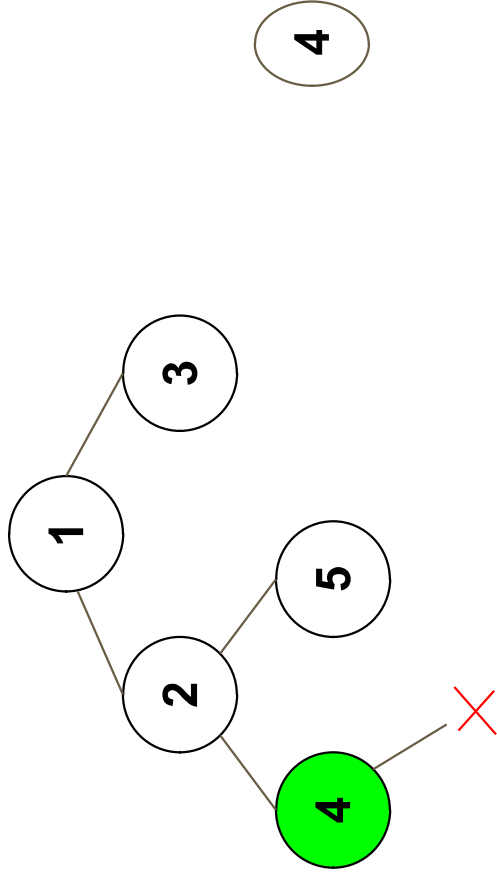


1	2	4	
---	---	---	--



Pre-order traversal

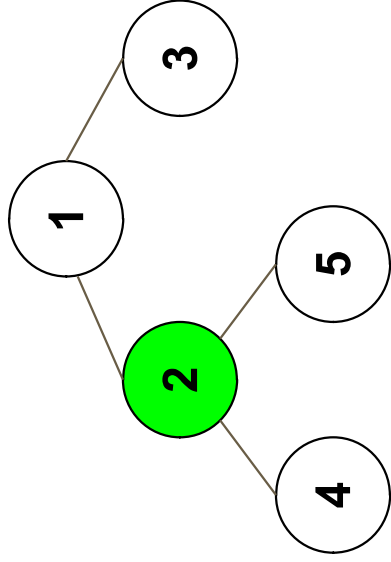
Pre-order traversal:
1,2,4



1	2			
---	---	--	--	--

Pre-order traversal

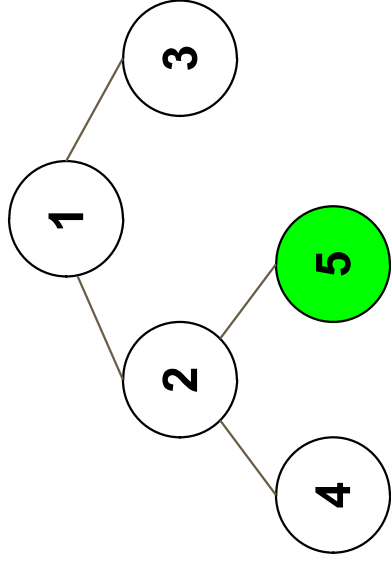
Pre-order traversal:
1,2,4



1	2			
---	---	--	--	--

Pre-order traversal

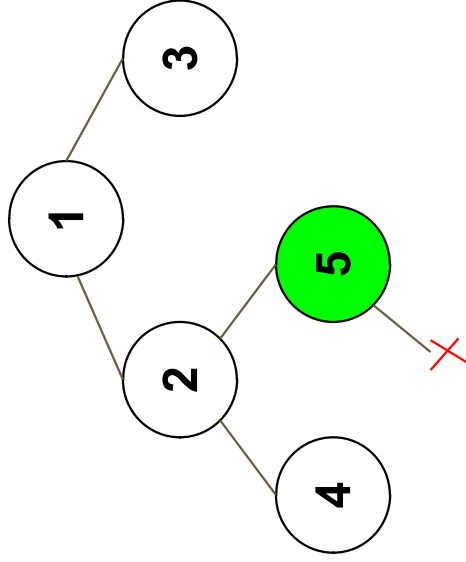
Pre-order traversal:
1,2,4,5



1	2			
---	---	--	--	--

Pre-order traversal

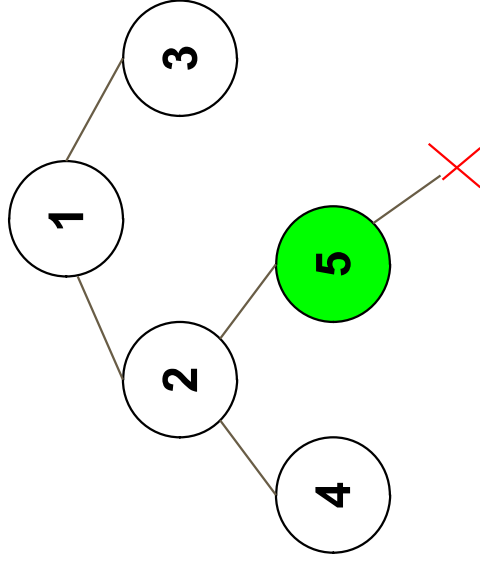
Pre-order traversal:
1,2,4,5



1	2				
---	---	--	--	--	--

Pre-order traversal

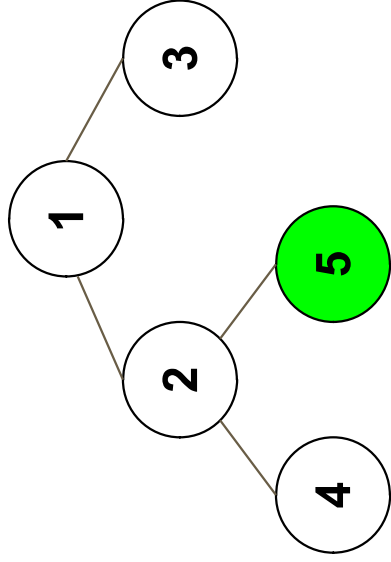
Pre-order traversal:
1,2,4,5



1	2				
---	---	--	--	--	--

Pre-order traversal

Pre-order traversal:
1,2,4,5



1	2			
---	---	--	--	--