Data Structures

Binary Tree Traversal 3

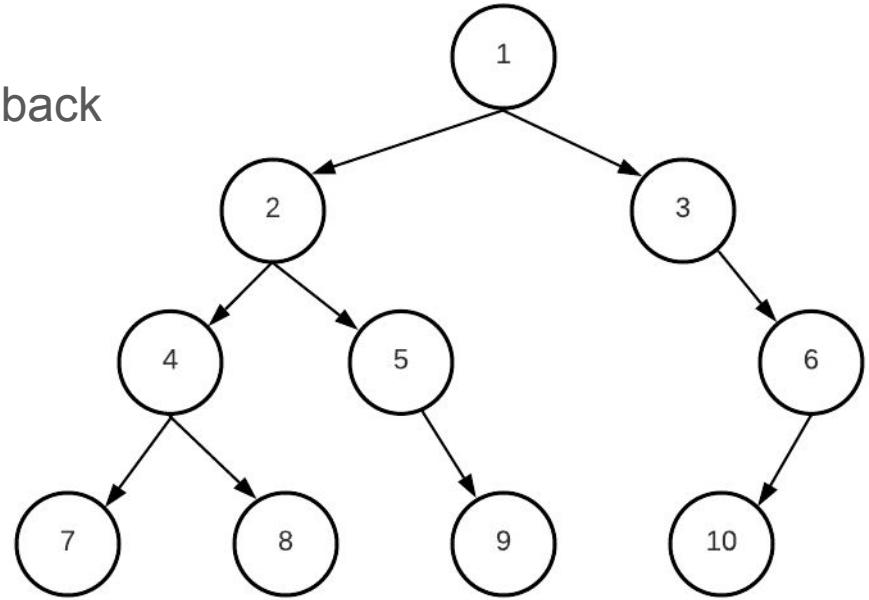
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

What is the in-order traversal?

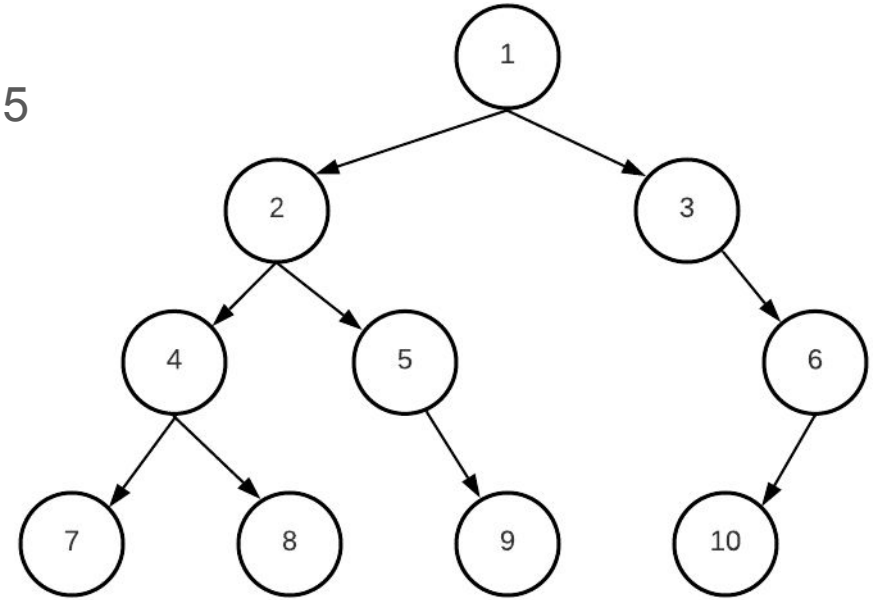
- Observe the code keep going to the **left**
- 1->2->4->7: No no further left.
- At 7: **Print** data ⇒ 7. No right return. Go back
- At 4: left call is done. **Print** data ⇒ 4
- Go right to 8

```
void print_inorder(Node* current) {
    if(!current)
        return;
    print_inorder(current->left);
    cout << current->data << " ";
    print_inorder(current->right);
}
```



What is the in-order traversal?

- At 8: No left/right. **Print** data ⇒ 8. Go back
- At 4: left & right done. Go back.
- At 2: left done. **Print** data ⇒ 2. Go right: 5

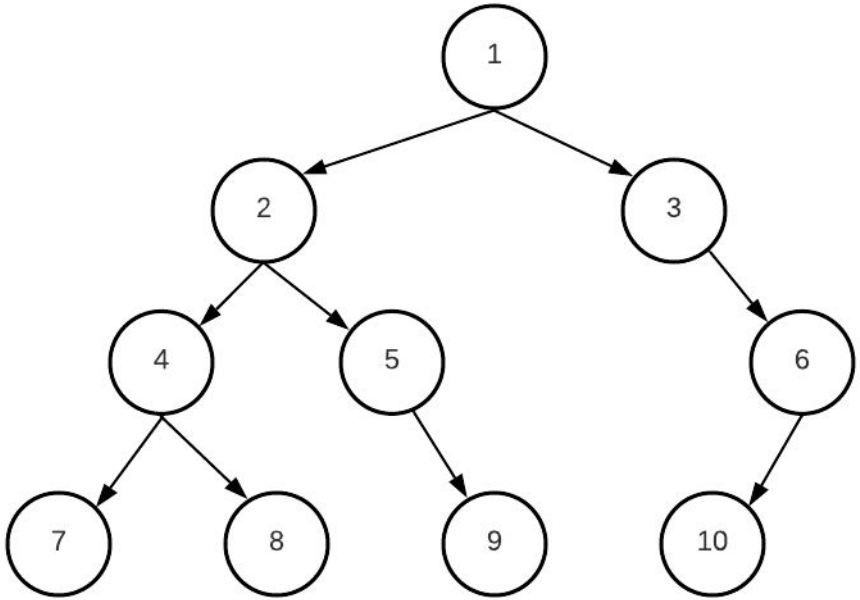



```
27 void print_inorder(Node* current) {
28     if(!current)
29         return;
30     print_postorder(current->left);
31     cout << current->data << " ";
32     print_postorder(current->right);
33 }
```

What is the in-order traversal?

- At 5: no left. **Print** data ⇒ 5 and Go right ⇒ 9
- At 9. **Print** 9 and go back
- At 5 left and right done: go back
- At 2 left and right done: go back
- At 1: left done. **Print** 1. Go right at 3

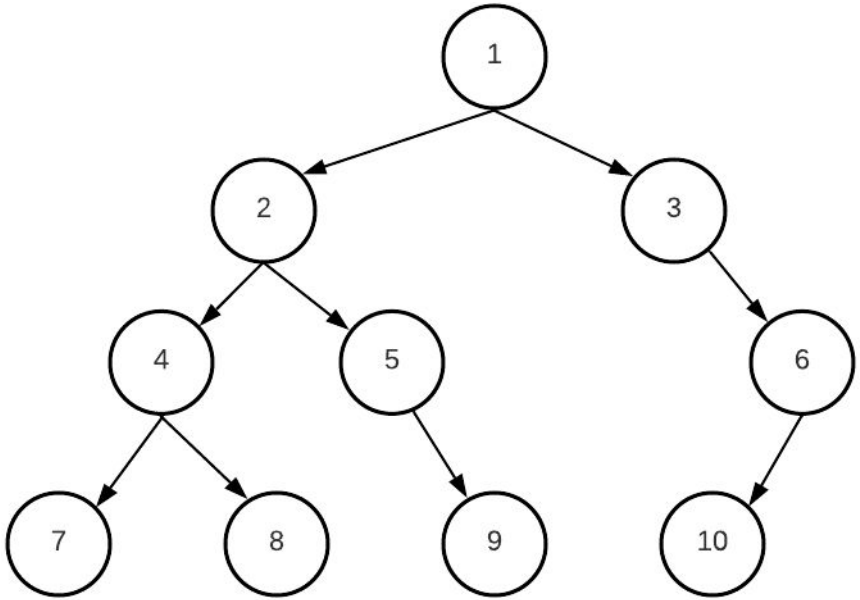
```
27 void print_inorder(Node* current) {
28     if(!current)
29         return;
30     print_postorder(current->left);
31     cout << current->data << " ";
32     print_postorder(current->right);
33 }
```



What is the in-order traversal?

- At 3: no left. **Print** 3. Go right at 6
- At 6: Go left at 10
- At 10: no left/right. **Print** 10. Go parent 6
- At 6: left done. **Print** 6. No right.
- Go parent. Go parent. Done.
- 7 4 8 2 5 9 1 3 10 6

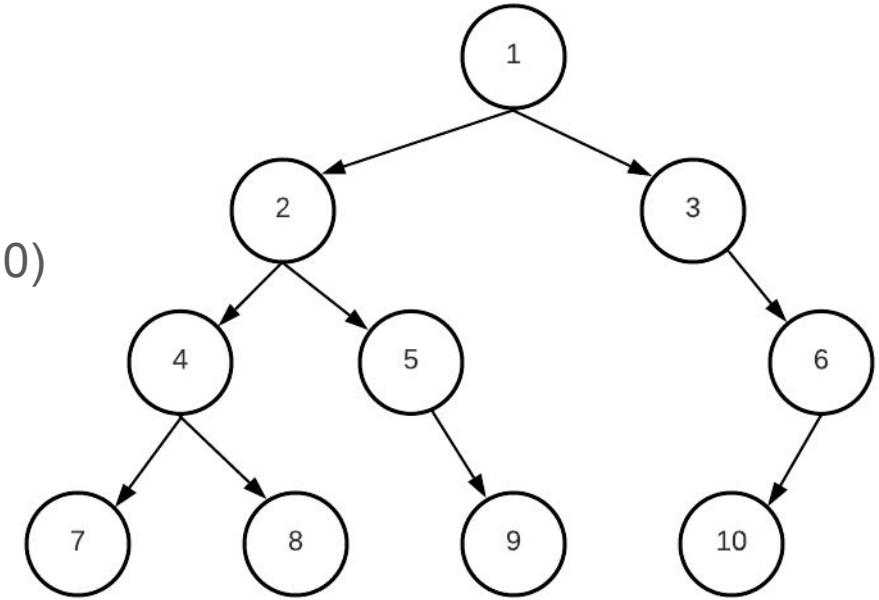
```
27 void print_inorder(Node* current) {
28     if(!current)
29         return;
30     print_postorder(current->left);
31     cout << current->data << " ";
32     print_postorder(current->right);
33 }
```



What is the in-order traversal?

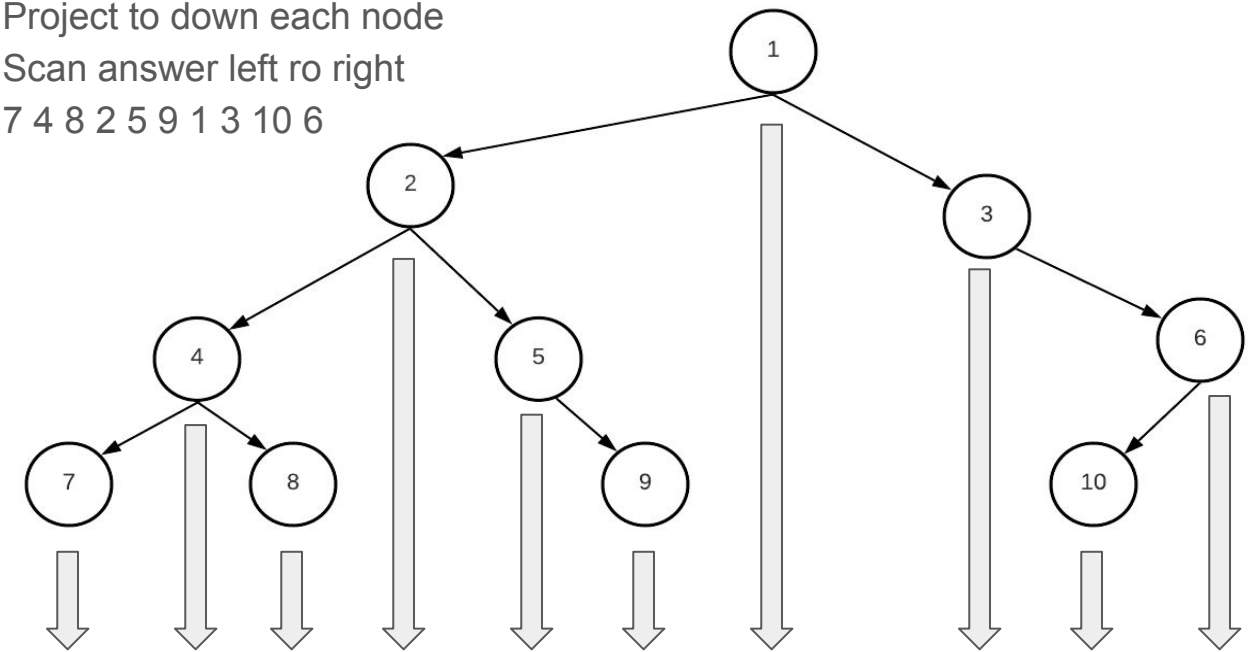
- So from any node: keep going left
  - Once no left or left is done, print current node
  - Go right and repeat
  - No right? Go parent
- **Most left node** is first printed: 7
- **Most right node** is last printed: 6 (not 10)

```
void print_inorder(Node* current) {
    if(!current)
        return;
    print_inorder(current->left);
    cout << current->data << " ";
    print_inorder(current->right);
}
```



What is the in-order traversal? **Visually**

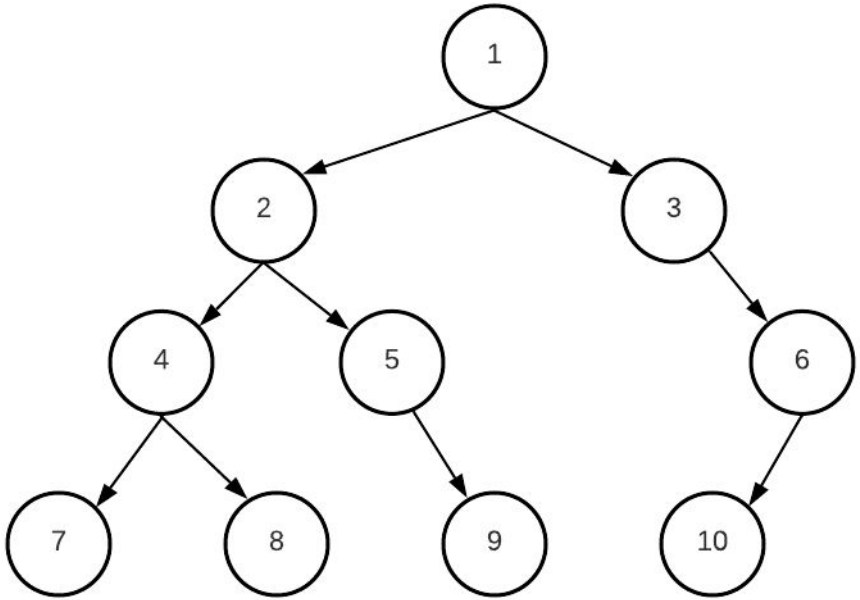
- If you draw it such as all my left nodes are on left and all right on right?
  - Project to down each node
  - Scan answer left ro right
  - 7 4 8 2 5 9 1 3 10 6



What is the post-order traversal?

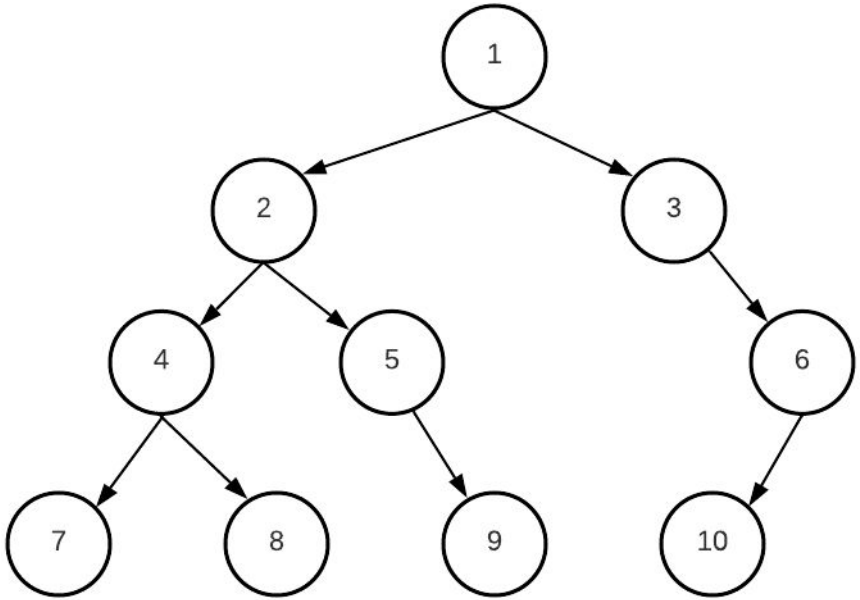
- Observe the code keep going to the **most left**
- 1->2->4->7: No no further left.
- Then move to right node
- Then again find most left
- Once no right or right done, print node
- Gp back to parent

```
19 void print_postorder(Node* current) {
20     if(!current)
21         return;
22     print_postorder(current->left);
23     print_postorder(current->right);
24     cout << current->data << " ";
25 }
```



What is the post-order traversal?

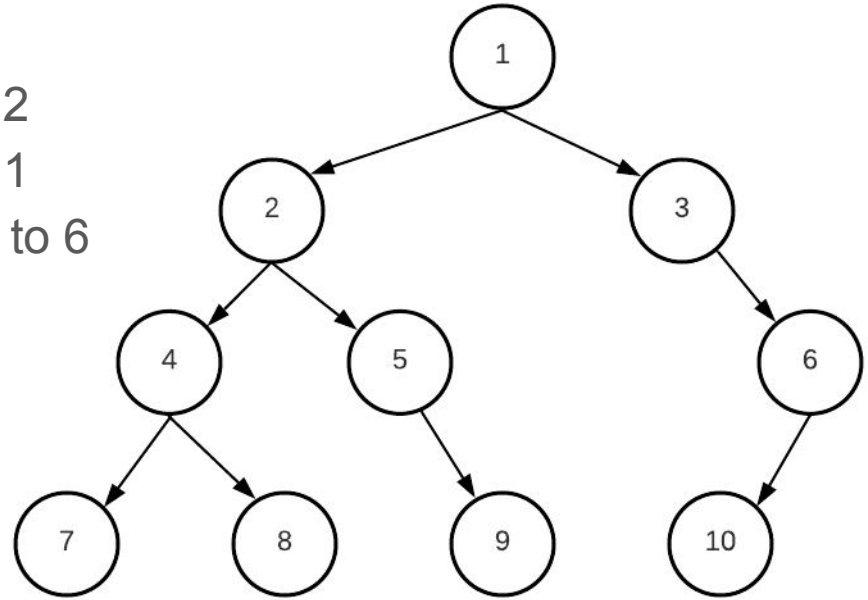
- From 1 goes to 7. No right. **Print** 7.
- Go parent 4. Go right at 8
- At 8: no left/right. **Print** 8. Go parent 4
- At 4: right is done. **Print** 4. Go parent 2
- At 2. Left done. Go right at 5.




```
19 void print_postorder(Node* current) {
20     if(!current)
21         return;
22     print_postorder(current->left);
23     print_postorder(current->right);
24     cout << current->data << " ";
25 }
```

What is the post-order traversal?

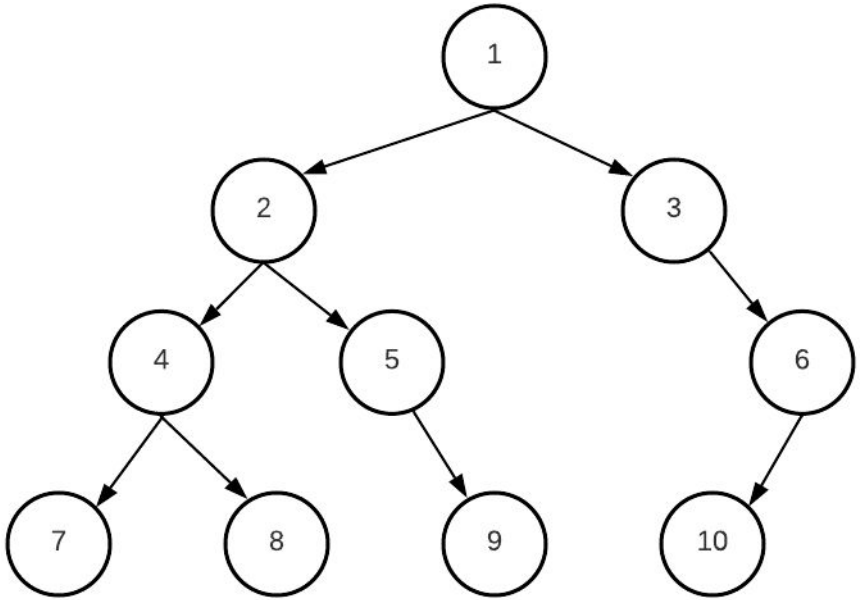
- At 5: No left. Go right 9
- At 9: **print** 9 and go back
- Back to 5: right done. **Print 5**. Go parent 2
- Back to 2: right done. **Print 2**. Go parent 1
- Back to 1: Go right at 3, which goes right to 6




```
19 void print_postorder(Node* current) {
20     if(!current)
21         return;
22     print_postorder(current->left);
23     print_postorder(current->right);
24     cout << current->data << " ";
25 }
```

What is the post-order traversal?

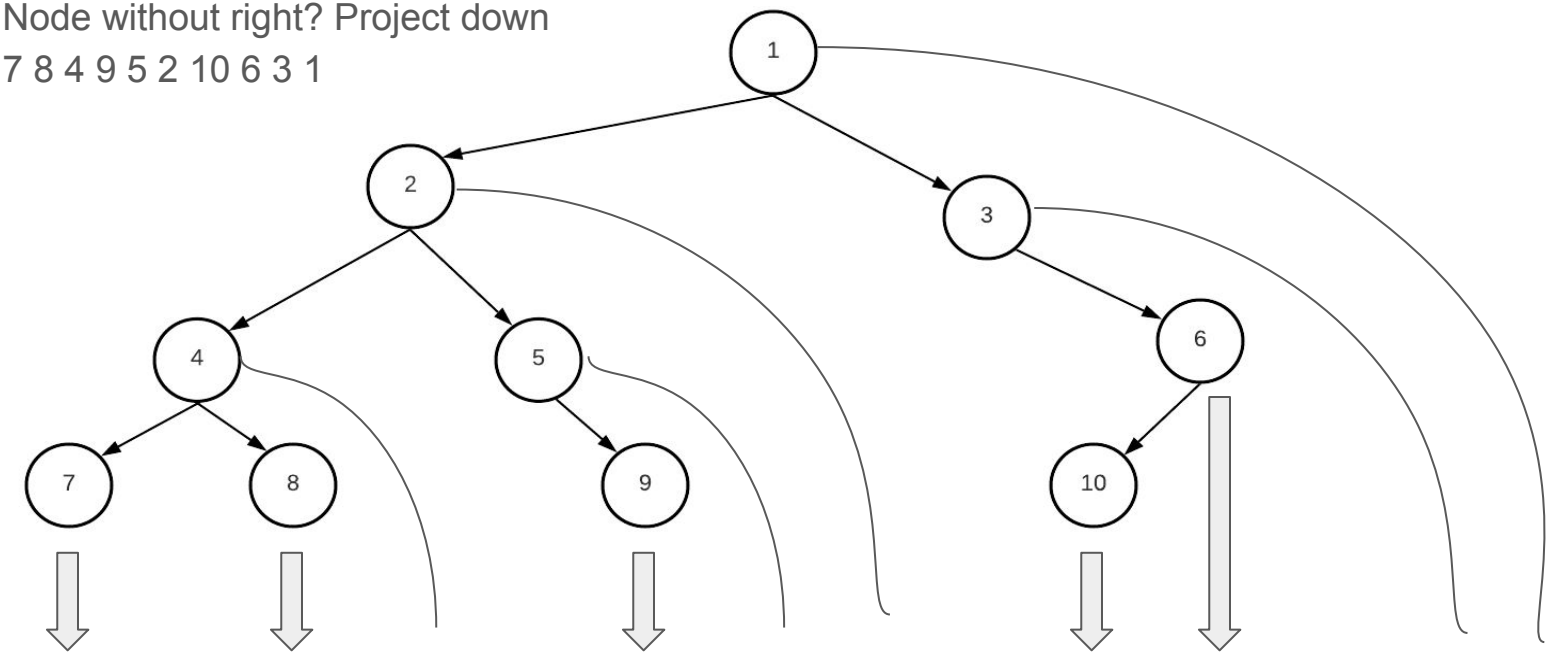
- At 6: go left to 10. **Print 10** and go back
- **Print 6, print 3, print 1**
- In total: 7 8 4 9 5 2 10 6 3 1
- The root, is the last printed value!




```
19 void print_postorder(Node* current) {
20     if(!current)
21         return;
22     print_postorder(current->left);
23     print_postorder(current->right);
24     cout << current->data << " ";
25 }
```

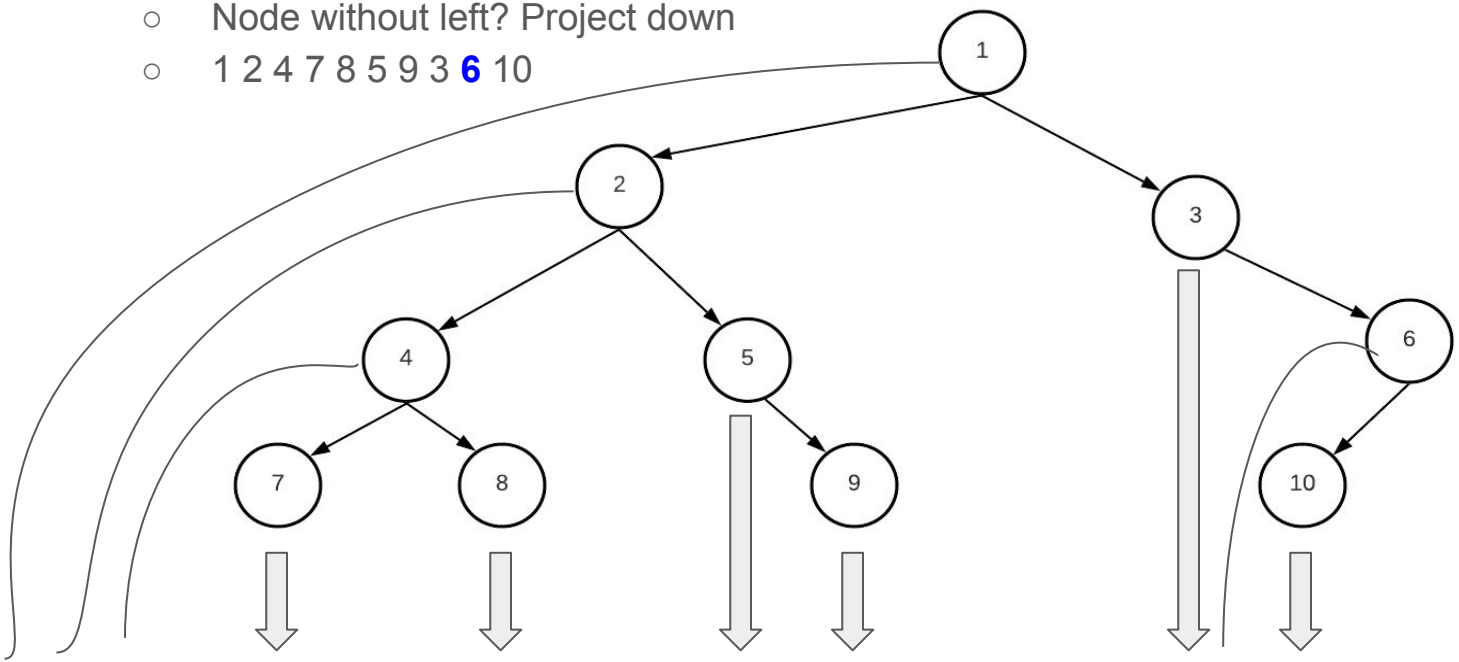
What is the post-order traversal? **Visually**

- Project its node **after its right subtree**
  - Node without right? Project down
  - 7 8 4 9 5 2 10 6 3 1



What is the pre-order traversal? **Visually**

- Project its node **before its left subtree**
  - Node without left? Project down
  - 1 2 4 7 8 5 9 3 **6** 10



Computations

- Most of tasks follows one of these traversal strategies
  - Find minimum value of a tree
  - Find height of a tree
  - Count how many leaf or non-leaf nodes
  - Etc
- In all of them you need to follow some style. Go preorder: **VLR**
  - *Proper basecase handling*
  - *Compute something based current->data*
  - *Compute left subtree recursively*
  - *Compute right subtree recursively*
  - *Compute the overall of these **3 values***
  - Examples in homework

Computations

- For some tasks, we might easily compute inorder traversal and check
  - Save the traversal in an array
  - Do the operation if applicable
    - Tree sum, min, max, if a value exists, if the tree has duplicate values
  - But the **downside**, we have to traverse the whole tree. E.g. not efficient for value searching

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."