Fibonacci Series :

1,1,2,3,5,7,12 ..

Recursion

Remember:

In recursion, we assume that the function we call does what it needs to.

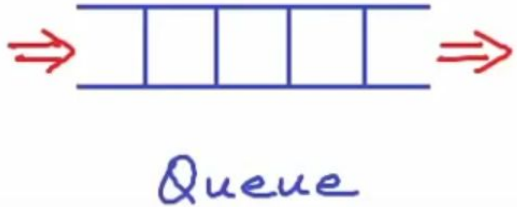
What does that mean?

Say that you want to compute the 4th Fibonacci number.

We need the 3rd and the 2nd Fibonacci numbers for that.

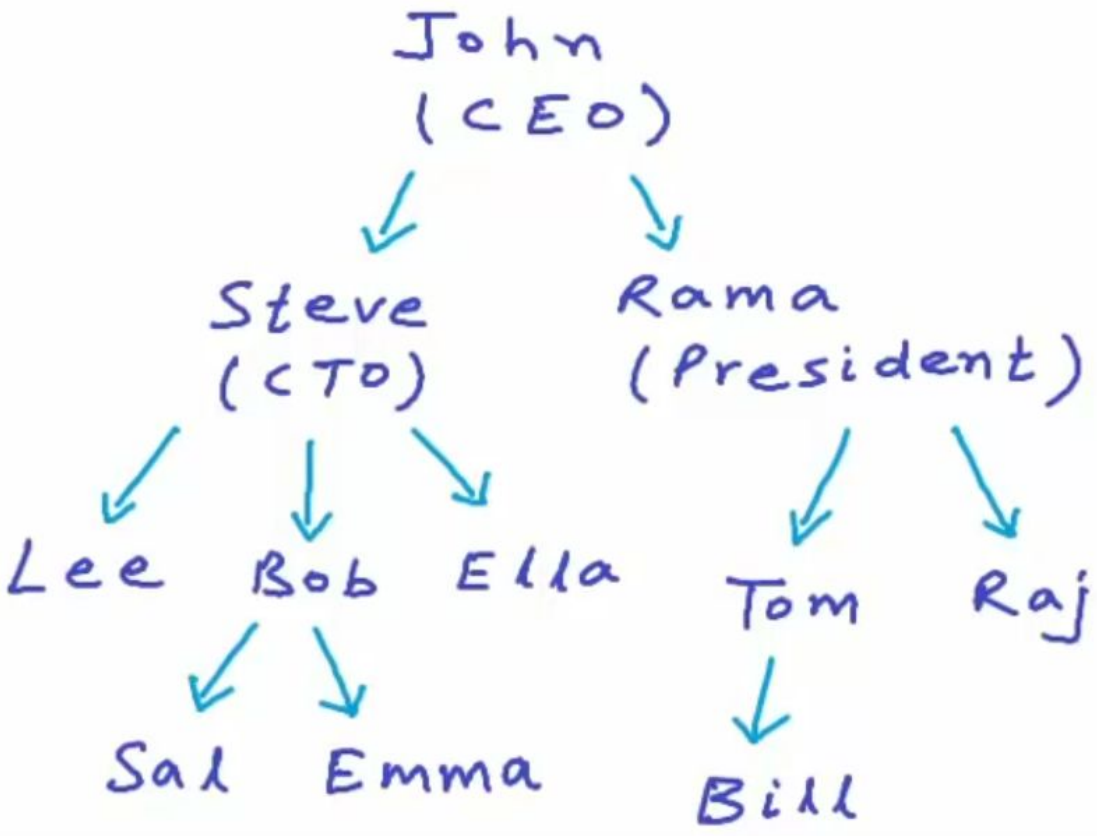
Introduction to Trees

Linear data structures:

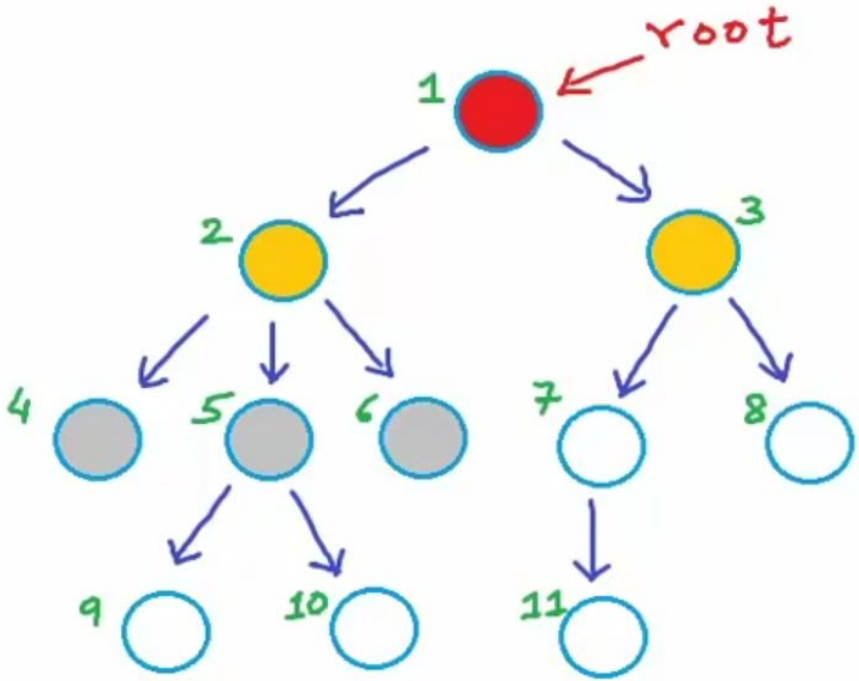


TREES

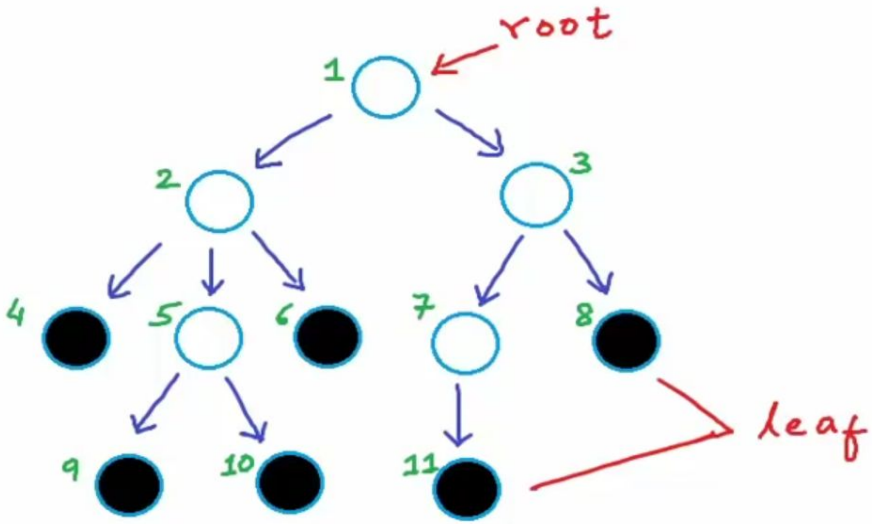
When can we use a tree?



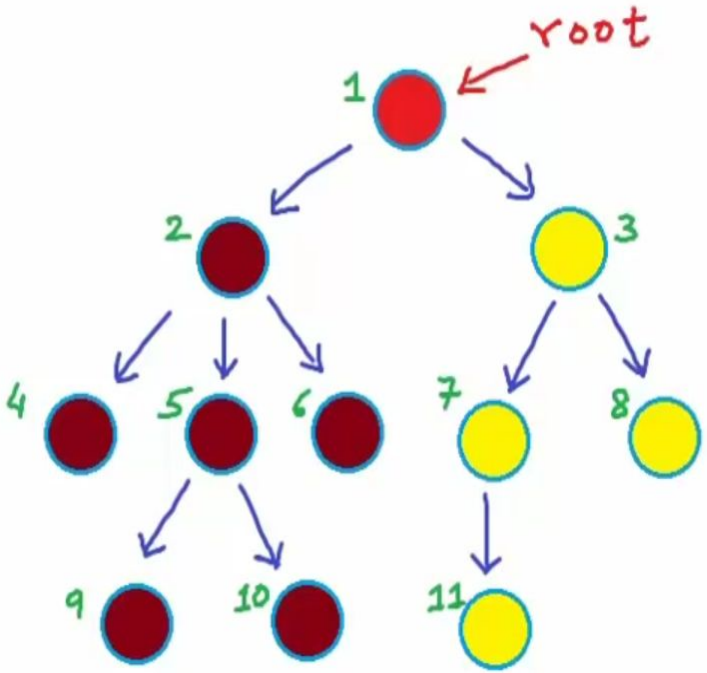
Introduction to Trees



root
children
Parent

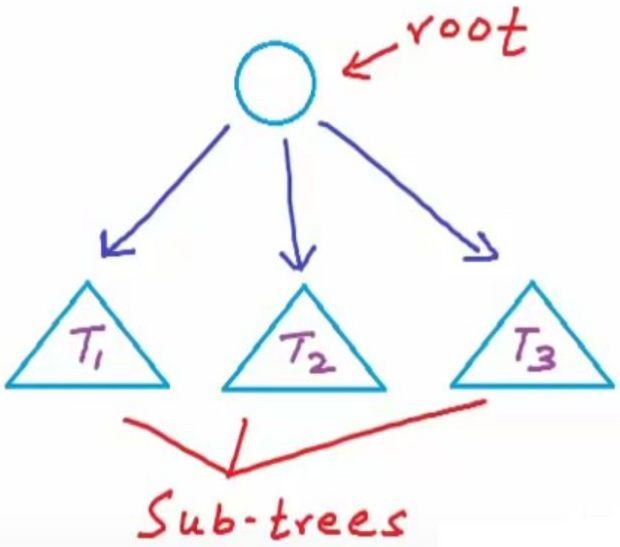


Introduction to Trees

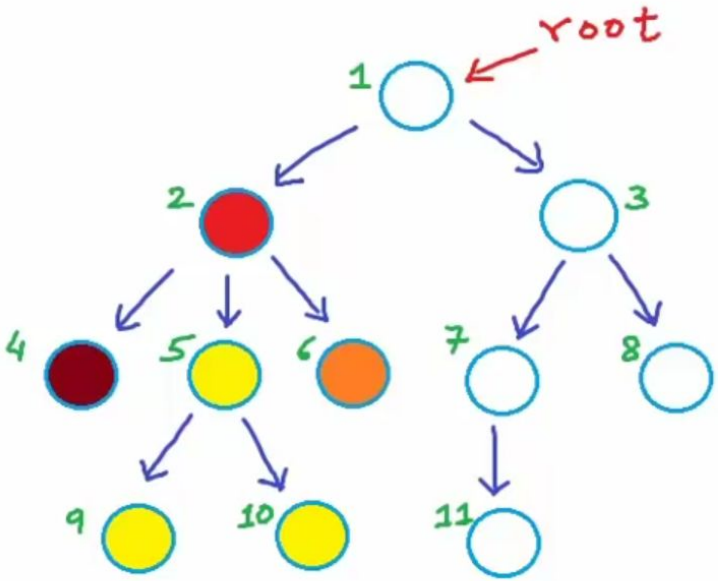


Tree

↳ recursive data structure

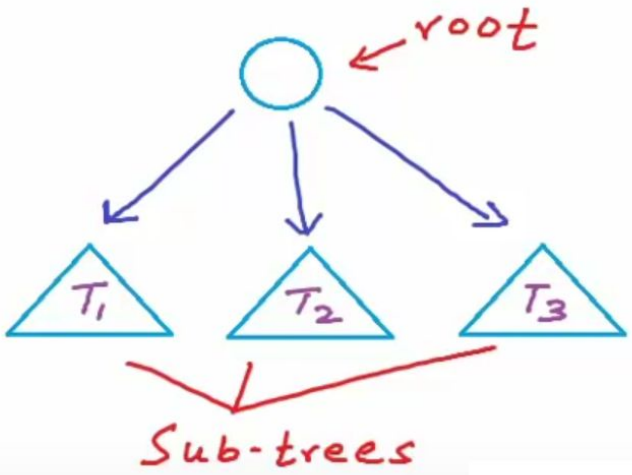


Introduction to Trees

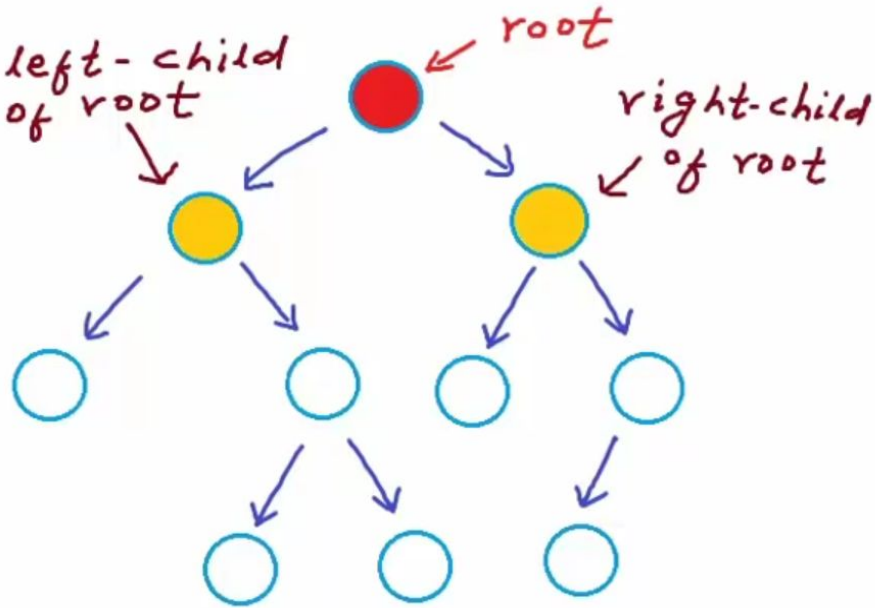


Tree

↳ recursive data structure

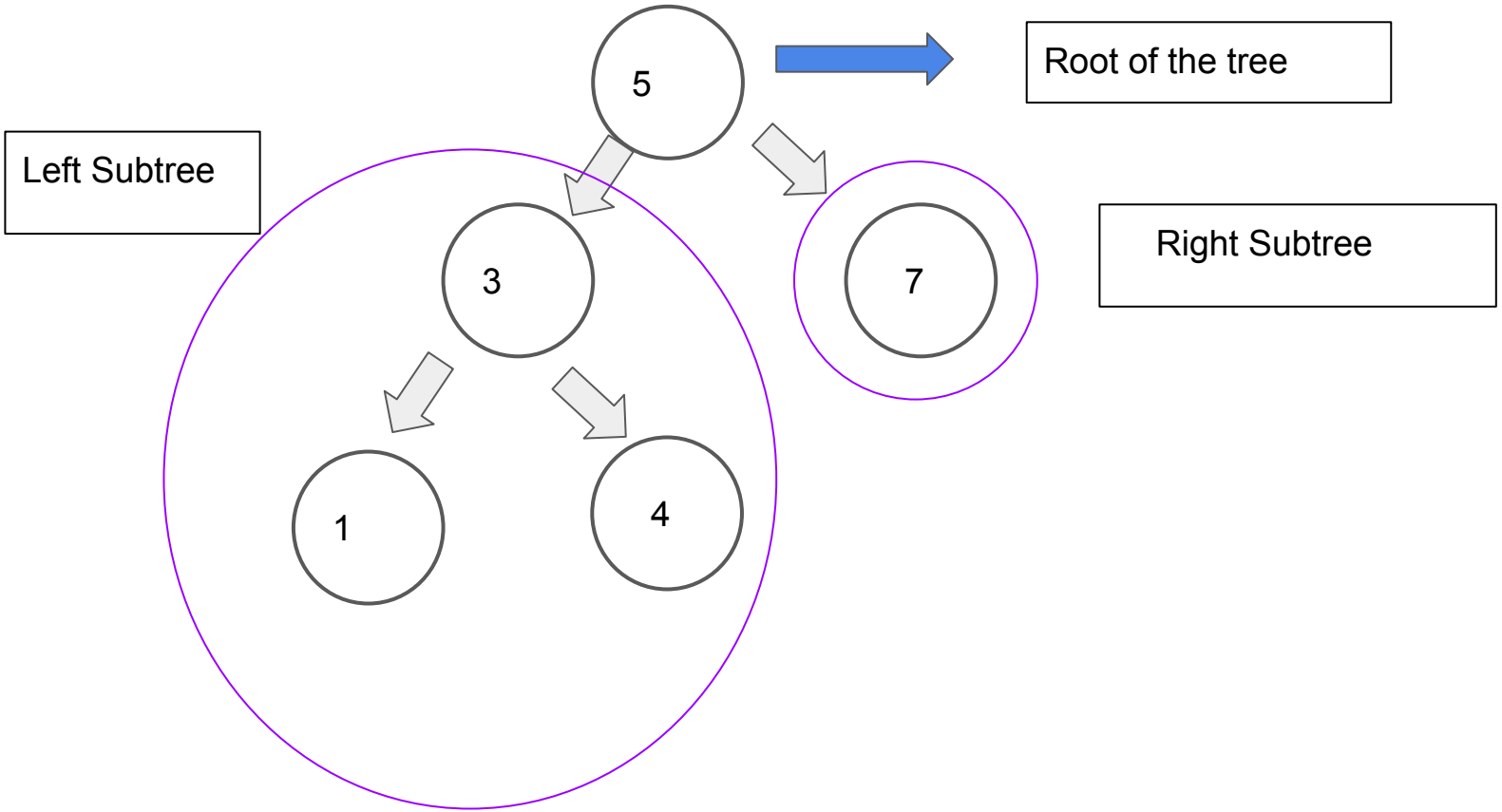


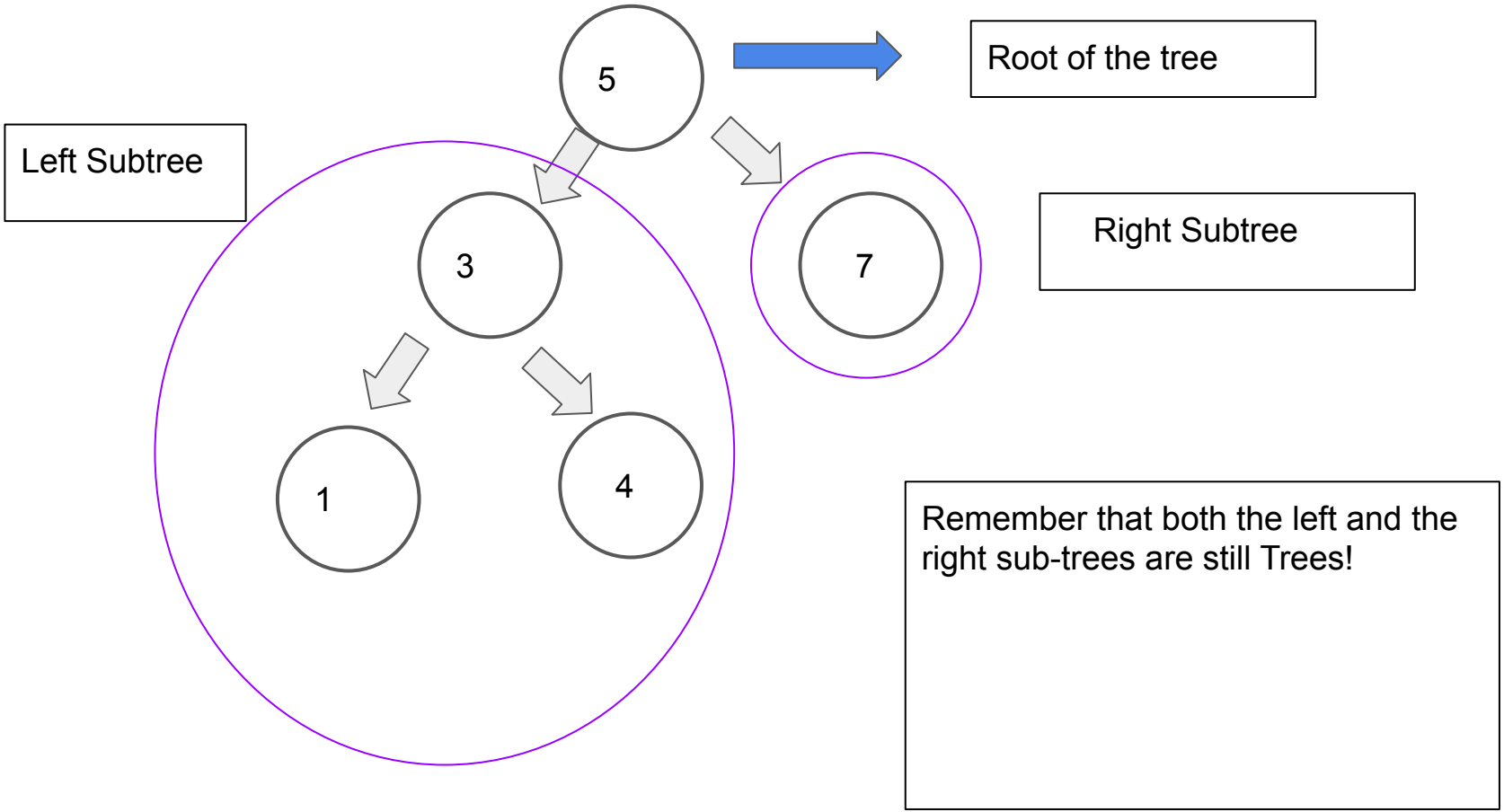
Binary Tree

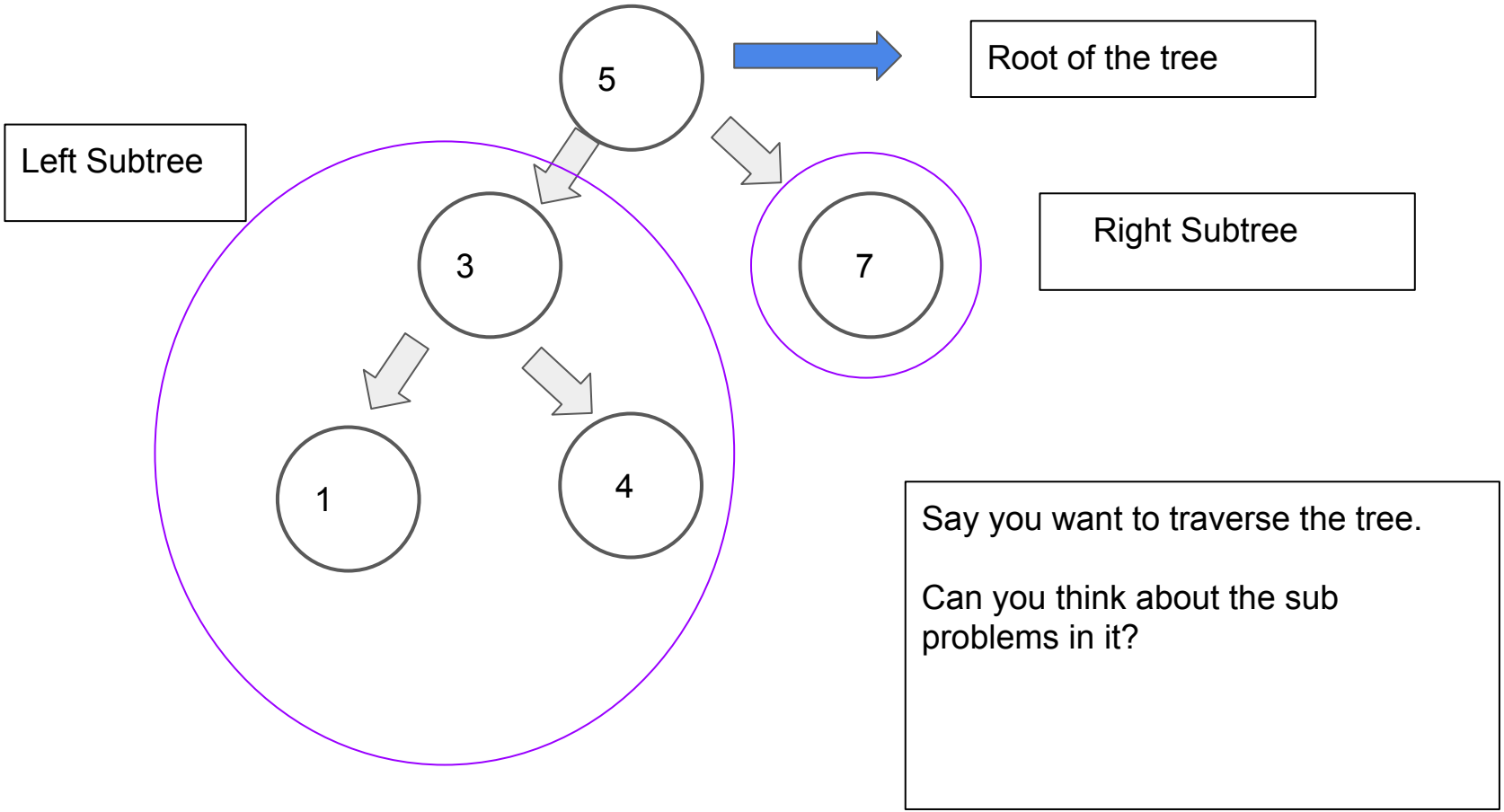


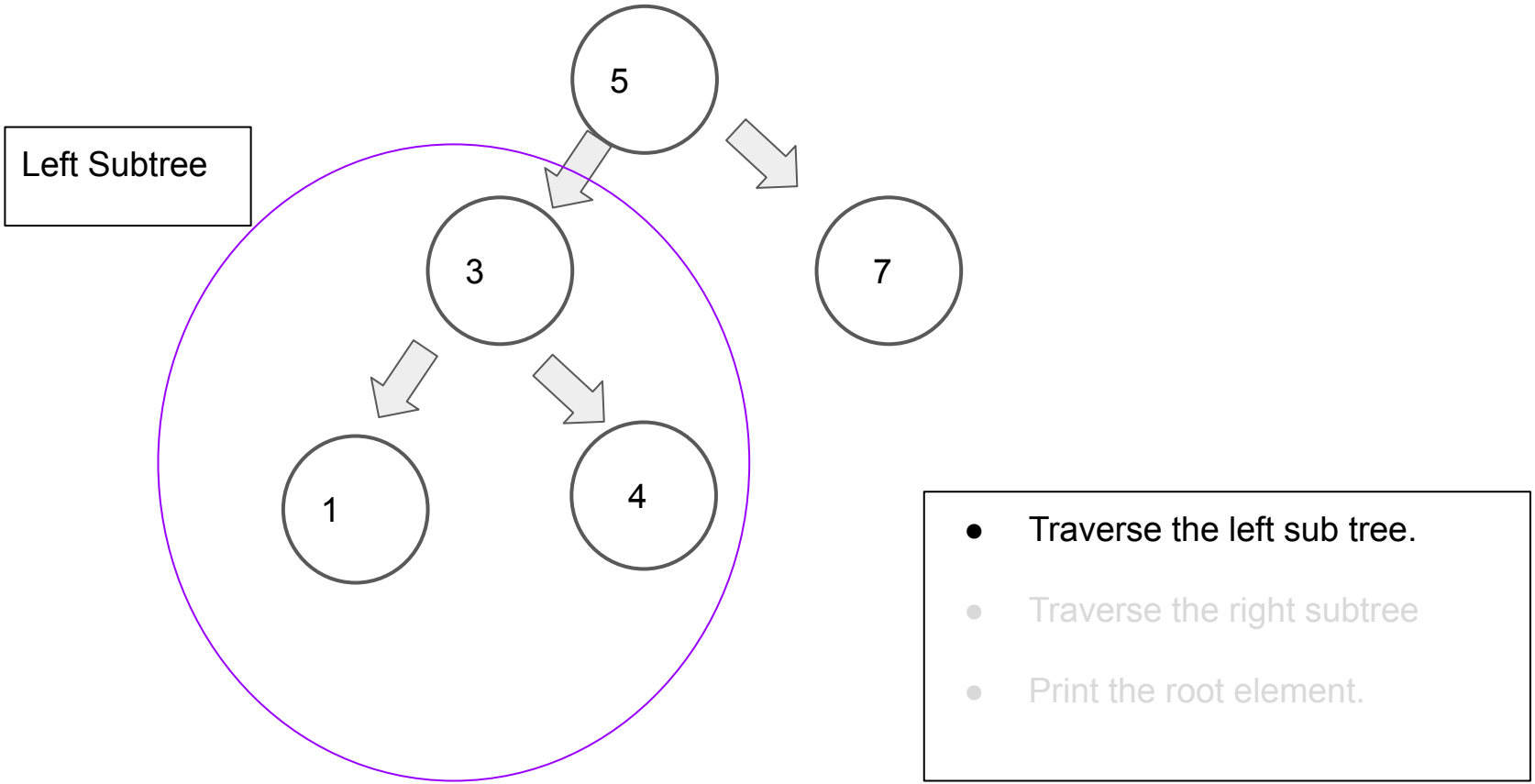
Binary tree

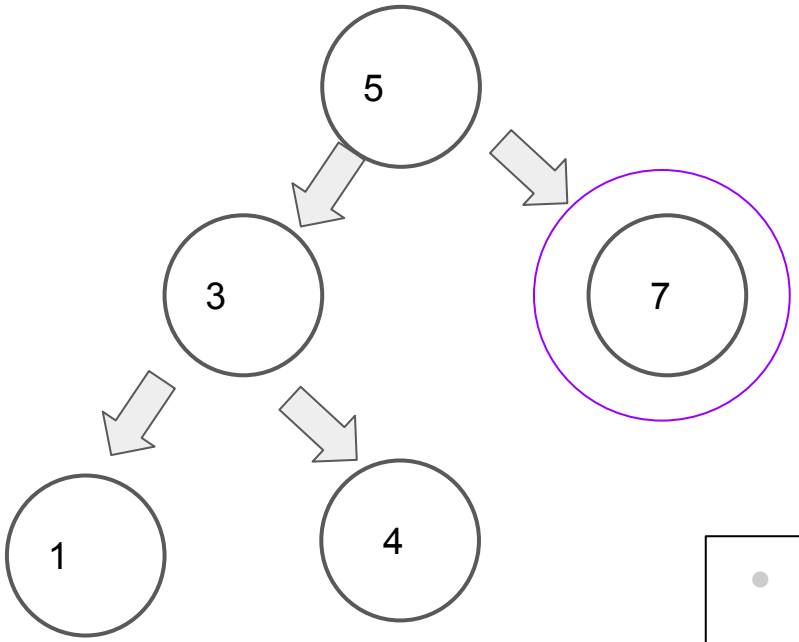
↳ each node can have at most 2 children


5
Root of the tree
Left Subtree
3
7
Right Subtree
1
4


5
Root of the tree
Left Subtree
3
7
Right Subtree
1
4
Remember that both the left and the right sub-trees are still Trees!

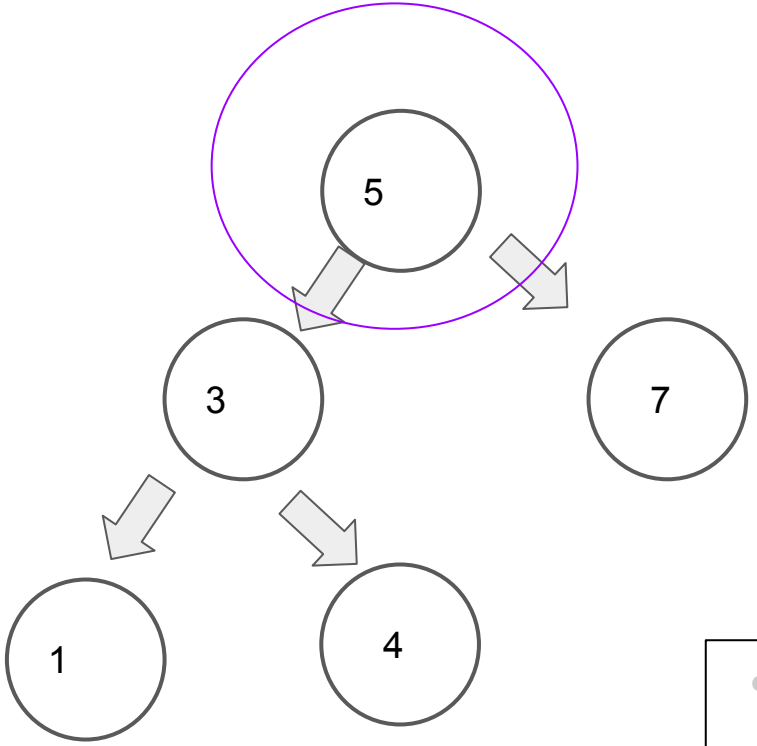

5
Root of the tree
Left Subtree
3
7
Right Subtree
1
4
Say you want to traverse the tree.
Can you think about the sub problems in it?


Left Subtree
5
3
7
1
4
Traverse the left sub tree.
Traverse the right subtree
Print the root element.



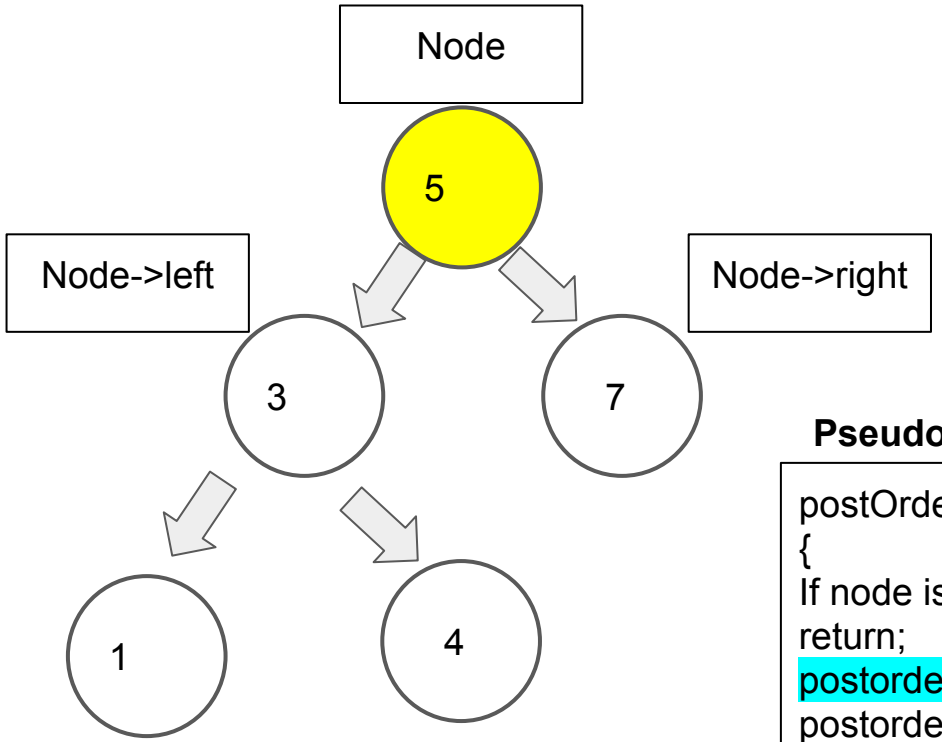
Right Subtree

- Traverse the left sub tree.
- Traverse the right subtree
- Print the root element.



Root element

- Traverse the left sub tree.
- Traverse the right subtree
- Print the root element.



Call stack

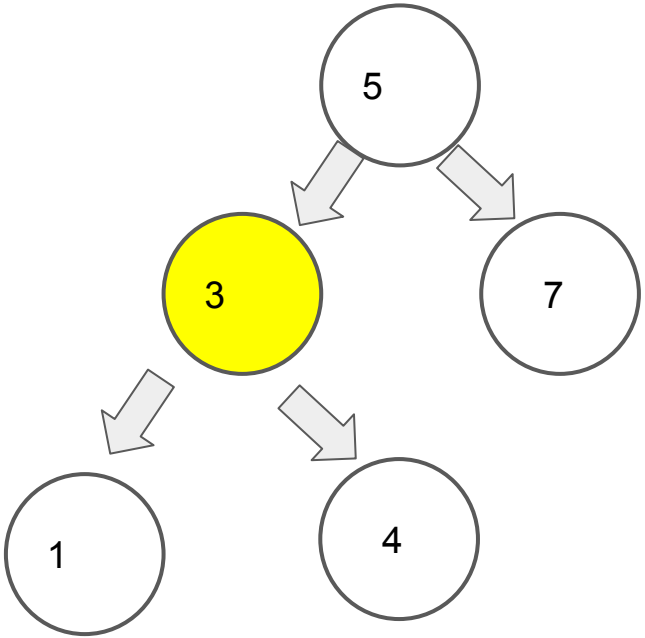
Note that we actually send the **pointer of the node** to the function

postorder(5)

Pseudo code

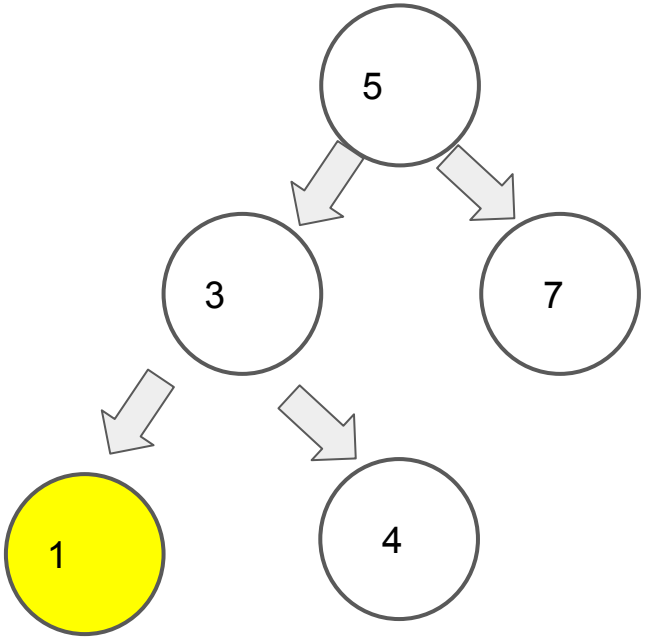
```
postOrder(Node)
{
If node is null:
return;
postorder(Node->left);
postorder(Node->right);
cout<<Node->key;

}
```

Recursively calling the same function on left and right children



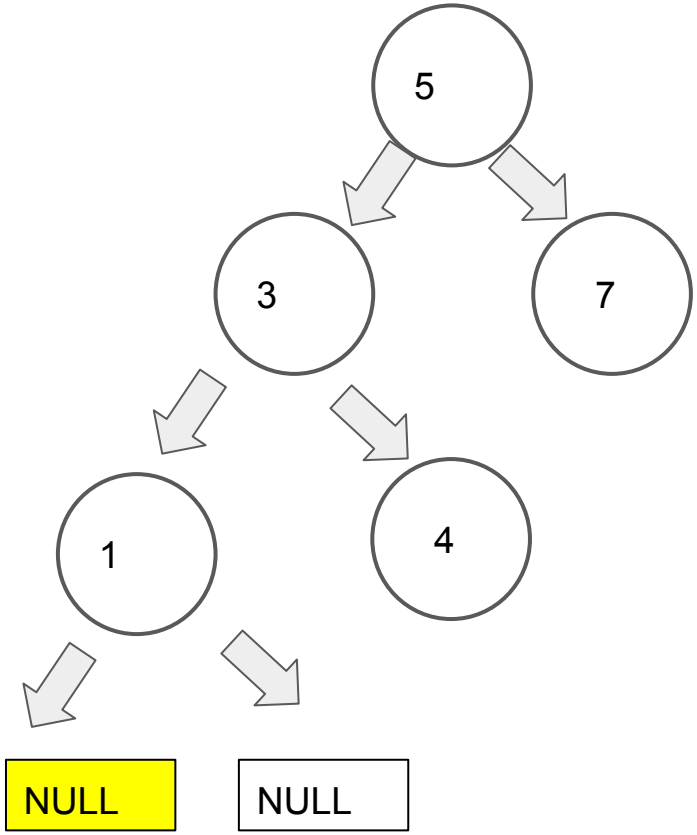
Call stack

postorder(3)
postorder(5)



Call stack

postorder(1)
postorder(3)
postorder(5)



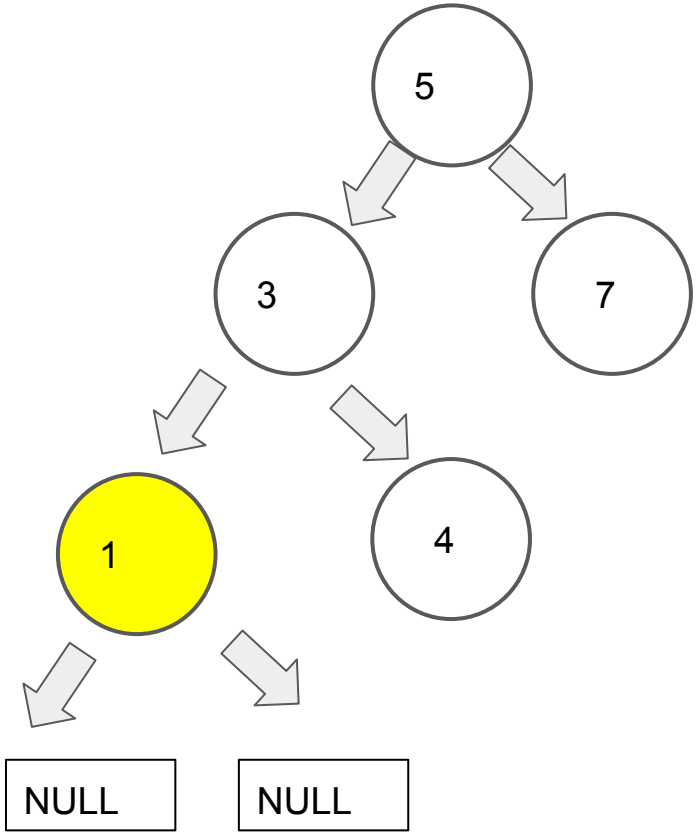
This function call returns.

Pseudo code

```
postOrder(Node)
{
If node is null:
return;
postorder(Node->left);
postorder(Node->right);
cout<<Node->key;

}
```

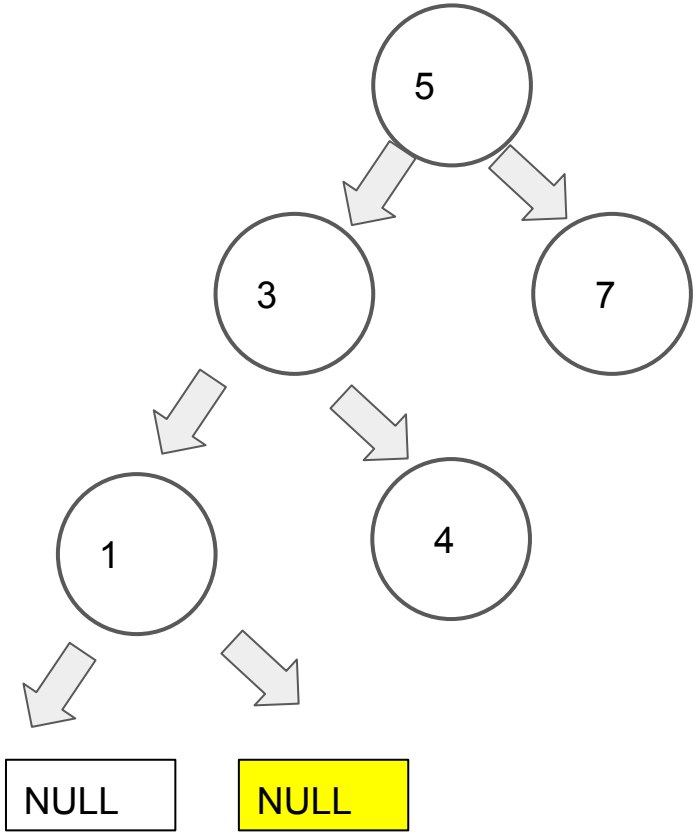
Call stack

- postorder(NULL)
- postorder(1)
- postorder(3)
- postorder(5)



Call stack

postorder(1)
postorder(3)
postorder(5)



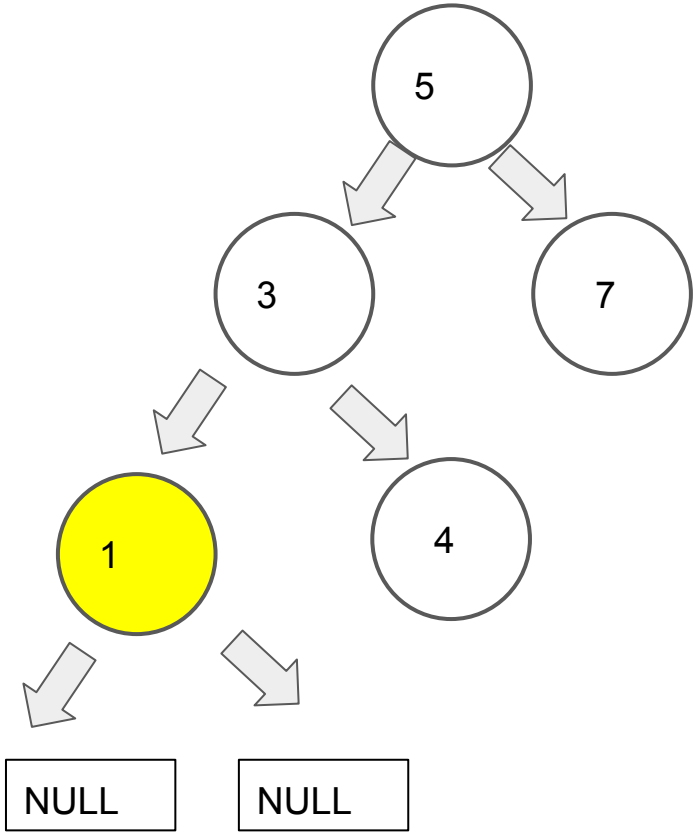
This function call returns.

Pseudo code

```
postOrder(Node)
{
If node is null:
return;
postorder(Node->left);
postorder(Node->right);
cout<<Node->key;

}
```

Call stack

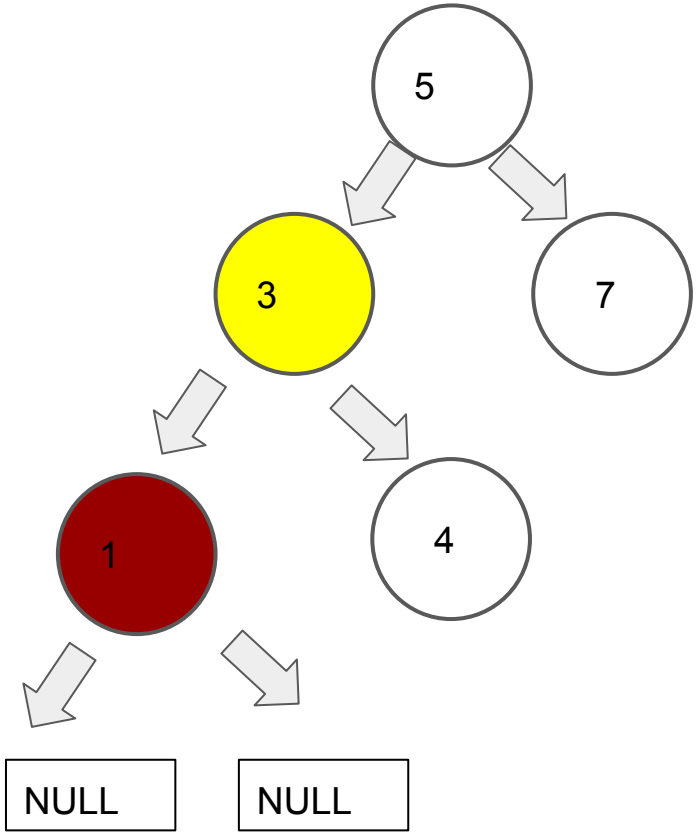
- postorder(NULL)
- postorder(1)
- postorder(3)
- postorder(5)



OUTPUT:
1

Call stack

postorder(1)
postorder(3)
postorder(5)

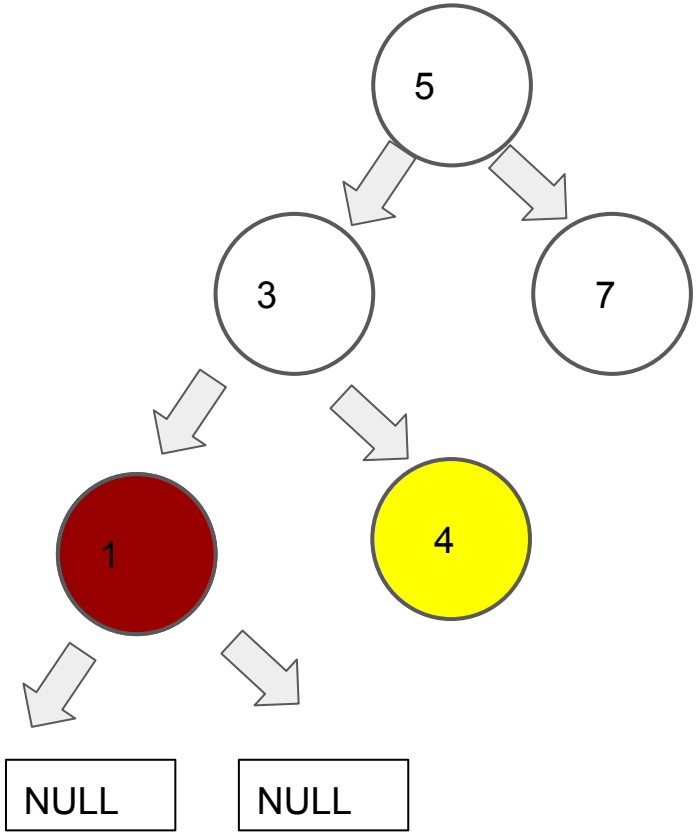


Call stack

OUTPUT:
1

postorder(3)

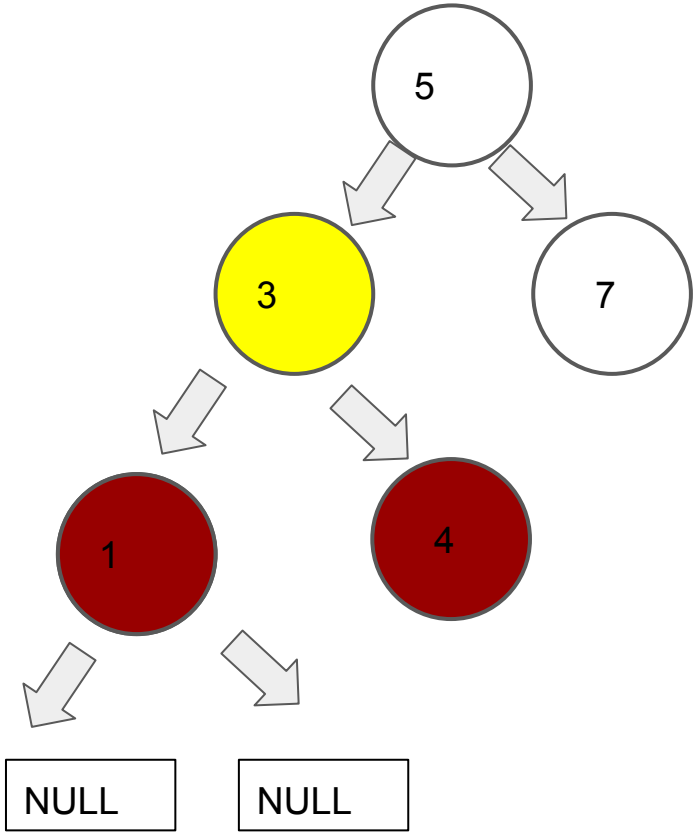
postorder(5)



OUTPUT:
1
4

Call stack

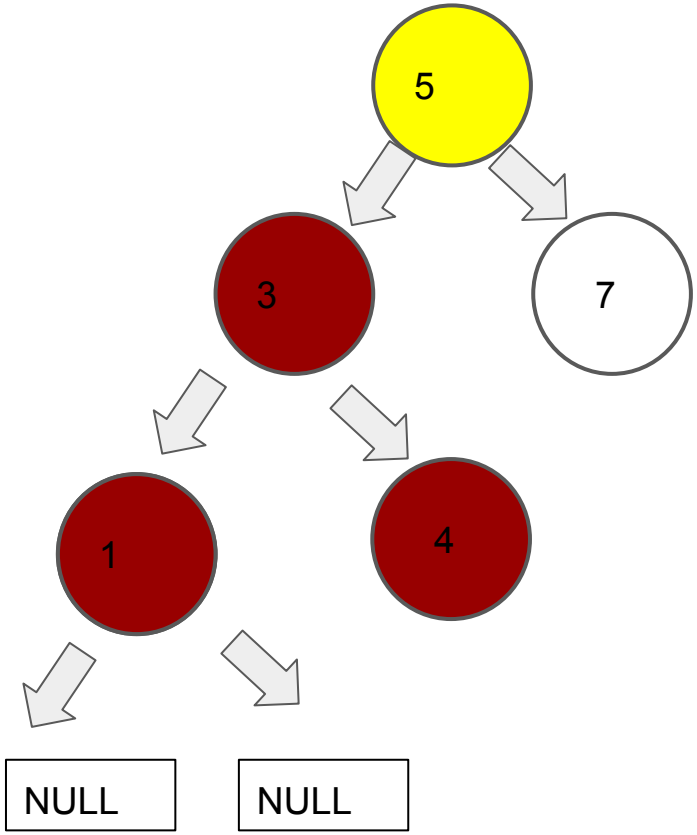
postorder(4)
postorder(3)
postorder(5)



OUTPUT:
1
4
3

Call stack

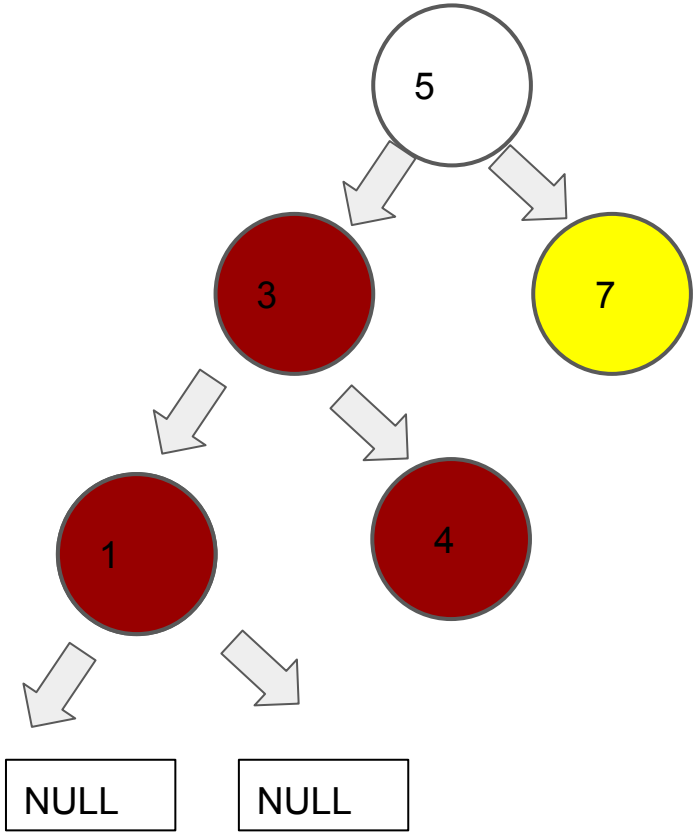
postorder(3)
postorder(5)



Call stack

OUTPUT:
1
4
3

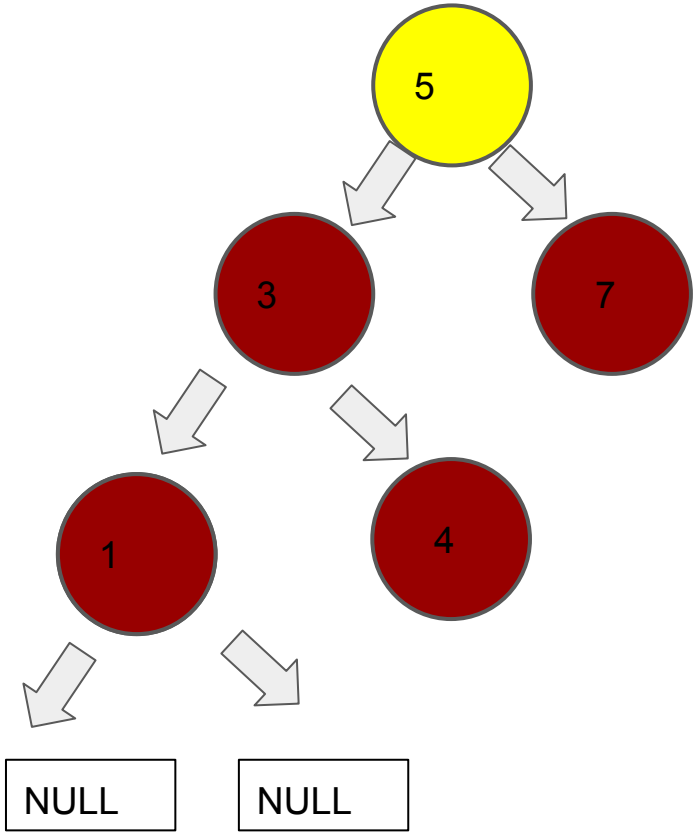
postorder(5)



Call stack

OUTPUT:
1
4
3

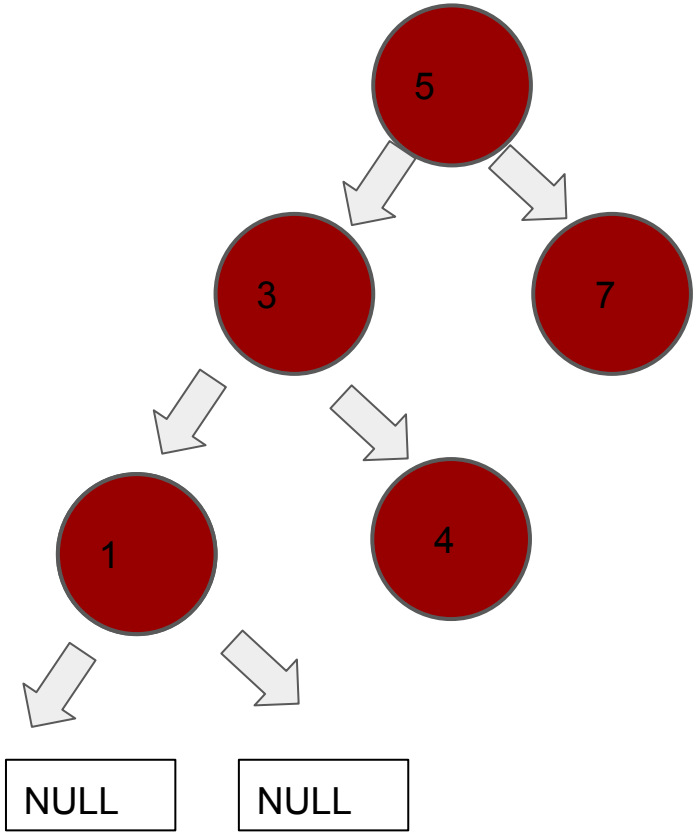
postorder(7)
postorder(5)



Call stack

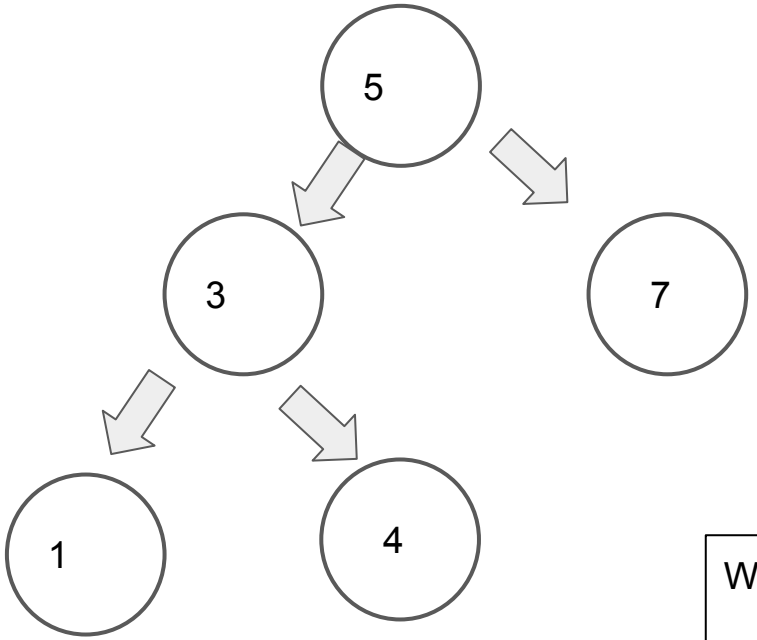
OUTPUT:
1
4
3
7

postorder(5)

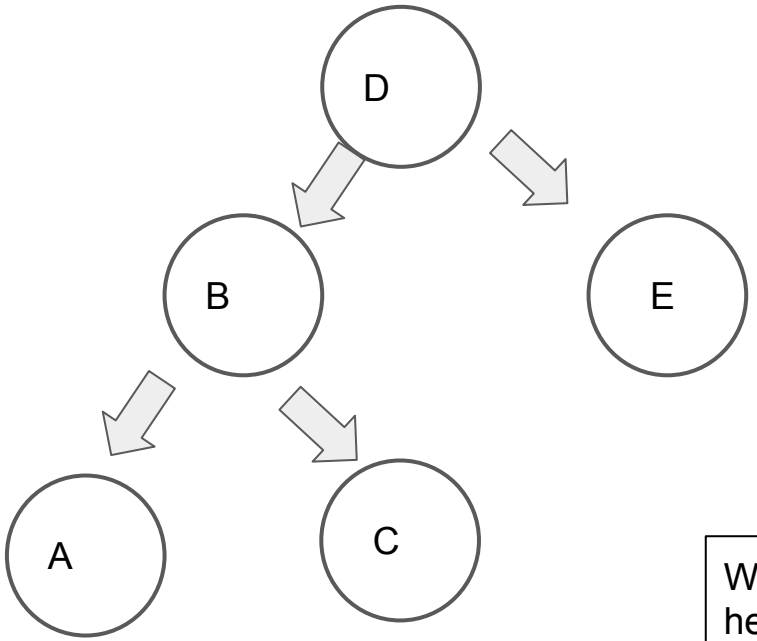


Call stack

OUTPUT:
1
4
3
7
5



What if I do an **inorder** traversal?



What if I do an **inorder** traversal here?

This will help with your printMovieInventory function in the assignment

Problems in the assignment/recitation:

- Delete a node in a Binary Search Tree(Be careful with the order of operations).
- Search in a Binary Search Tree
- Add in a Binary Search Tree
- Size of a Binary Search Tree
- Average of the keys in a Binary Search Tree