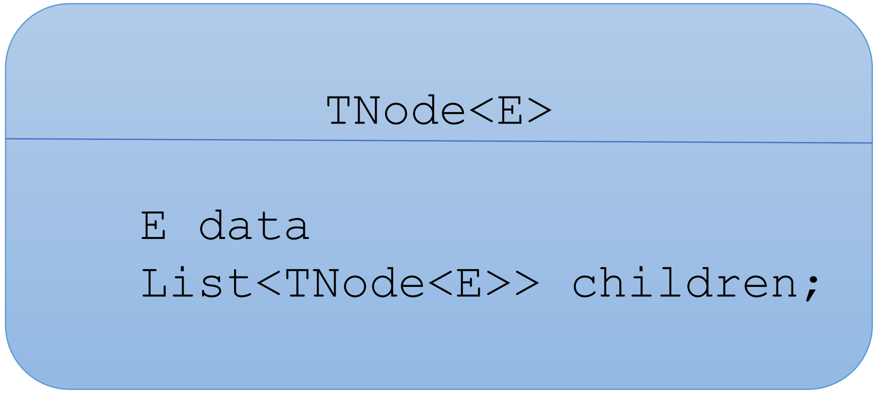
# Binary Tree

A tree data structure where each node is restricted to 0, 1, or 2 children.

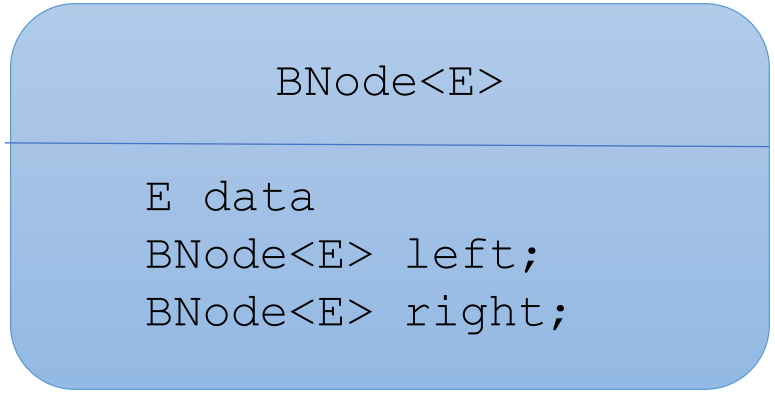
Regular Tree Node



```
TNode<E>
```

```
E data
List<TNode<E>> children;
```

Binary Tree Node



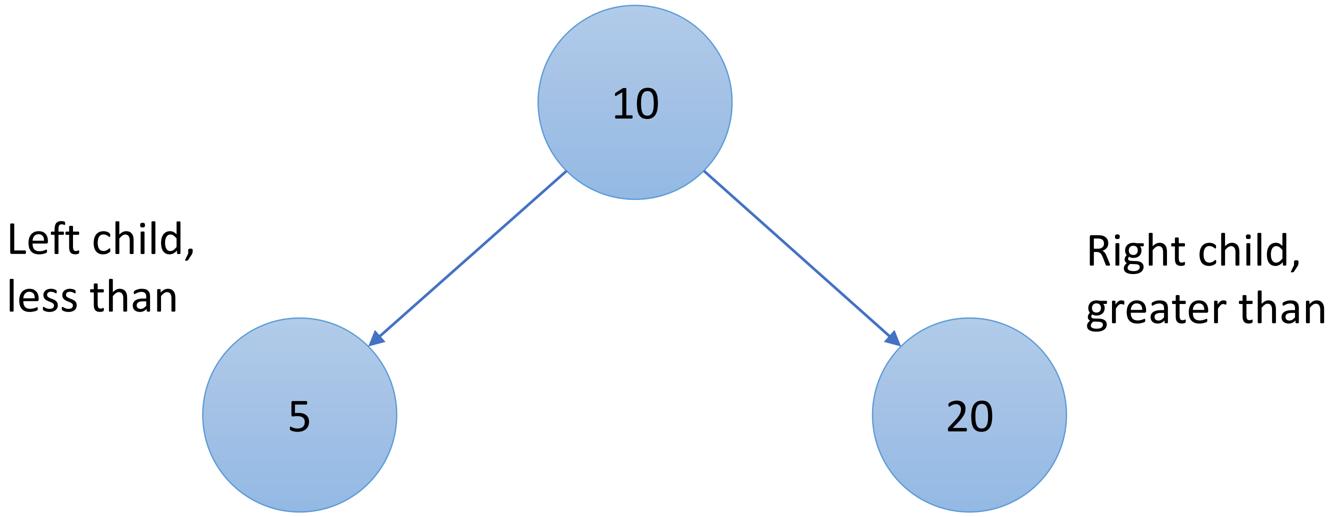
```
BNode<E>
```

```
E data
BNode<E> left;
BNode<E> right;
```

# Binary **Search** Tree

A Binary Tree with the following rules:

- The value of the left child is less than the value of *this* node
- The value of the right child is greater than the value of *this* node



Left child, less than

Right child, greater than

O(log n)

BSTs are used to quickly find a value in the tree

# Populating a BST

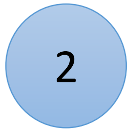
Add the following list of number to a Binary Search Tree

```
[2, 7, 9, 3, 8, 1, 6]
```

# Populating a BST

Add the following list of number to a Binary Search Tree

[~~2~~, 7, 9, 3, 8, 1, 6]

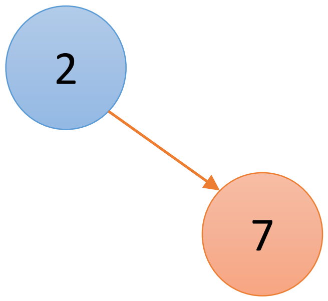


The **first value** added to a BST becomes the **root**

# Populating a BST

Add the following list of number to a Binary Search Tree

[~~2~~, ~~7~~, 9, 3, 8, 1, 6]

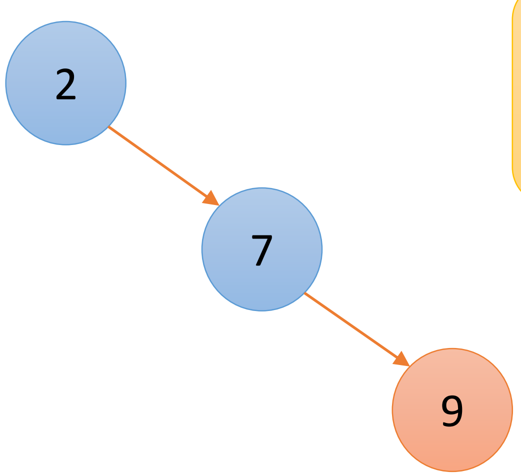


7 is **greater than** 2, so it is added as the right child

# Populating a BST

Add the following list of number to a Binary Search Tree

`[~~2~~, ~~7~~, ~~9~~, 3, 8, 1, 6]`



9 is greater than 2, so you look to the right node

9 is **greater than** 7, so add it as the **right** child

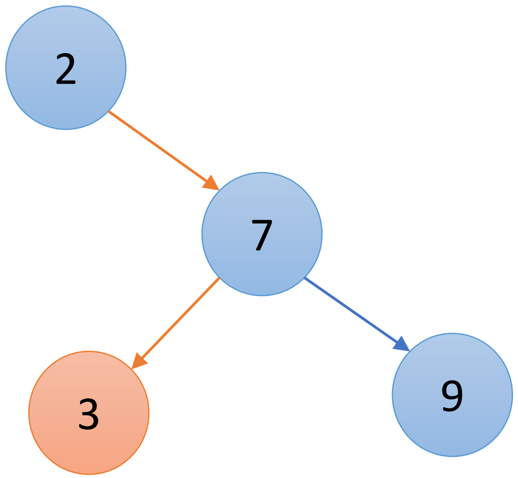
# Populating a BST

Add the following list of number to a Binary Search Tree

[~~2~~, ~~7~~, ~~9~~, ~~3~~, 8, 1, 6]

3 is greater than 2, so you look to the right node

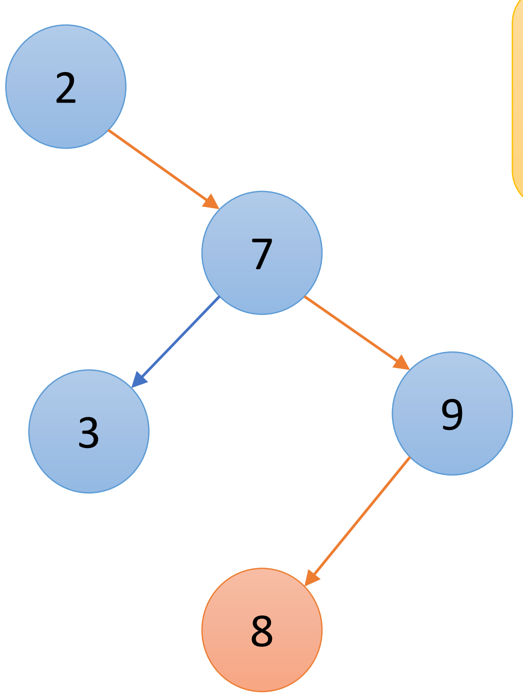
3 is **less than** 7, so add it as the **left** child



# Populating a BST

Add the following list of number to a Binary Search Tree

[~~2~~, ~~7~~, ~~9~~, ~~3~~, ~~8~~, 1, 6]



8 is greater than 2, so you look to the right node

8 is greater than 7, so you look to the right node

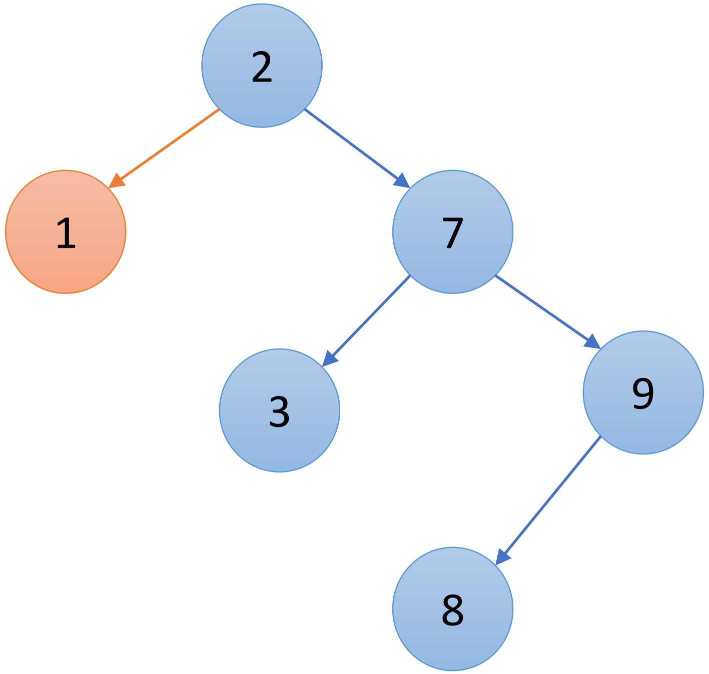
8 is **less than** 9, so it as the **left** child

# Populating a BST

Add the following list of number to a Binary Search Tree

`[`~~`2`~~`, `~~`7`~~`, `~~`9`~~`, `~~`3`~~`, `~~`8`~~`, `~~`1`~~`, 6]`

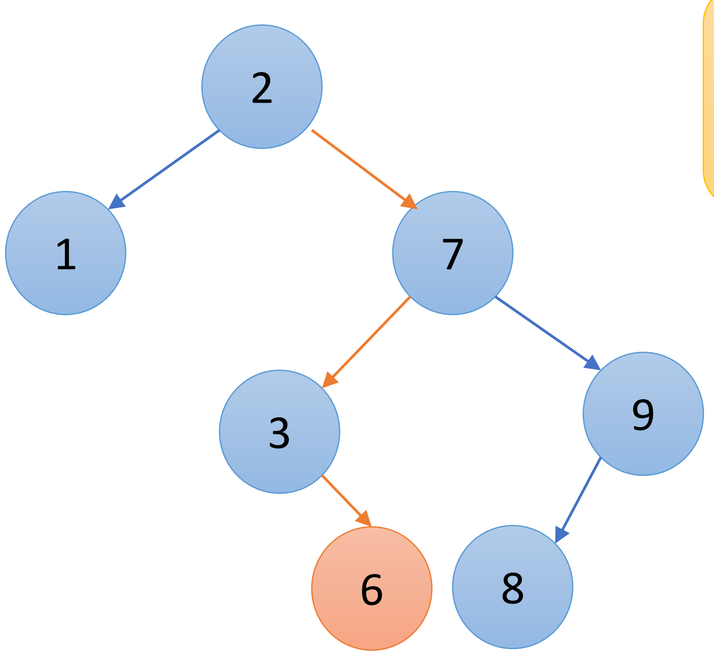
1 is **less than** 2, so add it as the **left** child



# Populating a BST

Add the following list of number to a Binary Search Tree

[~~2~~, ~~7~~, ~~9~~, ~~3~~, ~~8~~, ~~1~~, ~~6~~]



6 is greater than 2, so you look to the right node.

6 is less than 7, so you look to the left node

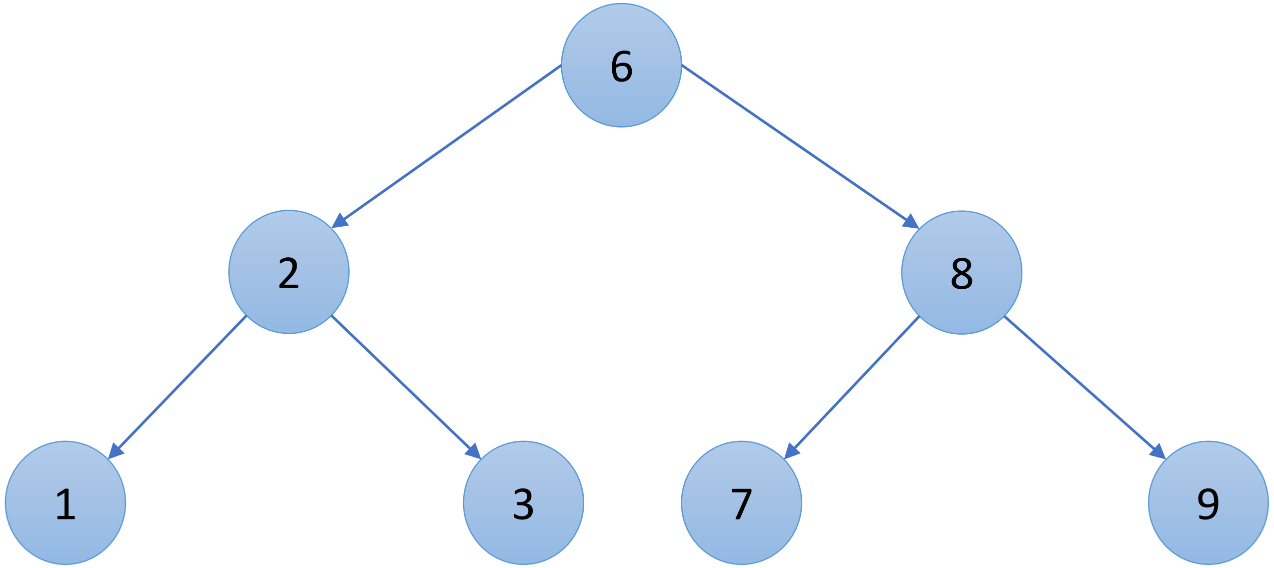
6 is **greater than** 3, so add it as the **right** child

# Activity: Populate a BST with the following numbers

```
[6, 2, 8, 3, 1, 9, 7]
```

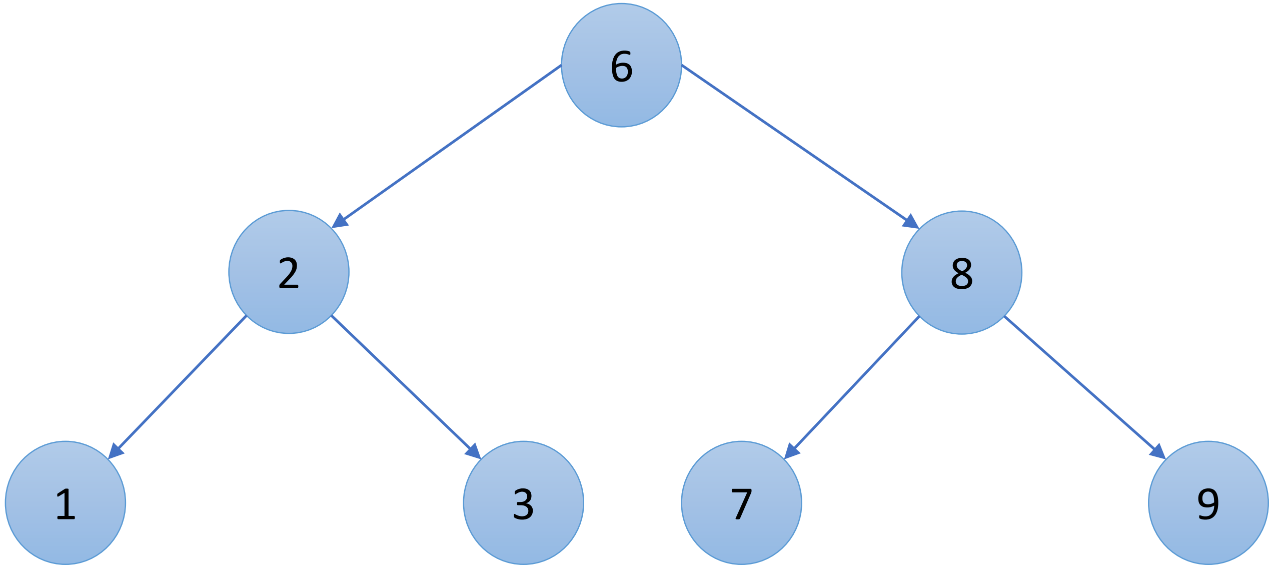
# Activity: Populate a BST with the following numbers

`[6, 2, 8, 3, 1, 9, 7]`

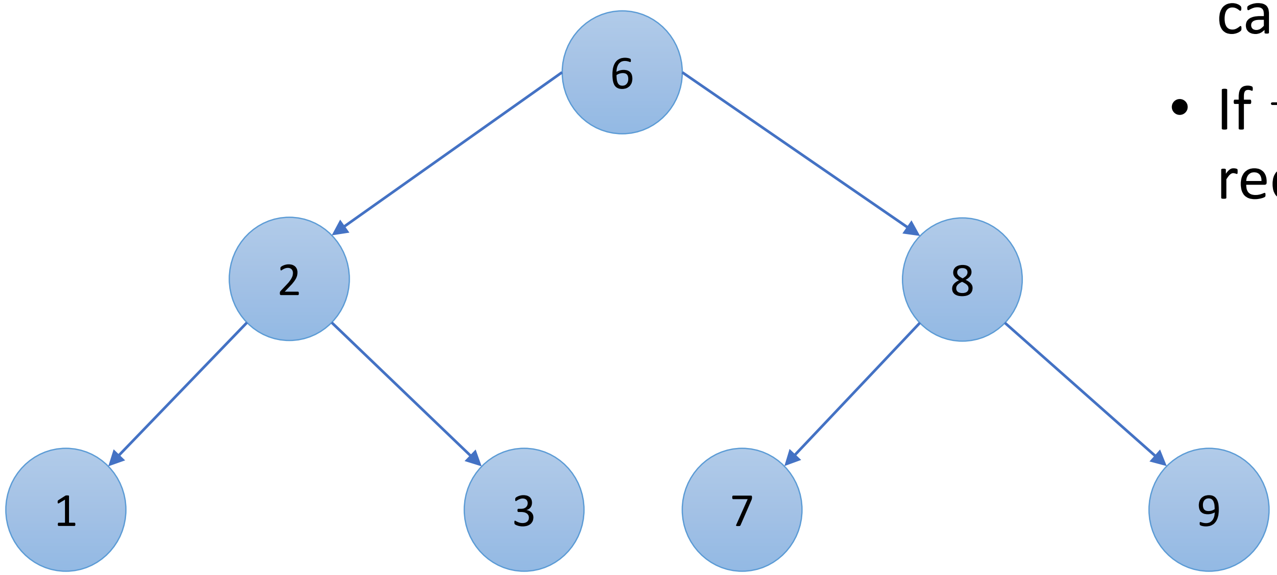


Same numbers;
Different tree!

# What is the algorithm to “search” for a value (`target`) in a BST?



# What is the algorithm to “search” for a value (`target`) in a BST?

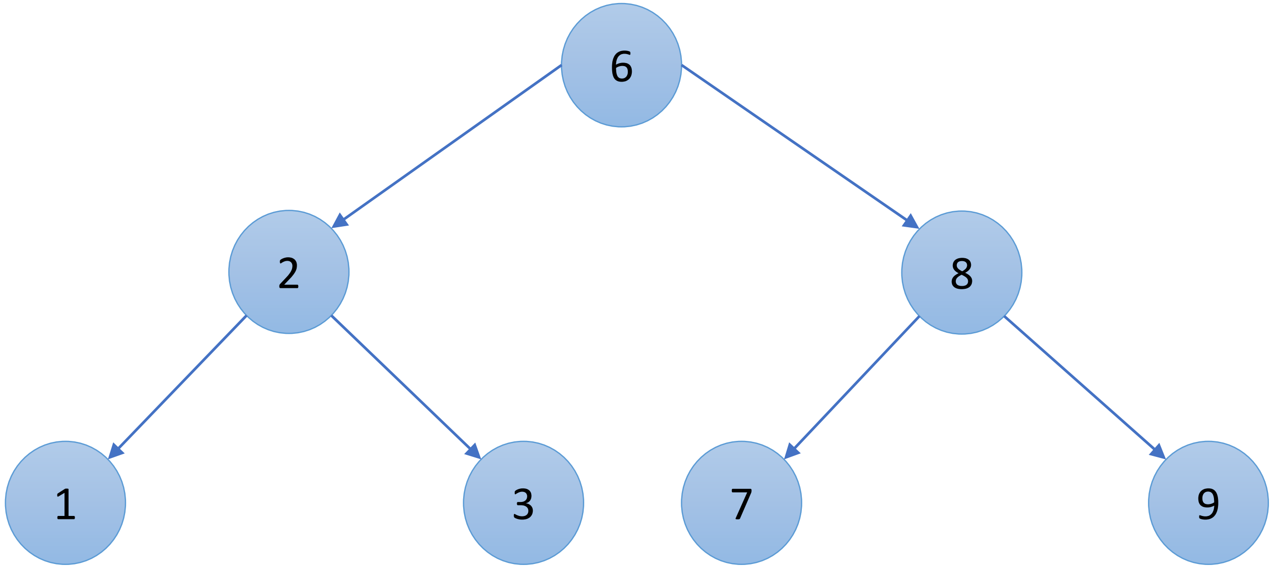


- Look at the root, is it `target`?
- If `target` is less than the root, reclusively call `search` on the left child
- If `target` is greater than the root, recursively call `search` on the right child

BST `search` returns `true` or `false`, if `target` is in the tree.

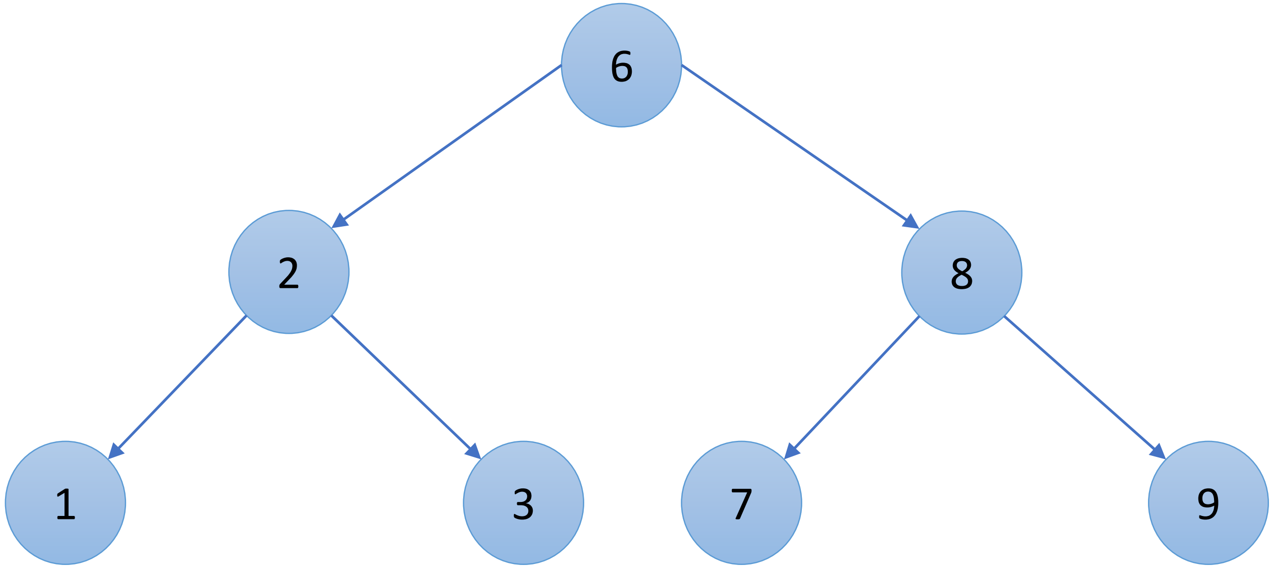
How many “checks” will it take to find the value 7 in this BST?

`[6, 2, 8, 3, 1, 9, 7]`



# What is the Big-O of the Search method on a BST?

```
[6, 2, 8, 3, 1, 9, 7]
```



# Printing the contents of a Binary Tree

There are three ways to print the contents of a Binary Tree:

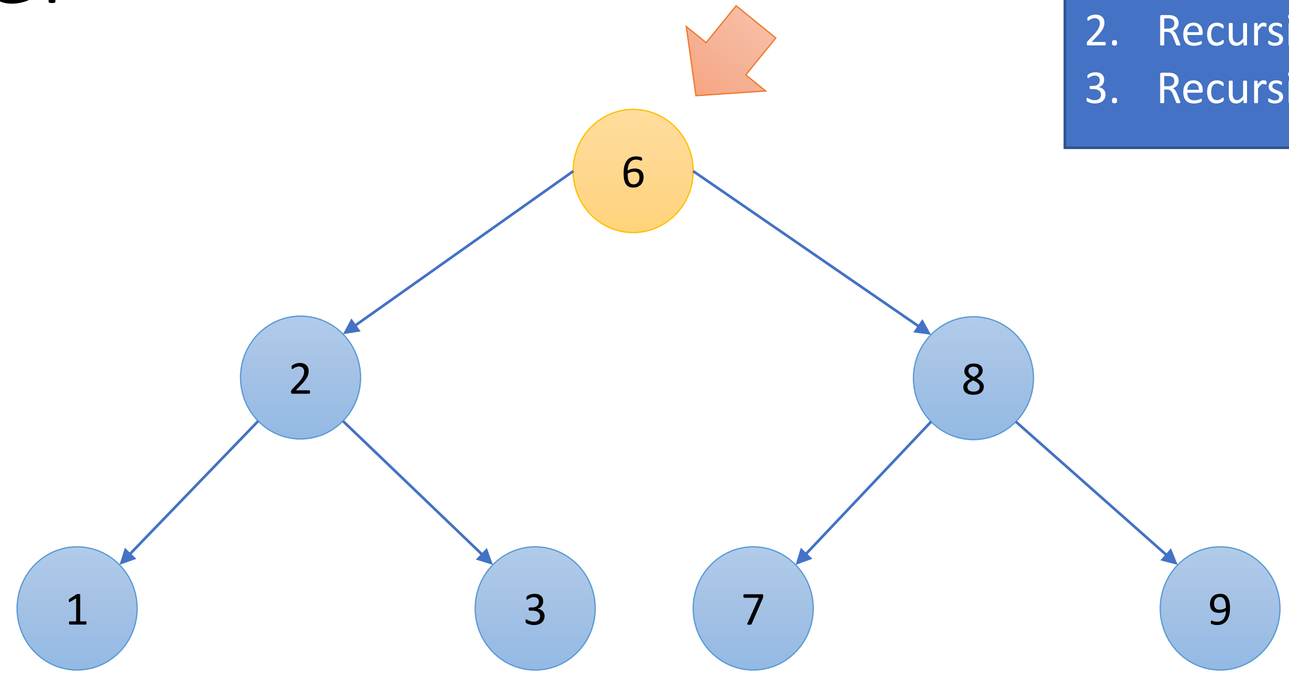
- Pre Order
- Post Order
- In Order

Same as with a regular tree

All of these names (pre, post, in) are in reference to the root node of a subtree

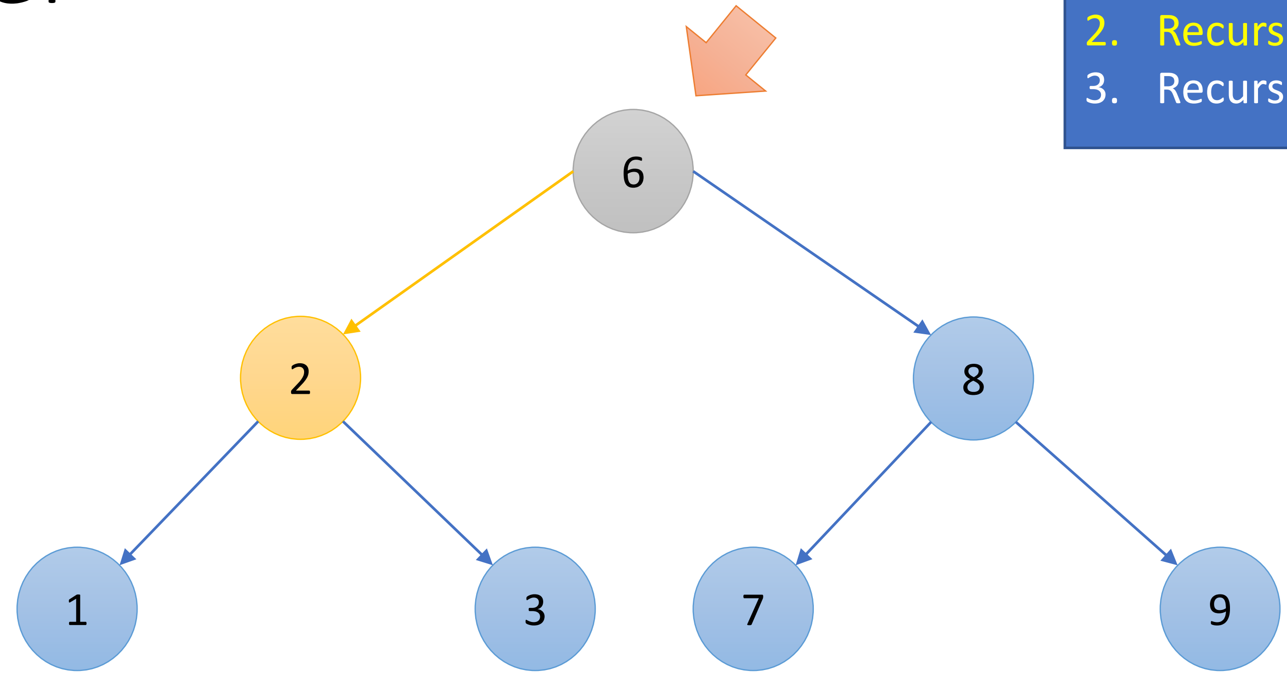
# Pre Order

1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child



6

# Pre Order

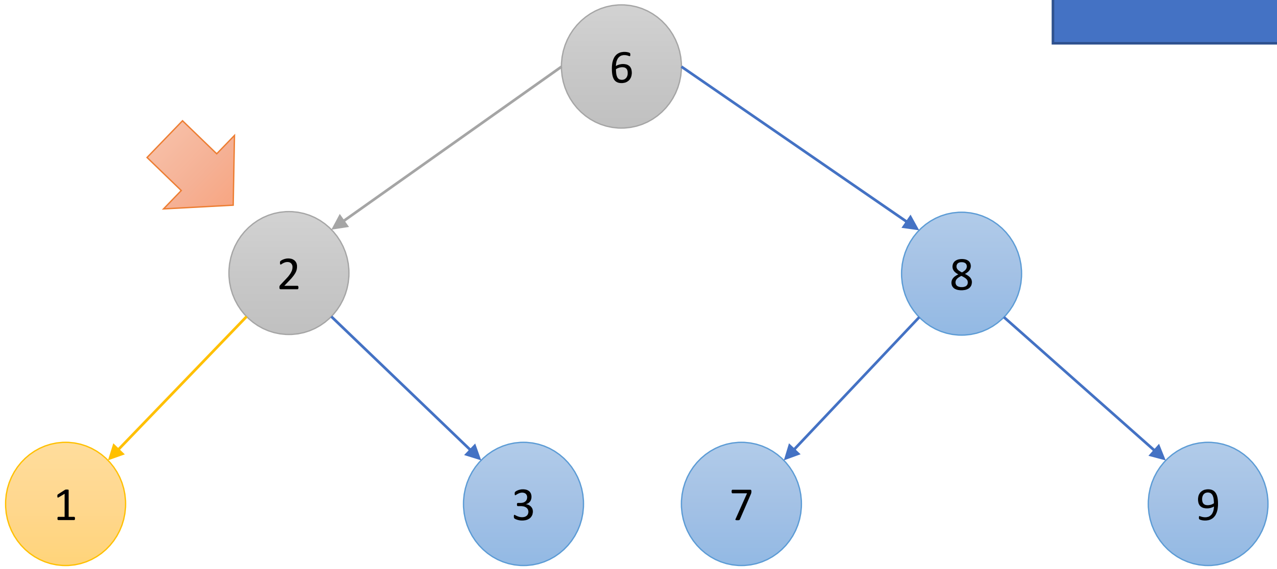


1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child

6 2

# Pre Order

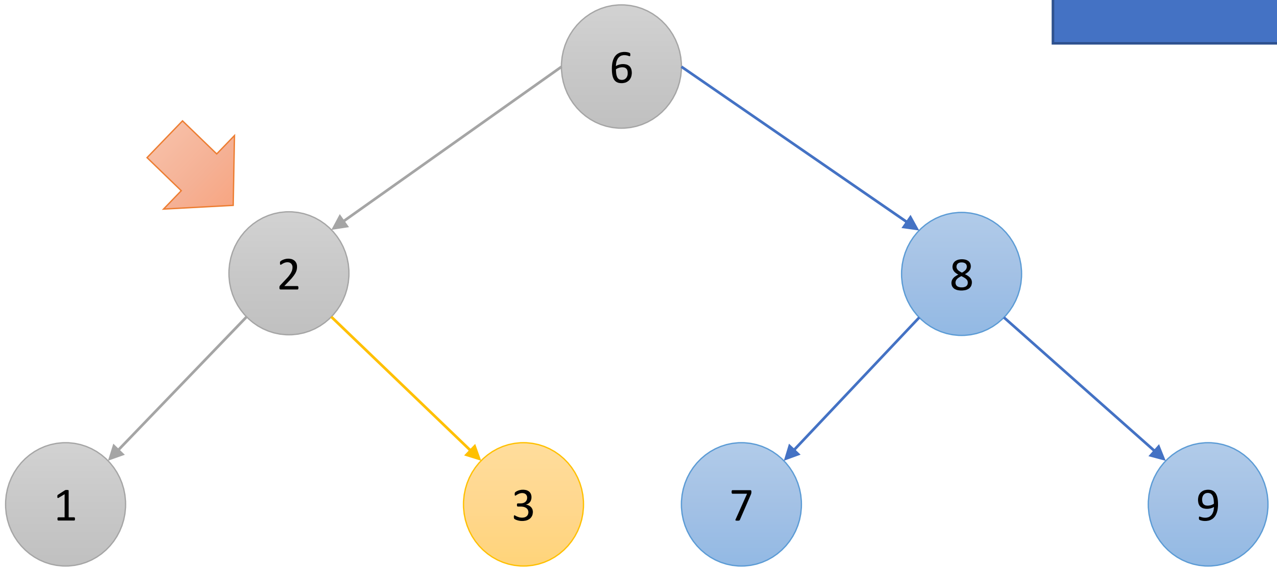
1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child



6 2 1

# Pre Order

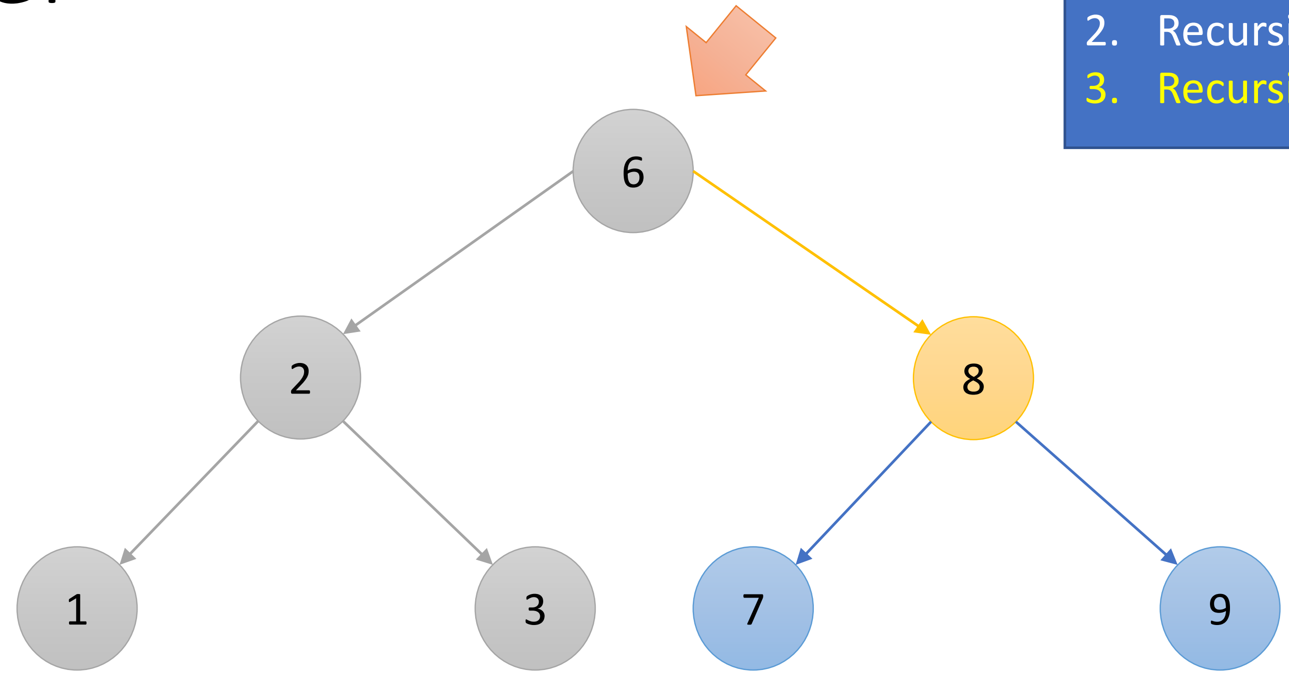
1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child



6 2 1 3

# Pre Order

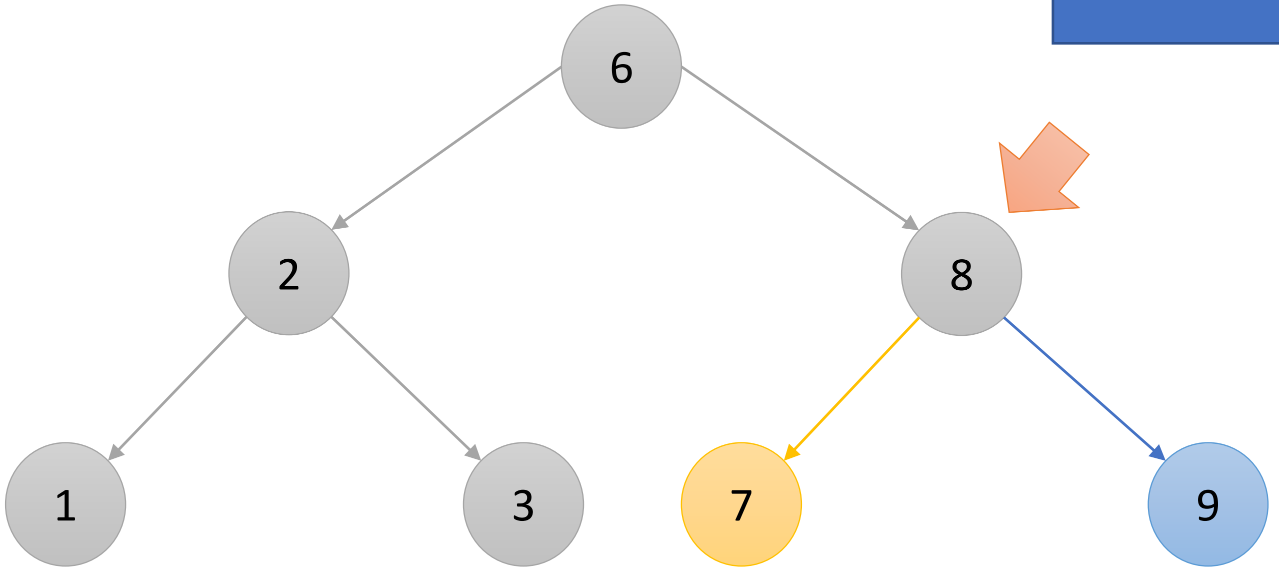
1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child



6 2 1 3 8

# Pre Order

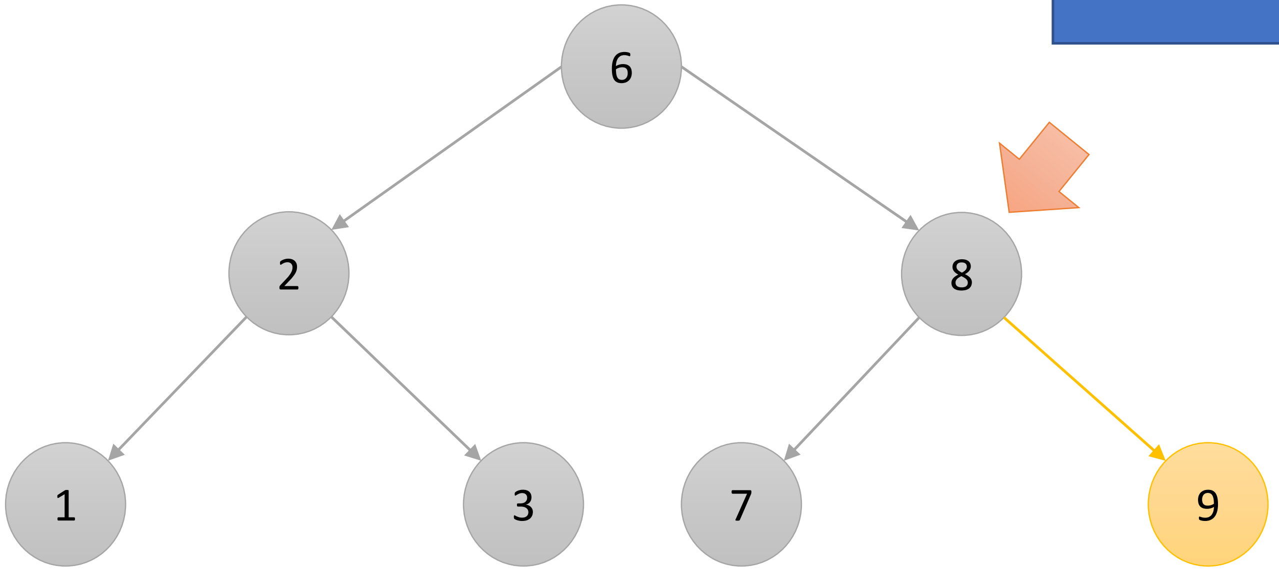
1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child



6 2 1 3 8 7

# Pre Order

1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child



6 2 1 3 8 7 9

# Activity: Pre Order

1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child

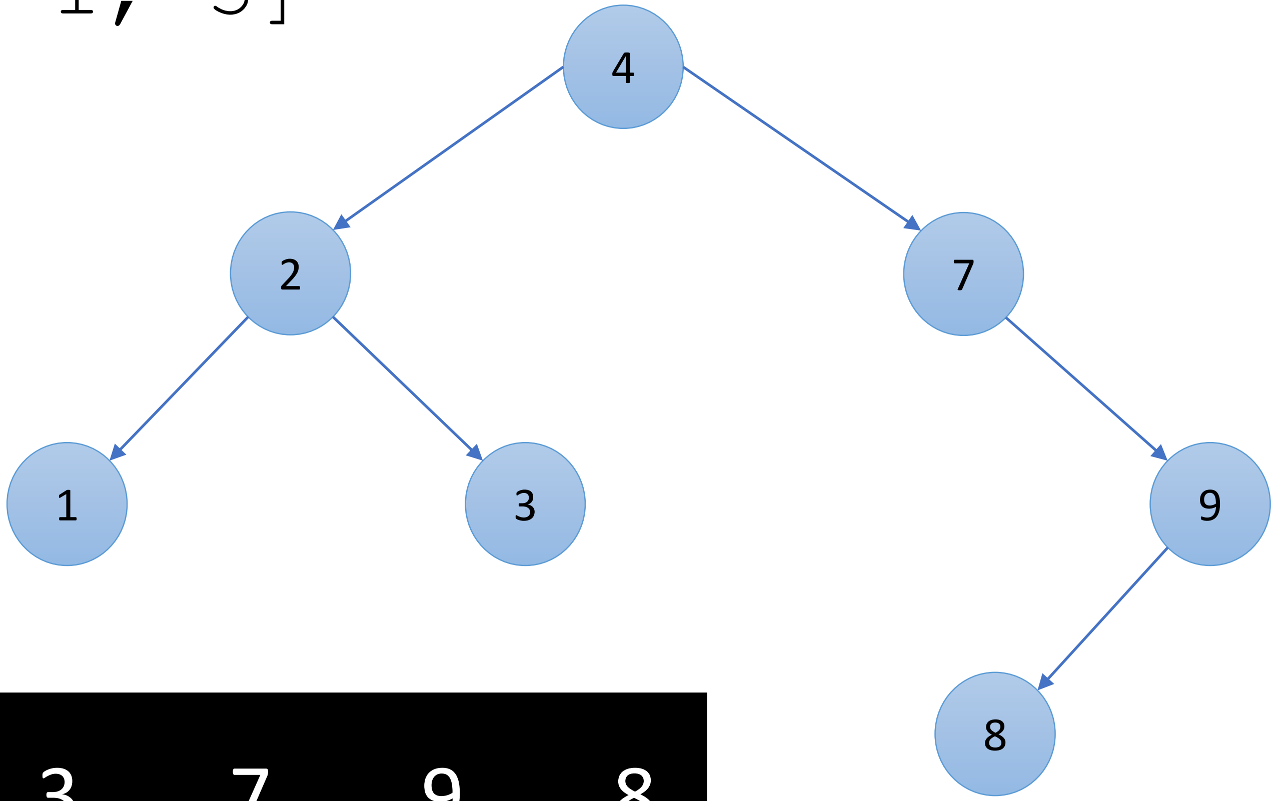
Draw a BST that represents this list of numbers, then write out the Preorder of the tree.

```
[4, 7, 9, 2, 8, 1, 3]
```

# Activity: Pre Order

1. Print the root node
2. Recursively Preorder the left child
3. Recursively Preorder the right child

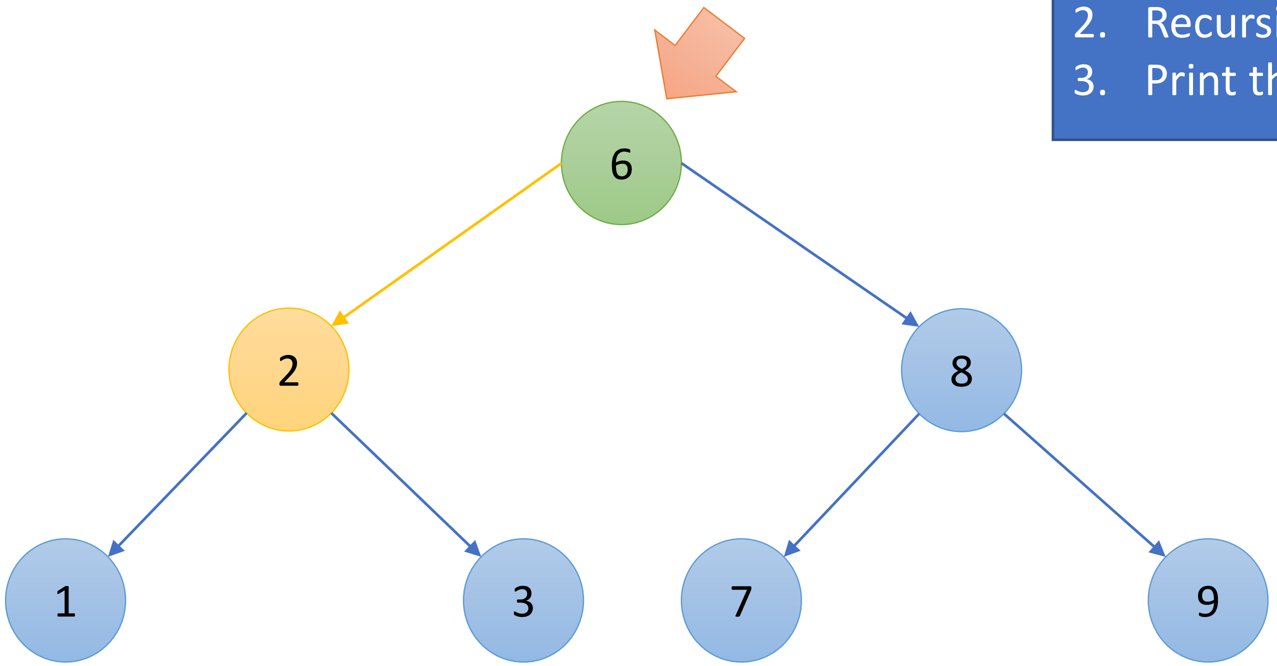
```
[4, 7, 9, 2, 8, 1, 3]
```



4 2 1 3 7 9 8

# Post Order

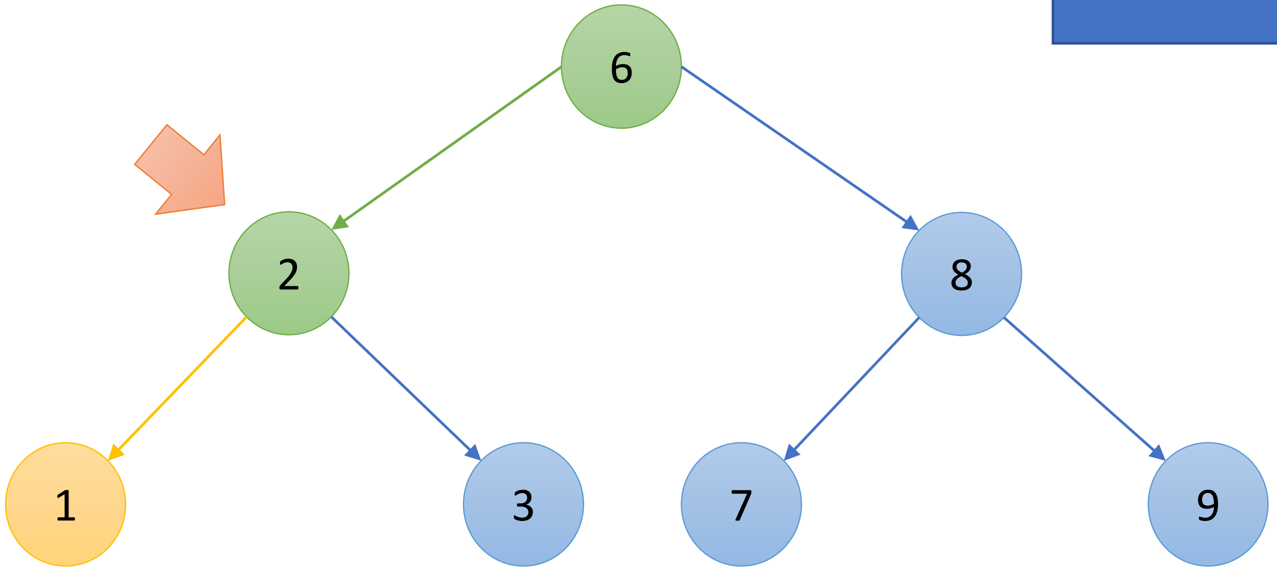
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



[redacted]

# Post Order

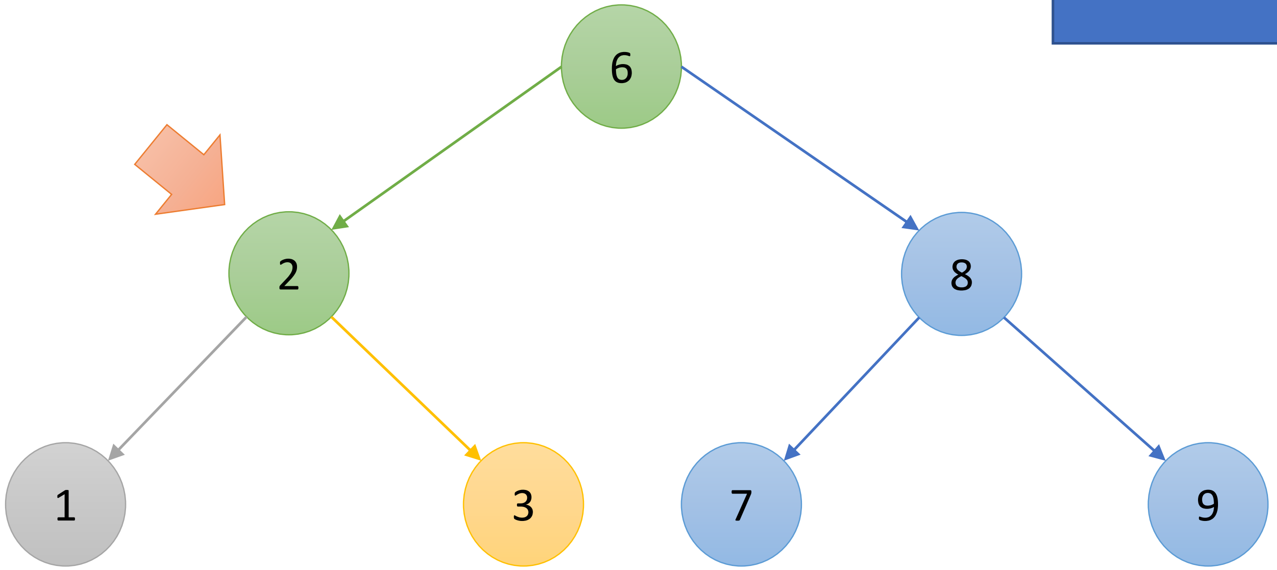
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1

# Post Order

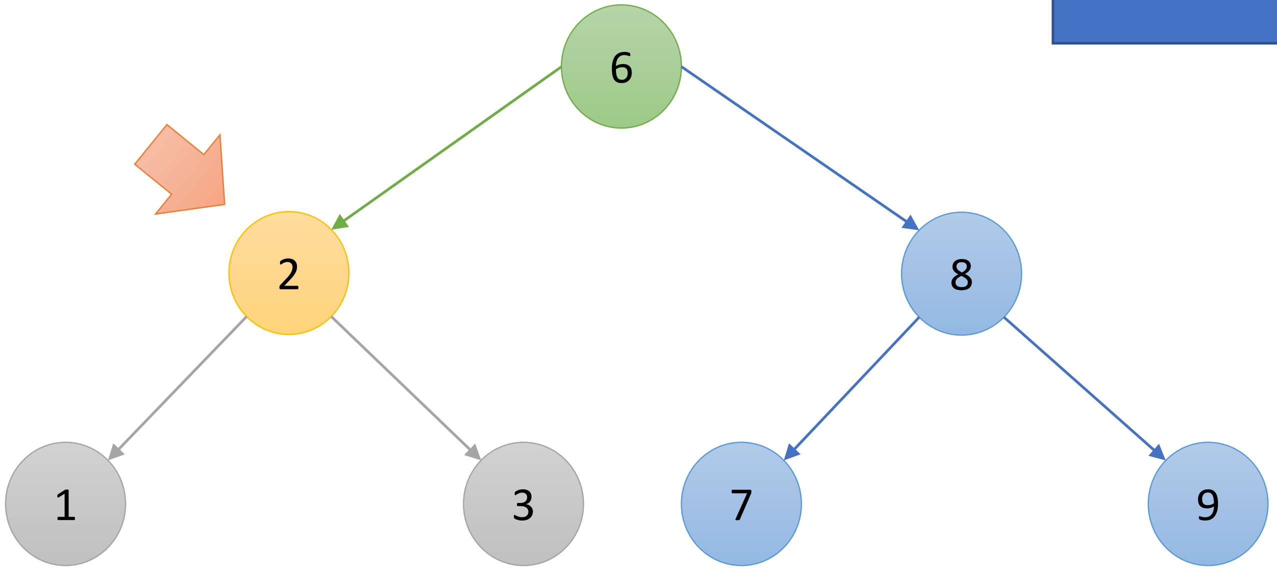
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1 3

# Post Order

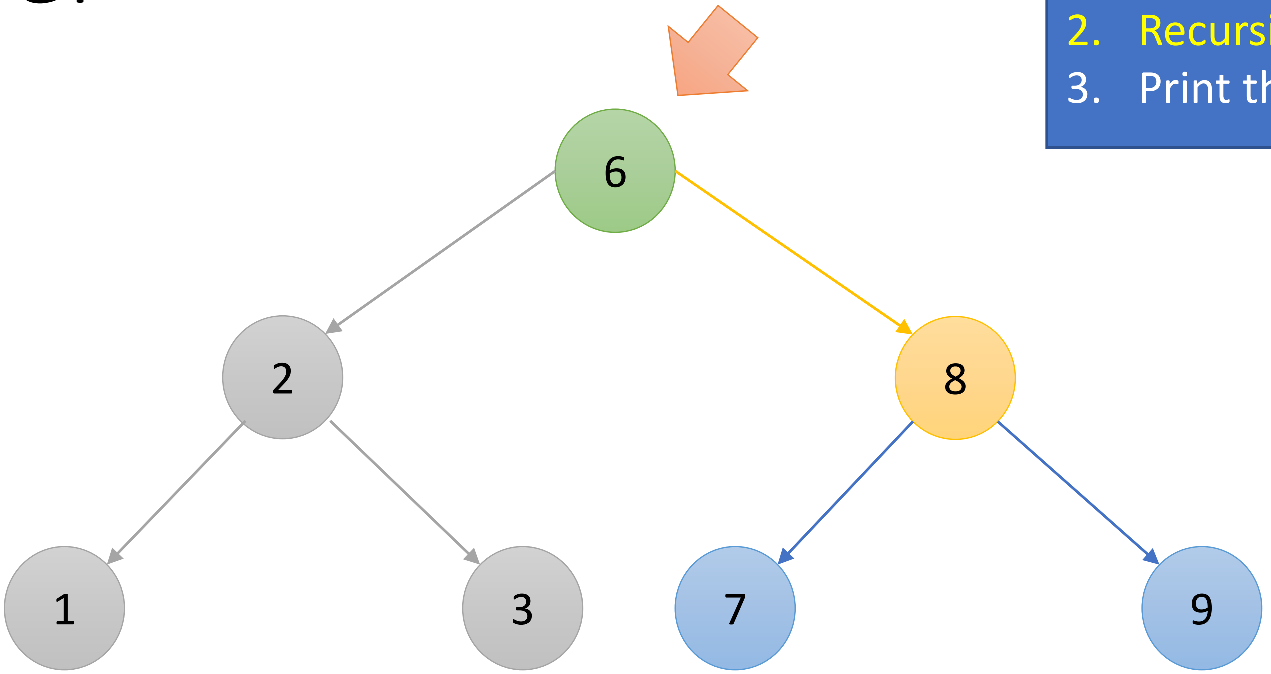
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1 3 2

# Post Order

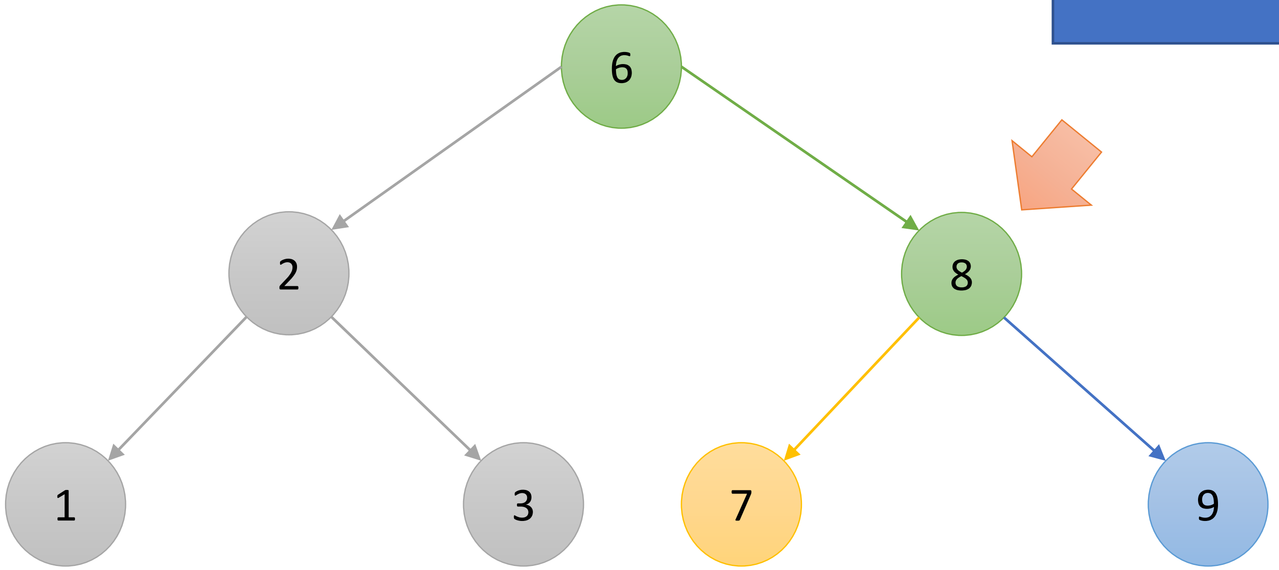
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1 3 2

# Post Order

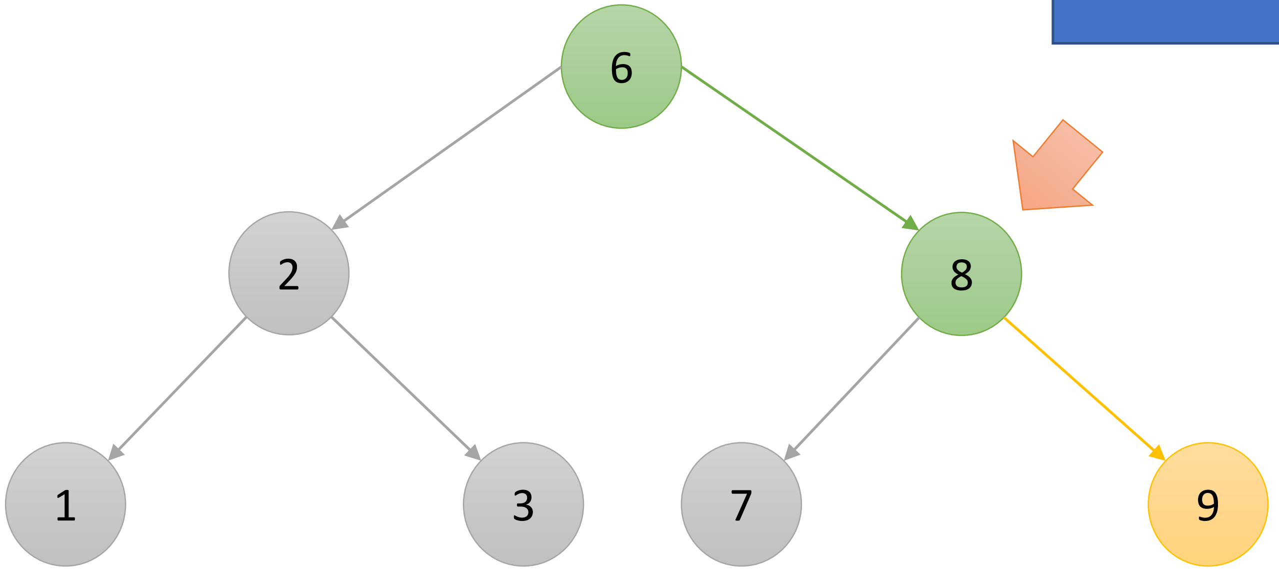
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1 3 2 7

# Post Order

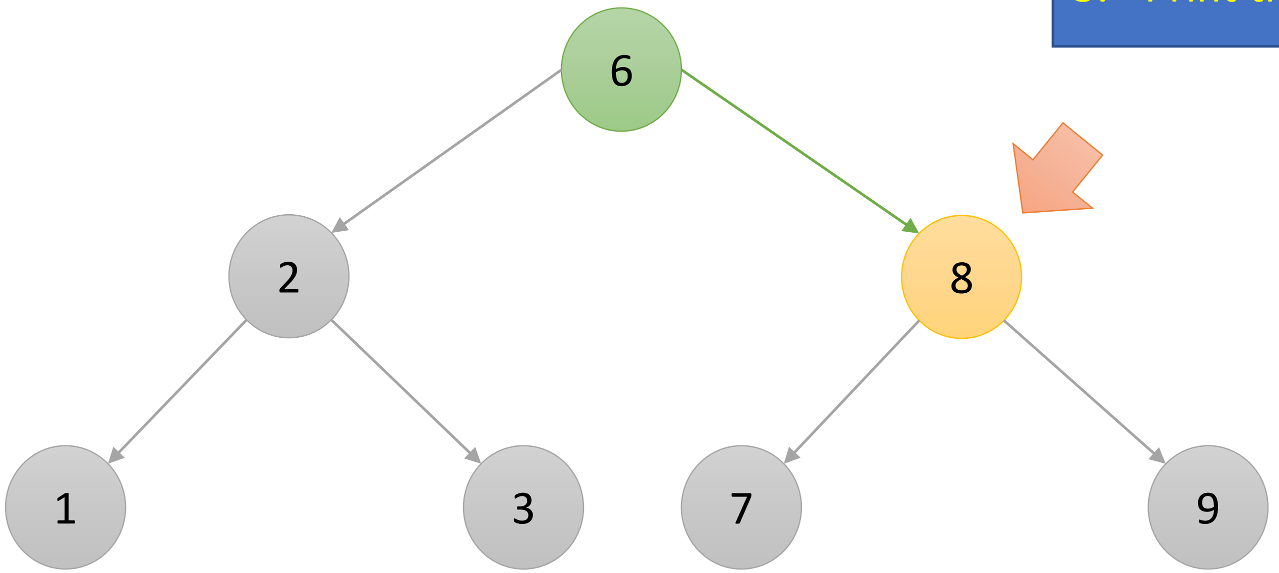
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1 3 2 7 9

# Post Order

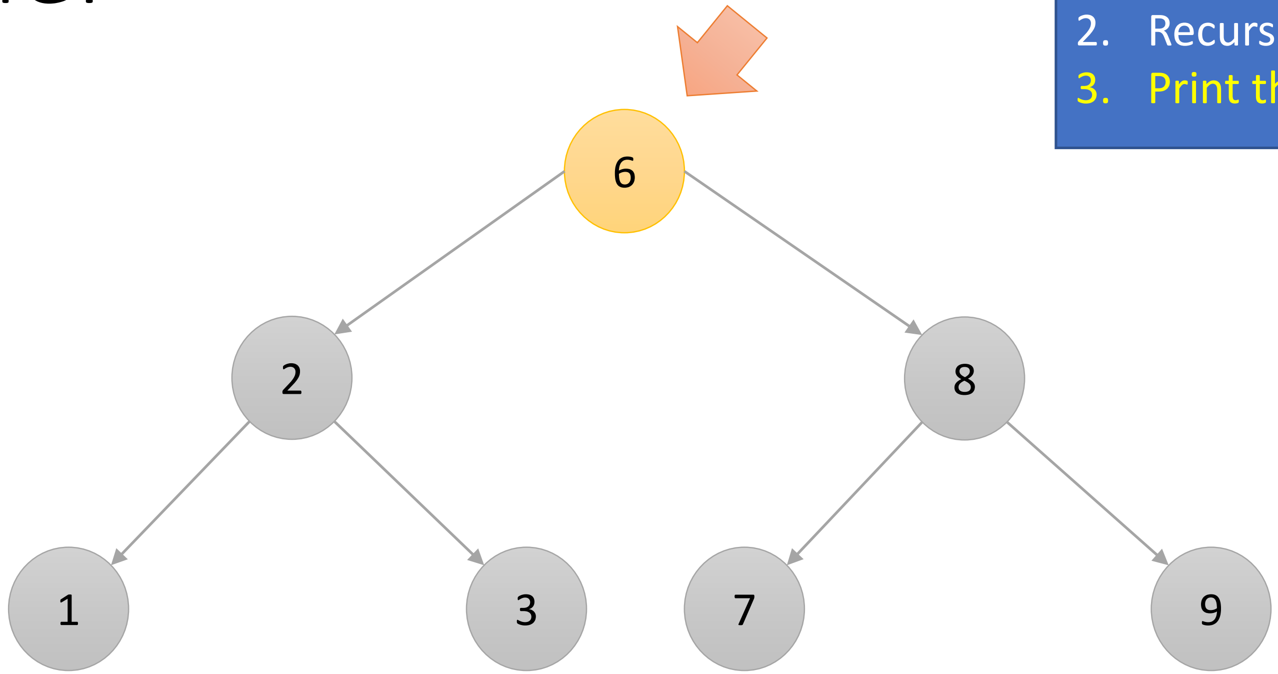
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1 3 2 7 9 8

# Post Order

1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node



1 3 2 7 9 8 6

# Activity: Post Order

1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node

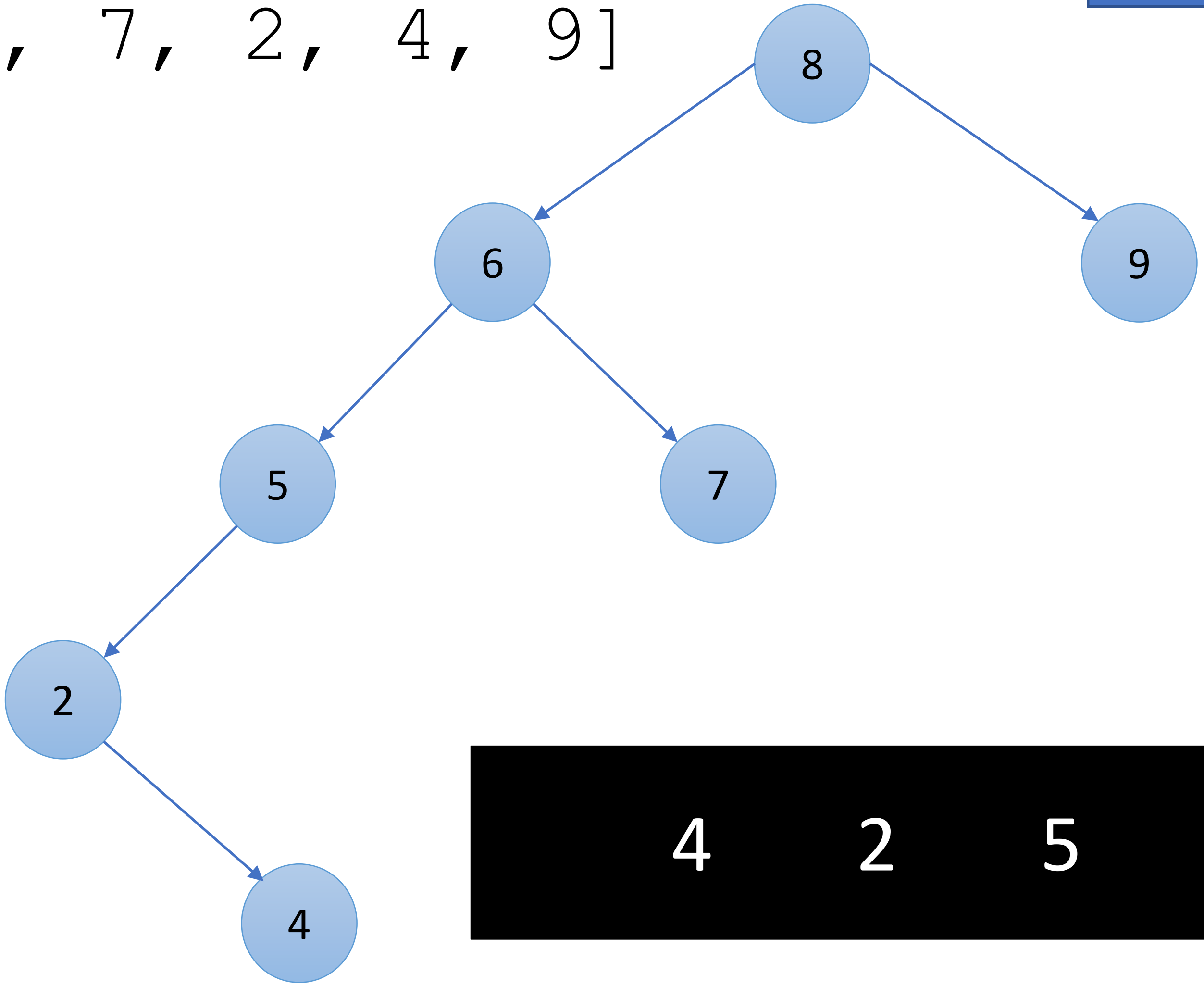
Draw a BST that represents this list of numbers, then write out the Postorder of the tree.

```
[8, 6, 5, 7, 2, 4, 9]
```

# Activity: Post Order

1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node

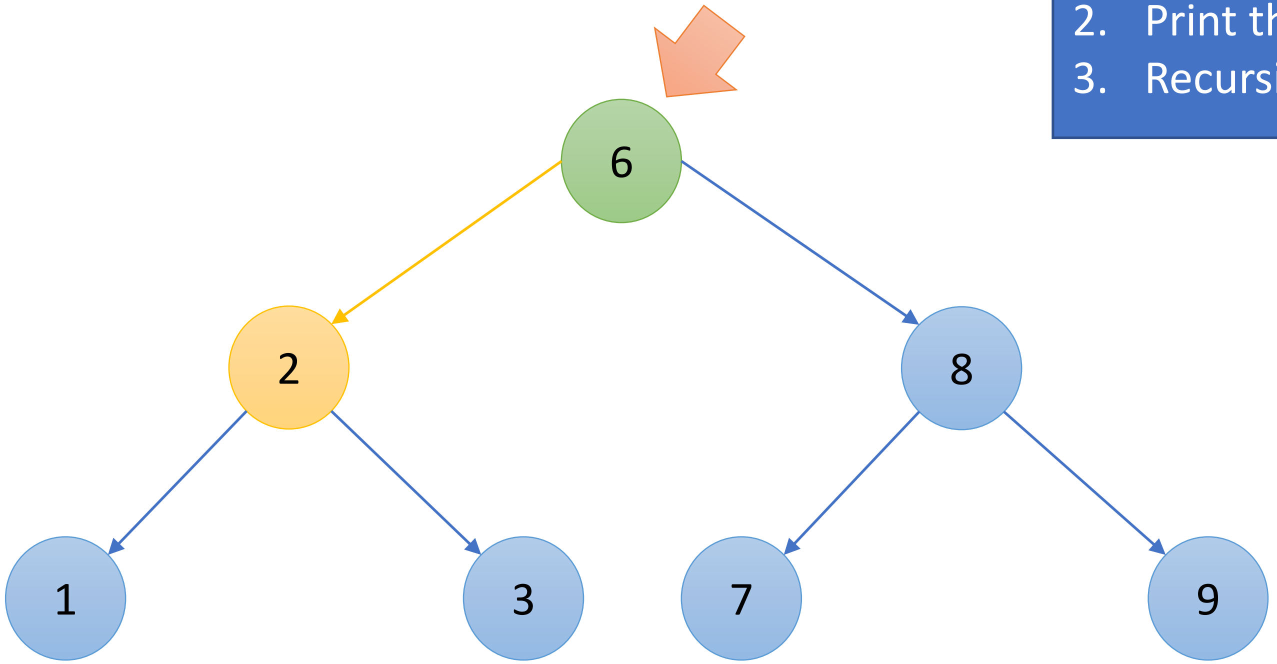
[8, 6, 5, 7, 2, 4, 9]



| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 2 | 5 | 7 | 6 | 9 | 8 |

# In Order

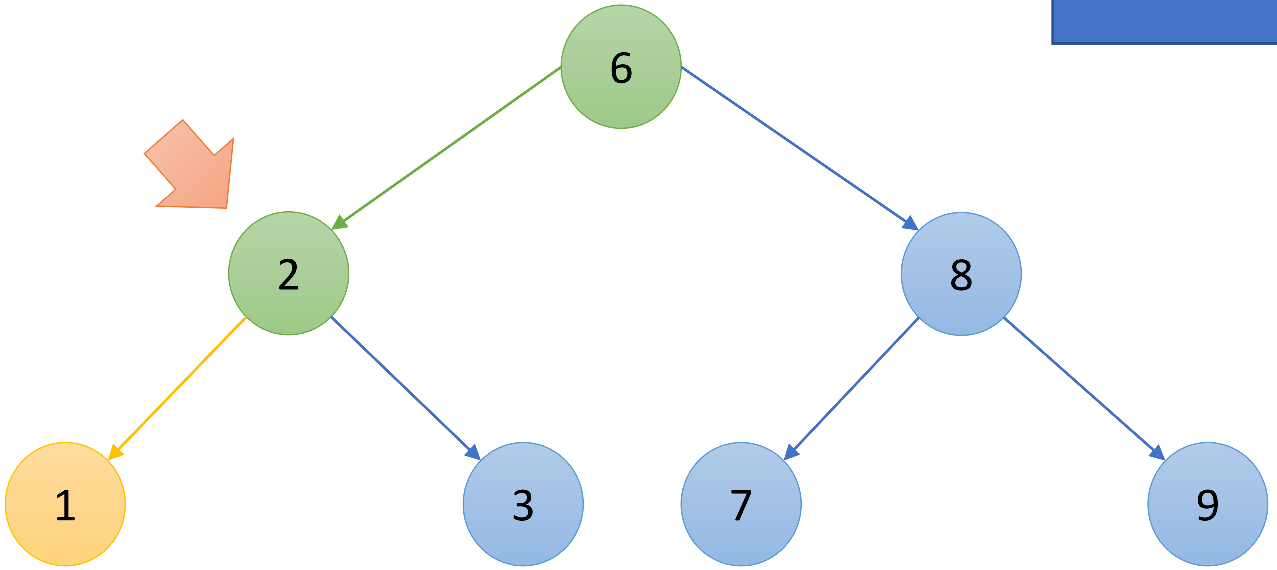
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



[redacted]

# In Order

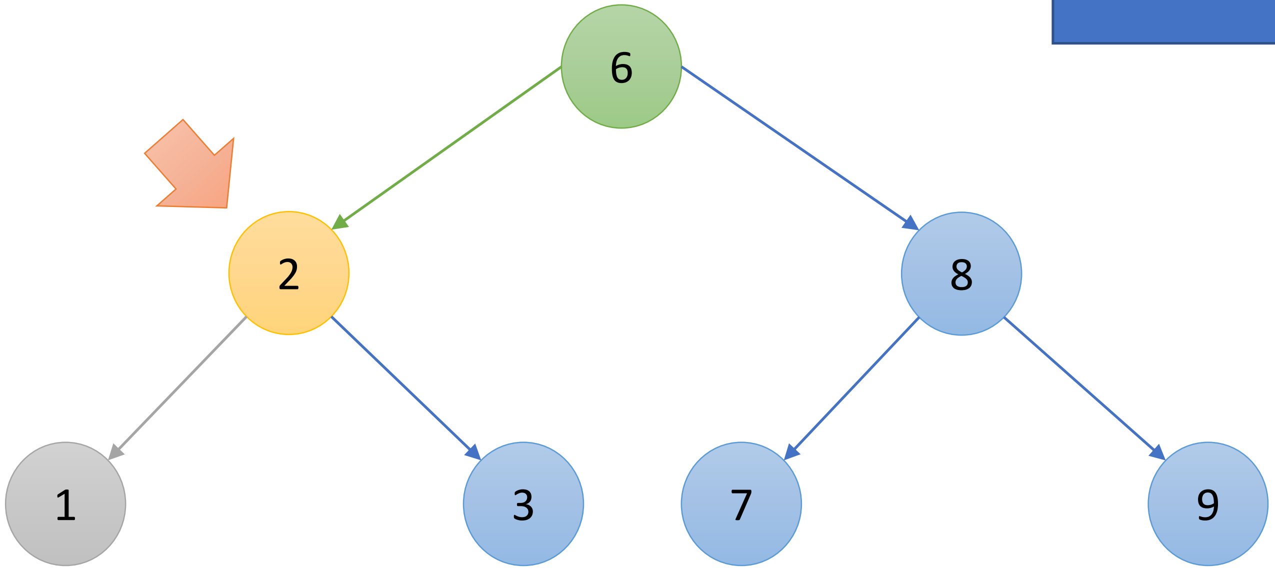
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1

# In Order

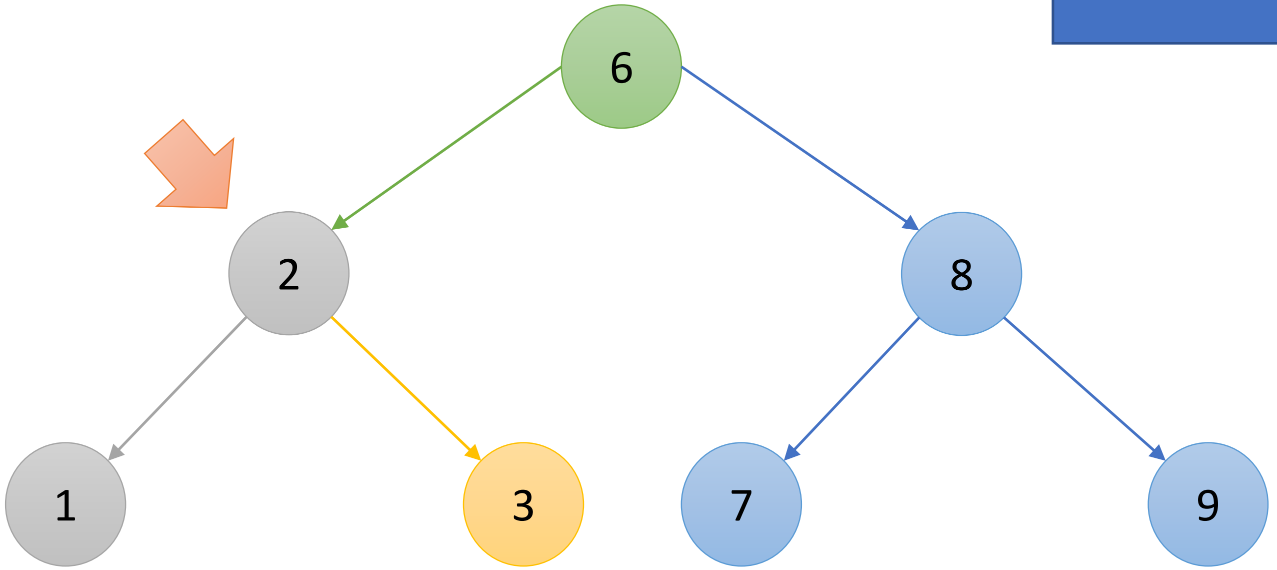
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1 2

# In Order

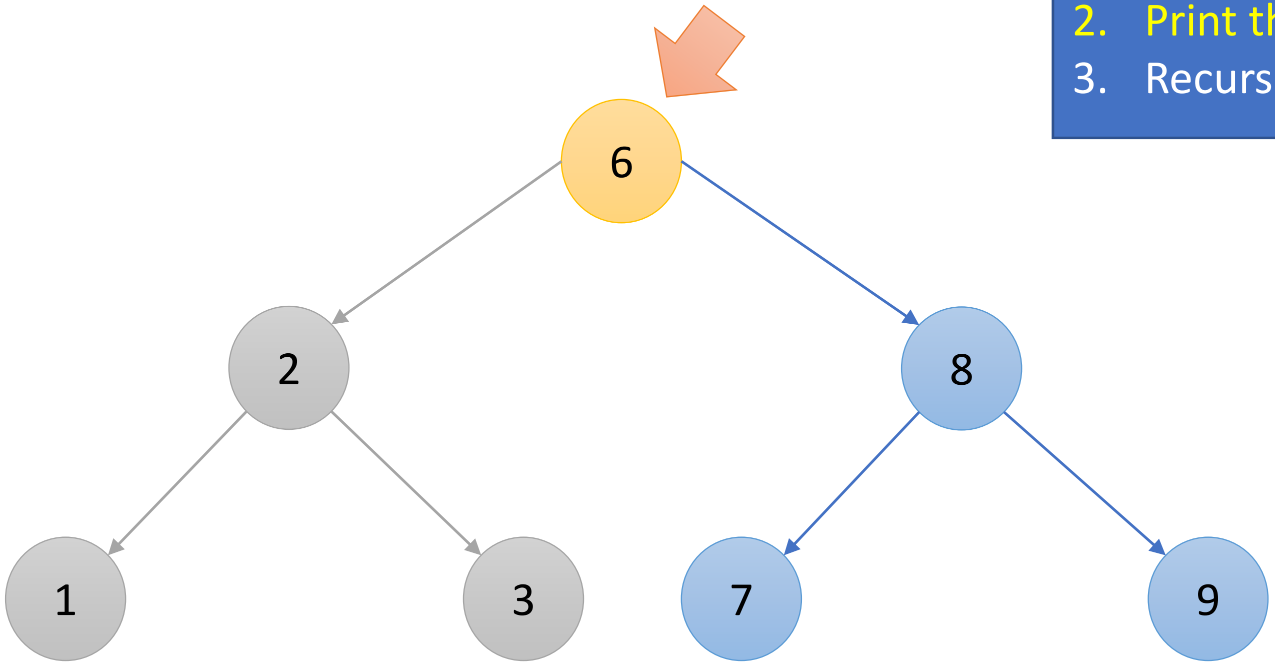
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1 2 3

# In Order

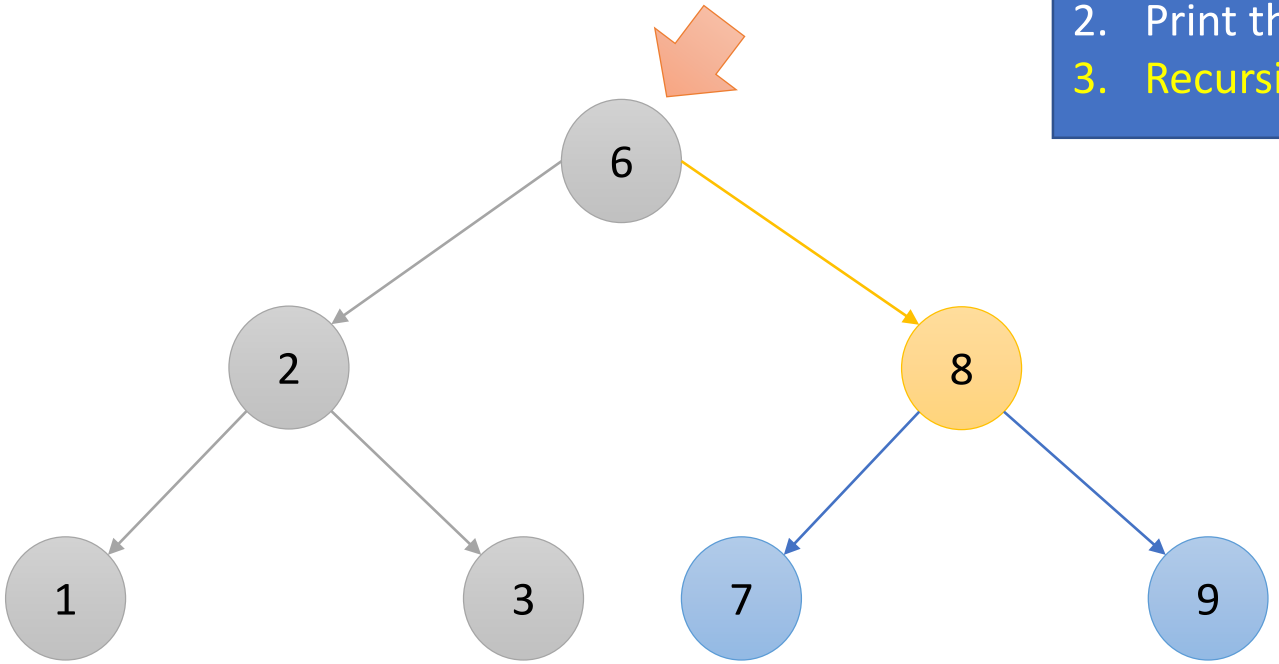
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1 2 3 6

# In Order

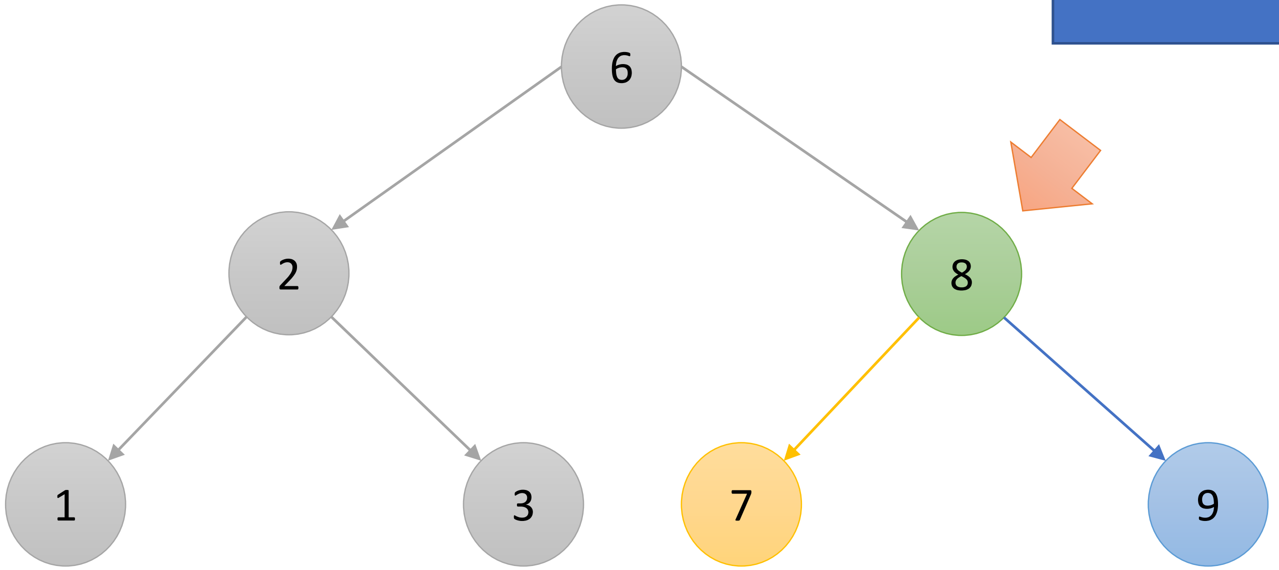
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1 2 3 6

# In Order

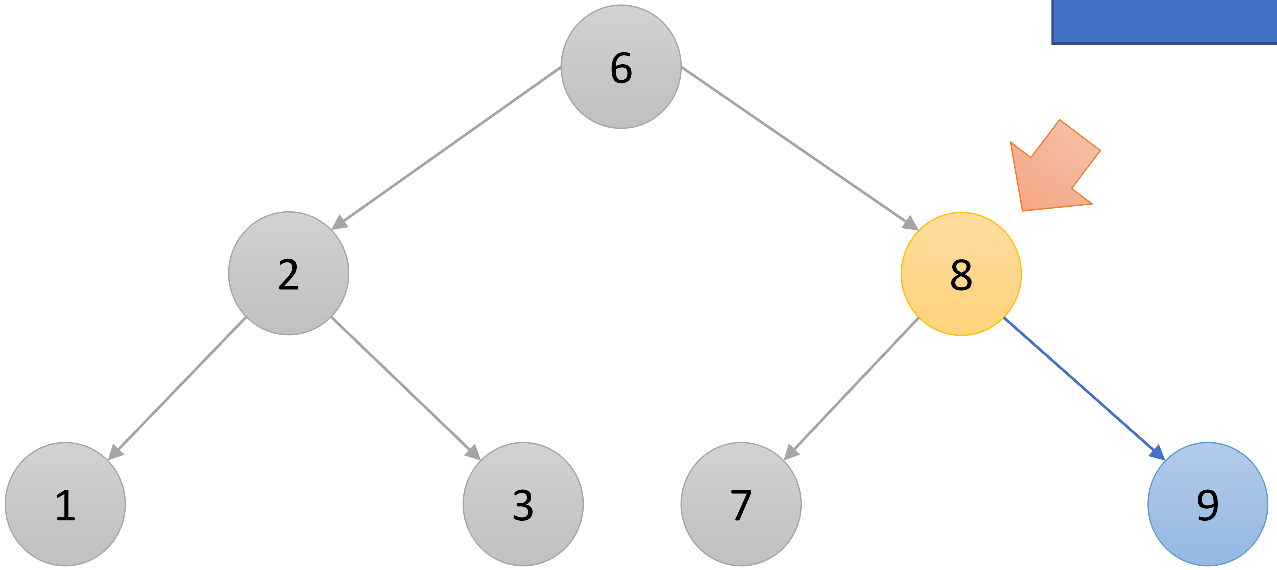
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1 2 3 6 7

# In Order

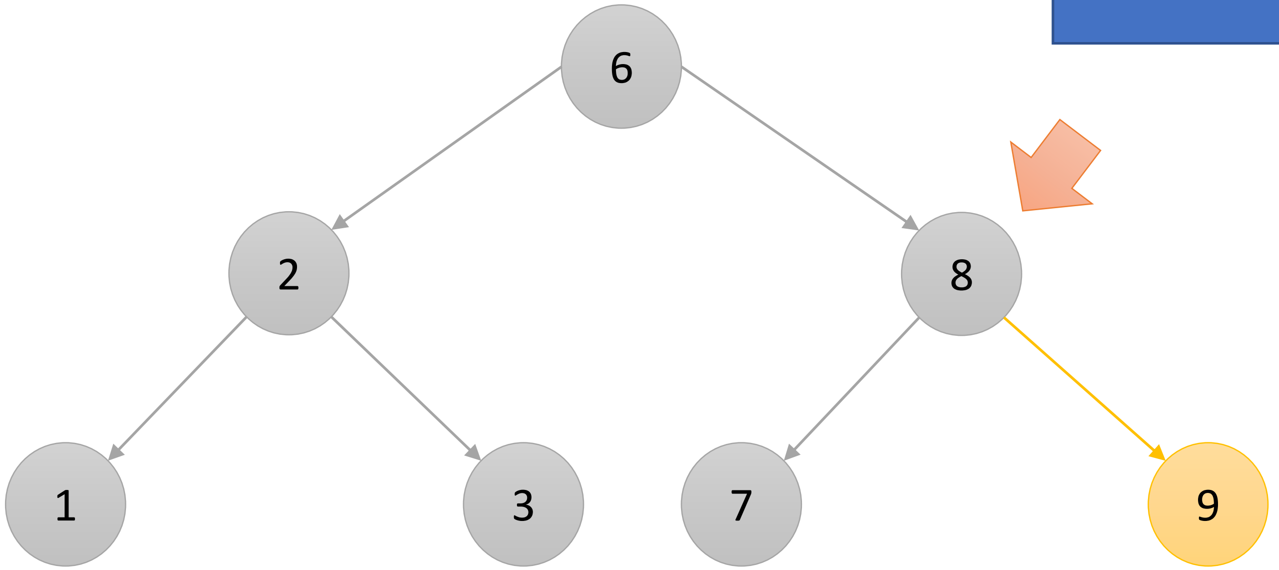
1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1 2 3 6 7 8

# In Order

1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child



1 2 3 6 7 8 9

# Activity: In Order

1. Recursively Inorder the left child
2. Print the root node
3. Recursively Inorder the right child

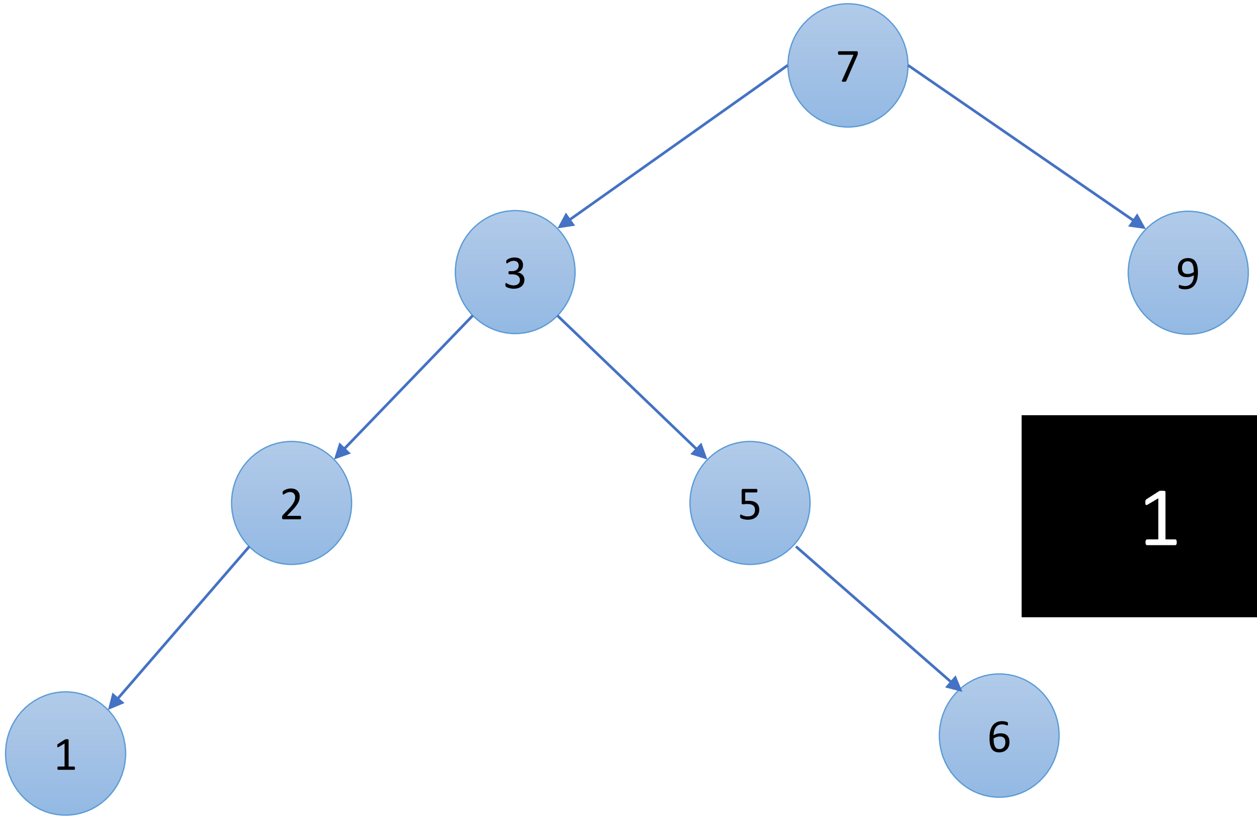
Draw a BST that represents this list of numbers, then write out the Inorder of the tree.

```
[7, 3, 5, 2, 6, 1, 9]
```

# Activity: Post Order

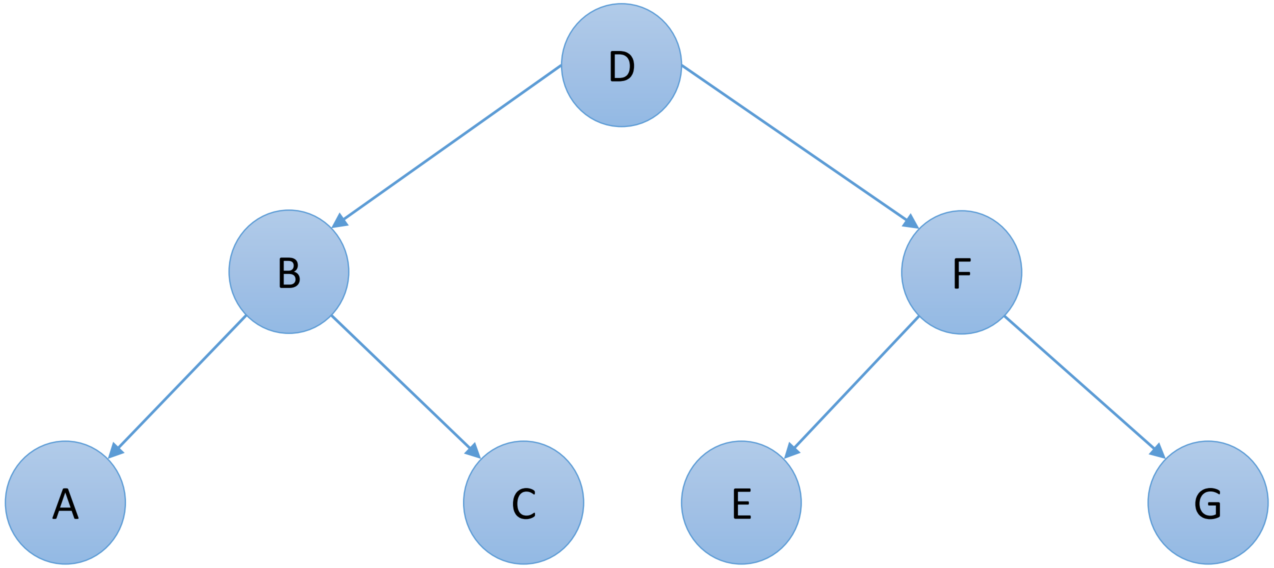
1. Recursively Postorder the left child
2. Recursively Postorder the right child
3. Print the root node

```
[7, 3, 5, 2, 6, 1, 9]
```



1 2 3 5 6 7 9

# Activity: Write each of the three representations of the following tree

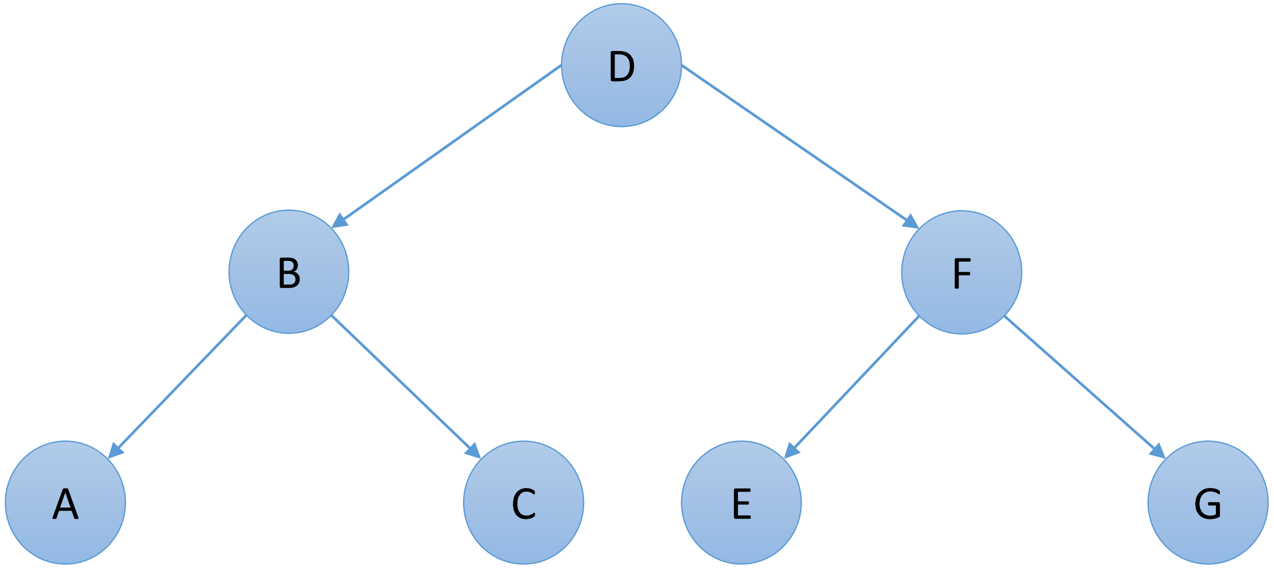


Pre:

In:

Post:

# Activity: Write each of the three representations of the following tree



Pre: D B A C F E G

In: A B C D E F G

Post: A C B E G F D

# Activity: Write each of the three representations of the following tree

Pre: D B A C F E G

In: A B C D E F G

In order prints the values in sorted order!

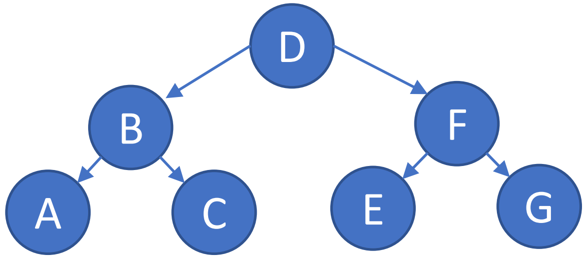
Post: A C B E G F D

# Activity: Create three new trees by adding the values (in the order provided)

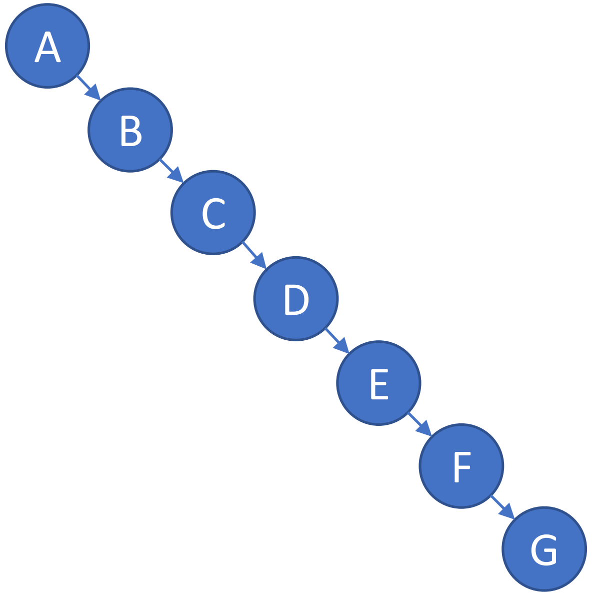
| | |
|---|---|
| Pre: | D B A C F E G |
| In: | A B C D E F G |
| Post: | A C B E G F D |

# Activity: Create three new trees by adding the values (in the order prvided)

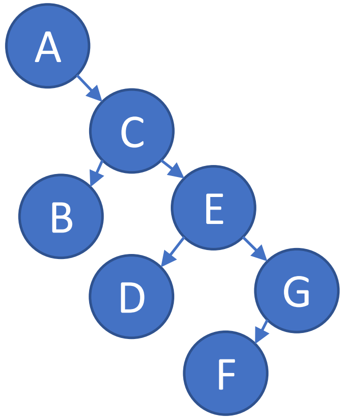
Pre:  D B A C F E G



In: A B C D E F G



Post:  A C B E G F D



Pre Order rebuilds the original tree!

# Uses of Pre/In/Post Order Traversals

Pre Order: Can be used to rebuild original tree

(save tree data to a text file/rebuild tree from text file)

In Order: Display data *in order* (sorted)

Post Order: Useful when processing/deleting nodes from a tree