COURSE HANDOUT

Course Code	ACSC13
Course Name	Design and Analysis of Algorithms
Class / Semester	IV SEM
Section	A-SECTION
Name of the Department	CSE-CYBER SECURITY
Employee ID	IARE11023
Employee Name	Dr K RAJENDRA PRASAD
Topic Covered	Efficient non recursive binary tree traversal algorithms
Course Outcome/s	Make Use of appropriate tree traversal techniques for finding shortest path.
Handout Number	21
Date	

Content about topic covered : Efficient non recursive binary tree traversal algorithms

Following are three different tree traversal techniques

1. **Pre-order:** In this pre-order traversal, traverse the tree to visit in the order of left node, right node, and then root node
2. **In-order:** In this pre-order traversal, traverse the tree to visit the root node as first, then left node, and finally to visit the right node.
3. **Post-order:** In this pre-order traversal, traverse the tree to visit the left node as first, then root node, and finally to visit the right node.

These traversals are implement with the non-recursive approach (or also called as iterative approach) and these algorithms are as follows:

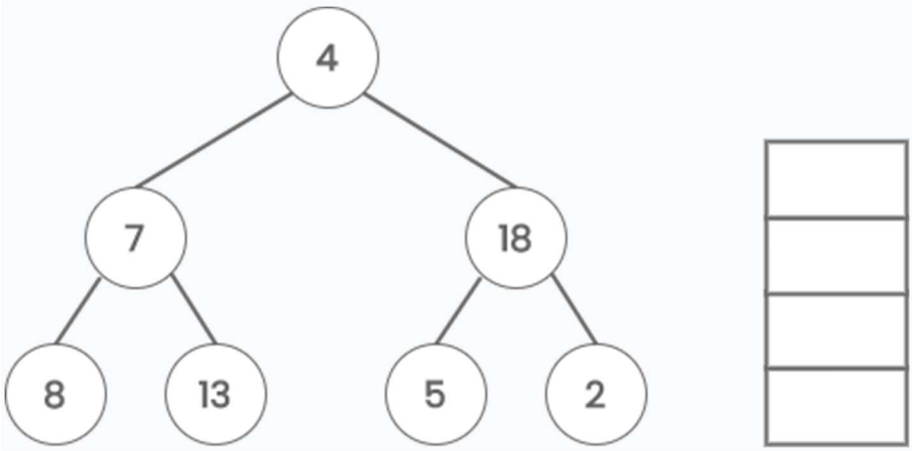
1. **Pre-order traversal with non-recursive :**

1. Start with root node and push onto stack.
2. Repeat while the stack is not empty
    1. POP the top element (PTR) from the stack and process the node.
    2. PUSH the right child of PTR onto to stack.
    3. PUSH the left child of PTR onto to stack.

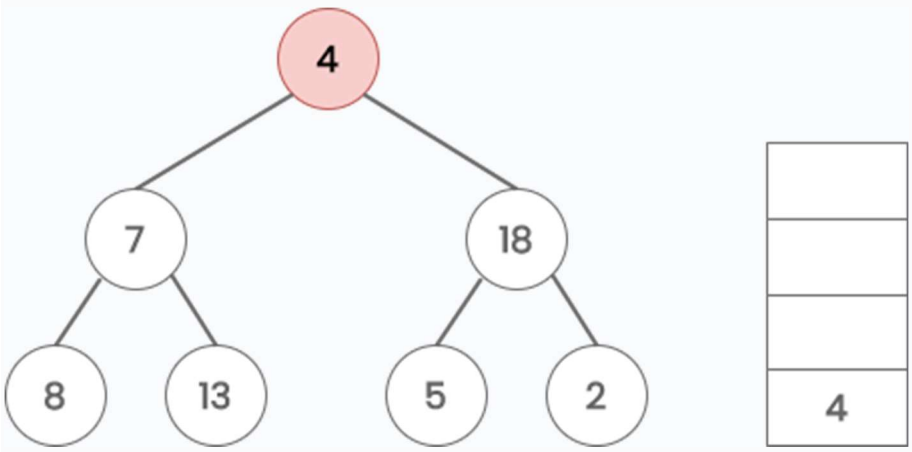
Algorithm Approach for Pre-order traversal

```
1. Set TOP = 1. STACK[1] = NULL and PTR = ROOT.
2. Repeat Steps 3 to 5 while PTR ≠ NULL:
3.    Apply PROCESS to PTR->DATA.
4.    If PTR->RIGHT ≠ NULL, then:
        Set TOP = TOP + 1, and STACK[TOP] = PTR->RIGHT.
      [End of If]
5.    If PTR->LEFT ≠ NULL, then:
         Set PTR = PTR -> LEFT.
      Else:
         Set PTR = STACK[TOP] and TOP = TOP - 1.
      [End of If]
   [End of Step 2 loop.]
6. Exit.
```

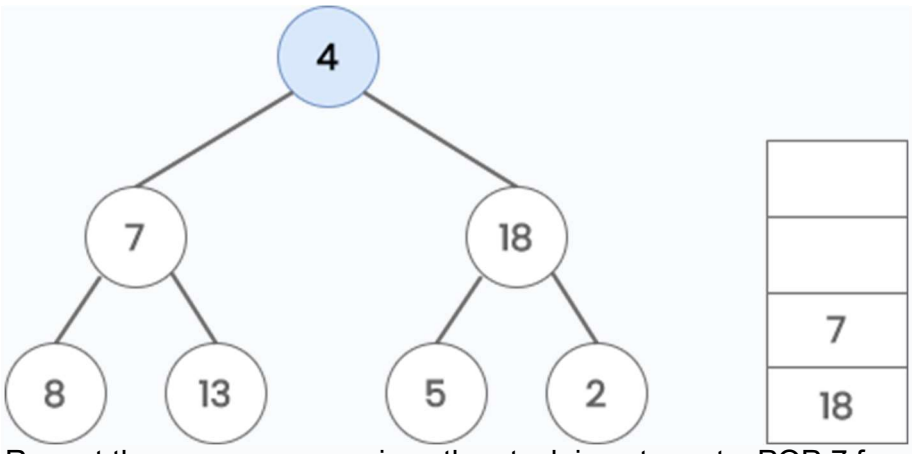
Consider the following tree.



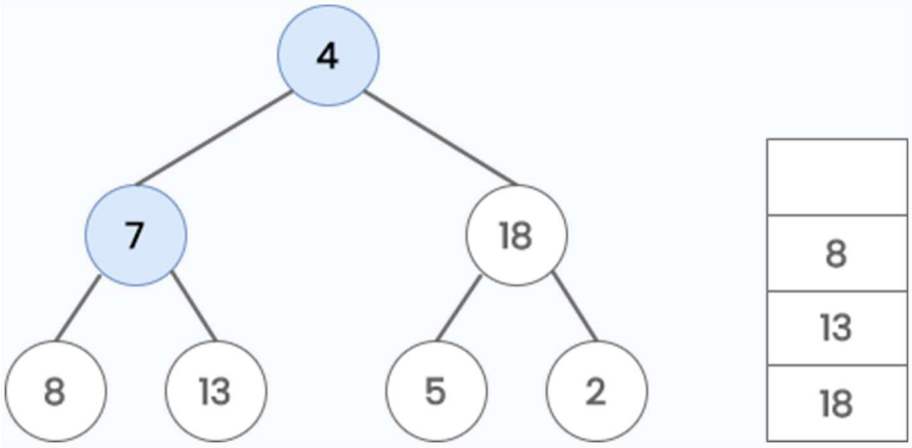
Start with node 4 and push it onto the stack.



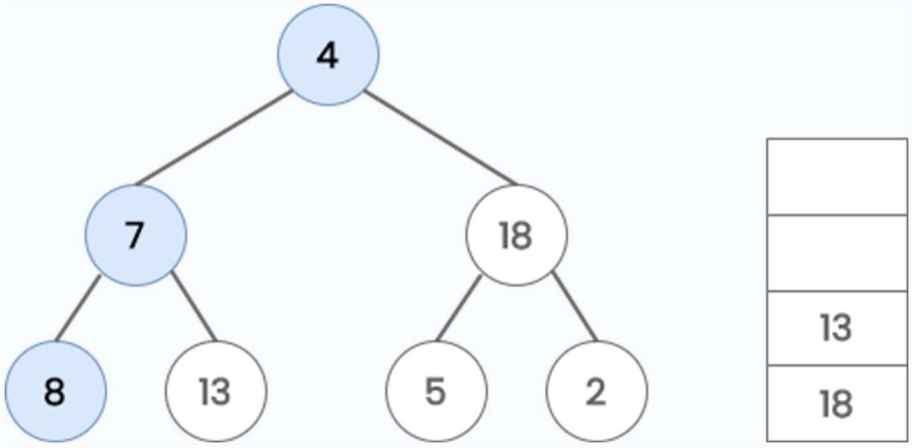
Since the stack is not empty, POP 4 from the stack, process it and PUSH its left(7) and right(18) child onto the stack.



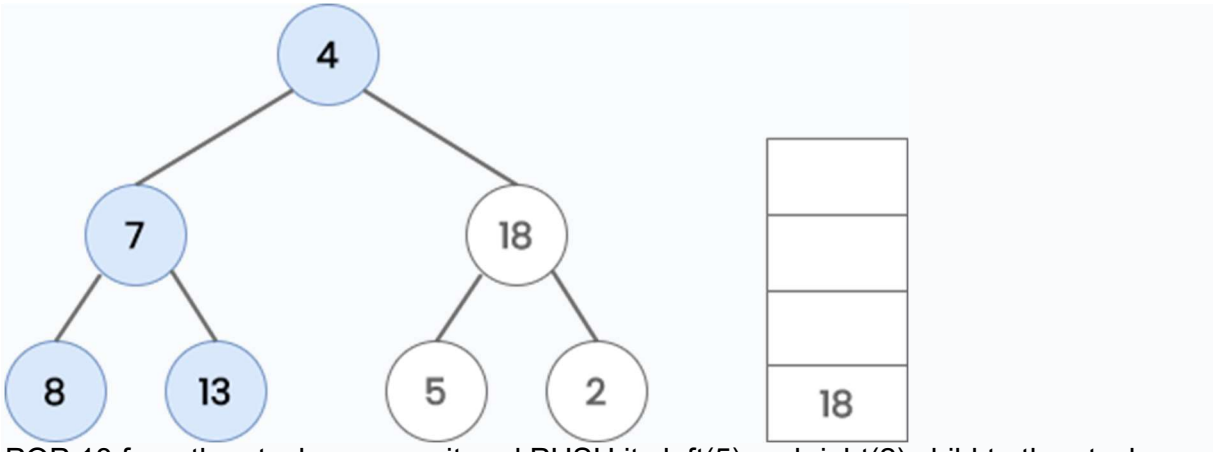
Repeat the same process since the stack is not empty. POP 7 from the stack, process it and PUSH its left(8) and right (13) child onto the stack.



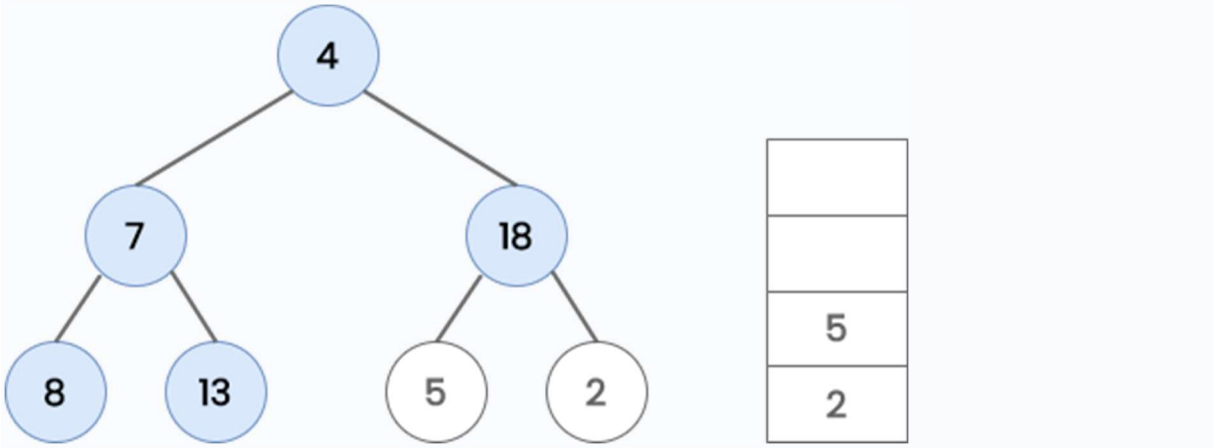
Again, POP 8 from the stack and process it. Since it has no right or left child we don't have to PUSH anything to the stack.



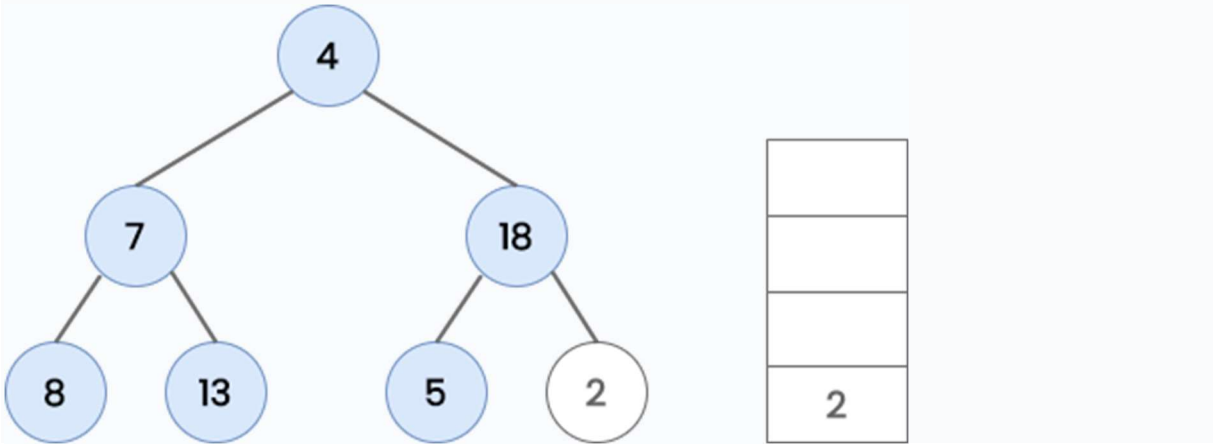
Now POP 13 from the stack and process it. We don't have to PUSH anything to the stack because it also doesn't have any subtrees.

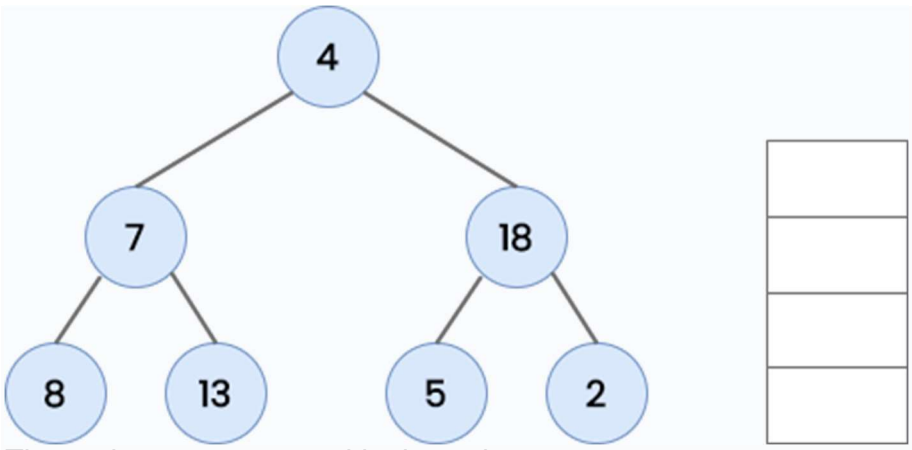


POP 18 from the stack, process it and PUSH its left(5) and right(2) child to the stack.



Similarly POP 5 and 2 from the stack one after another. Since both these nodes don't have any child, we don't have to PUSH anything onto the stack.





The nodes are processed in the order

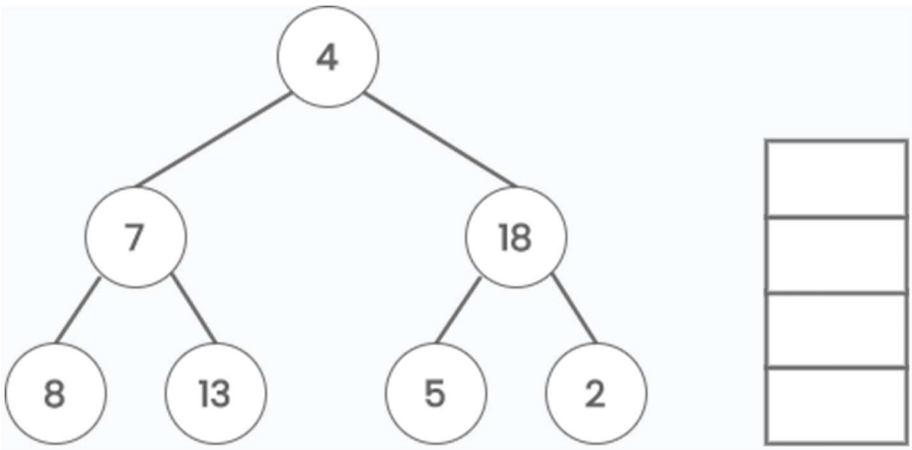
```
[ 4, 7, 8, 3, 18, 5, 2 ]
```

. This is the required preorder traversal of the given tree.

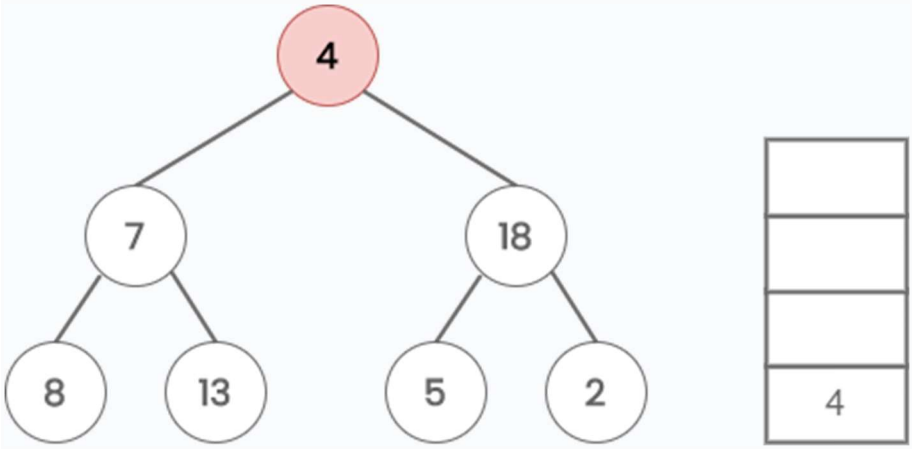
2. In-order traversal with non-recursive :

```
1. Set TOP = 1, STACK[1] = NULL and PTR = ROOT.
2. Repeat while PTR ≠ NULL:
      (a) Set TOP = TOP + 1 and STACK[TOP] = PTR.
      (b) Set PTR = PTR->LEFT.
   [End of loop.]
3. Set PTR = STACK[TOP] and TOP = TOP - 1.
4. Repeat Steps 5 to 7 while PTR ≠ NULL:
5.    Apply PROCESS to PTR->INFO.
6.    If PTR->RIGHT ≠ NULL, then:
         (a) Set PTR = PTR->RIGHT.
         (b) Go to Step 3.
      [End of If]
7. Set PTR = STACK[TOP] and TOP = TOP -1.
   [End of Step 4 loop.]
8. Exit.
```

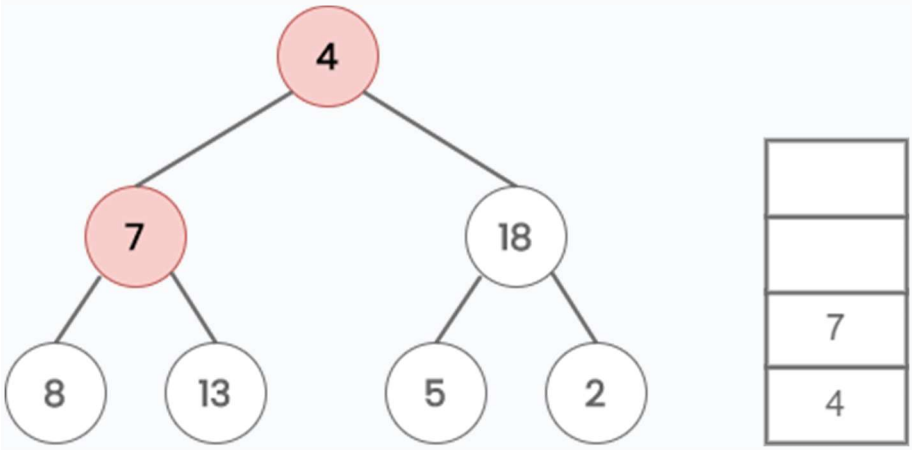
Consider the following tree.



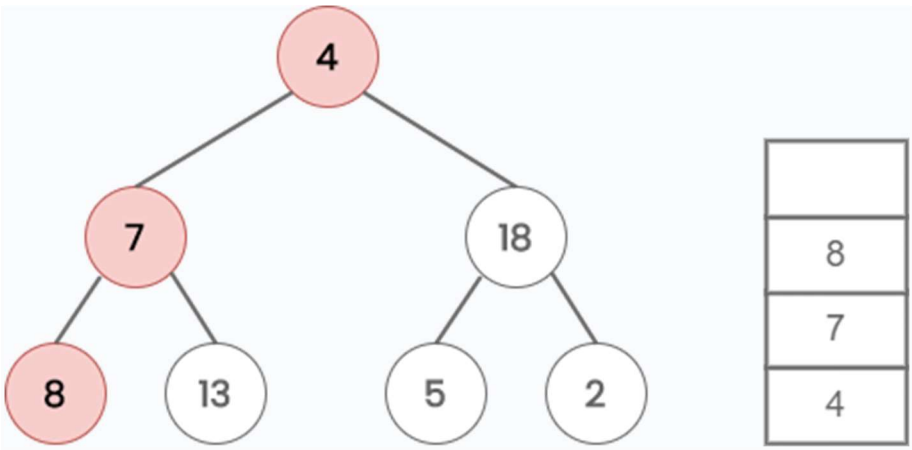
Start with node 4 and call it PTR. Since PTR is not NULL, PUSH it onto the stack.



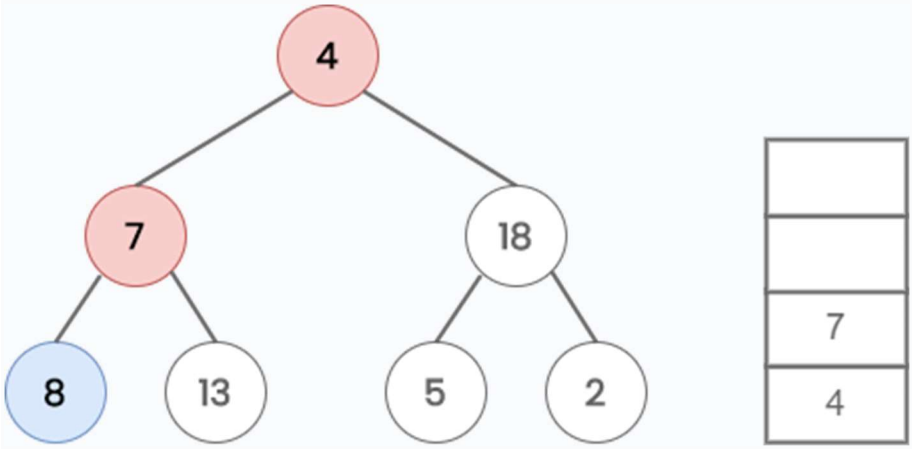
Move to the left of node 4. Now PTR is node 7, which is not NULL. So PUSH it onto the stack.



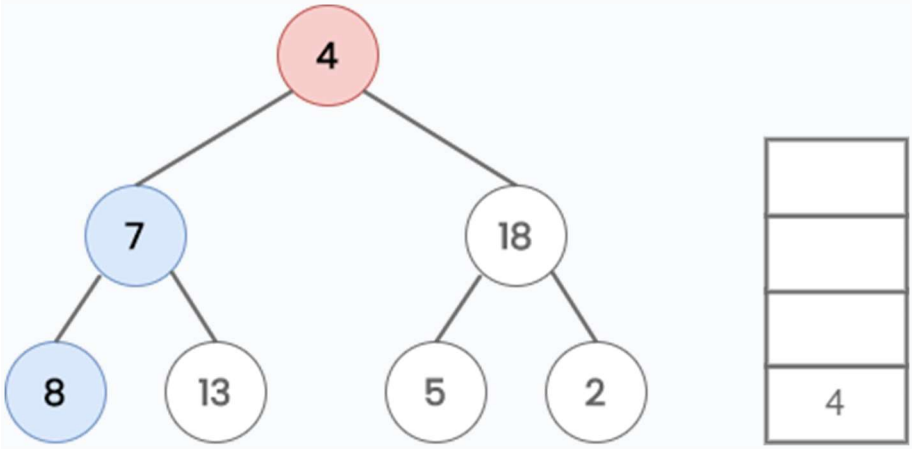
Again, move to the left of node 7. Now PTR is node 8, which is not NULL. So PUSH it onto the stack.



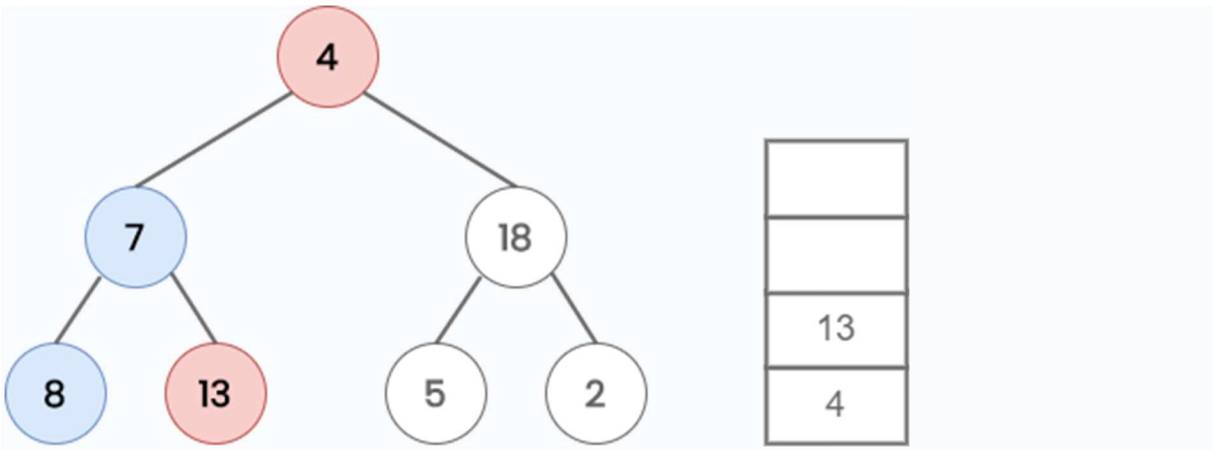
When we move again to the left of node 8, PTR becomes NULL. So POP 8 from the stack. Process PTR (8).



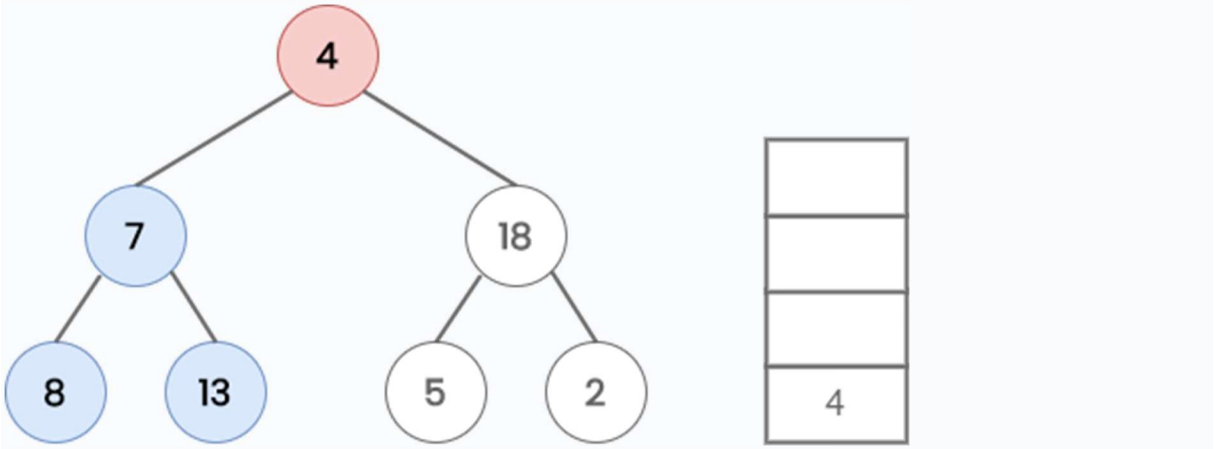
Move to the right child of PTR(8), which is NULL. So POP 7 from the stack and process it. Now PTR points to 7.



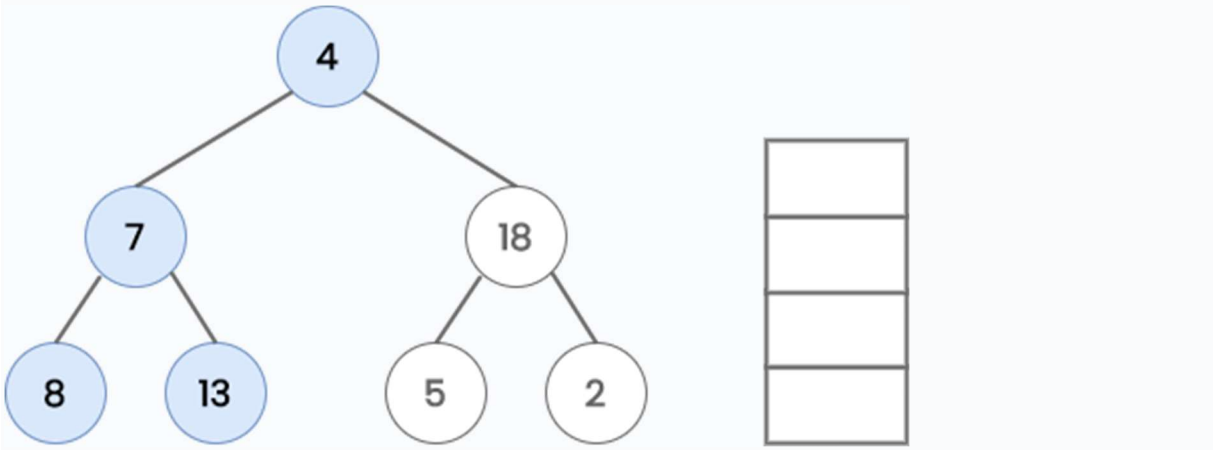
Move to the right child(13) of PTR and PUSH it onto the stack.



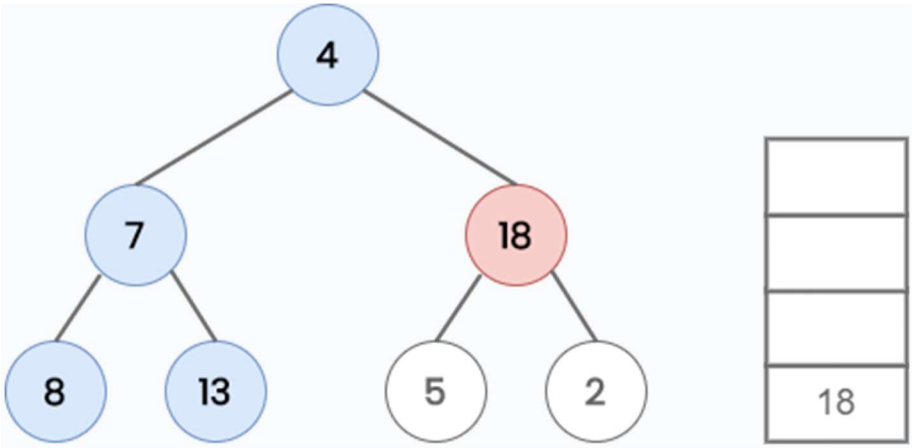
Move to the left of node 13, which is NULL. So POP 13 from the stack and process it.



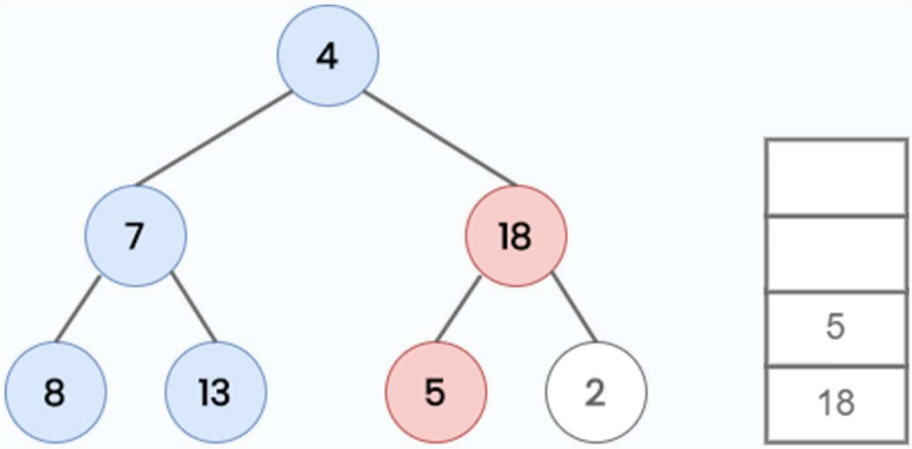
Since node 13 don’t have any right child, POP 4 from the stack and process it. Now PTR points to node 4.



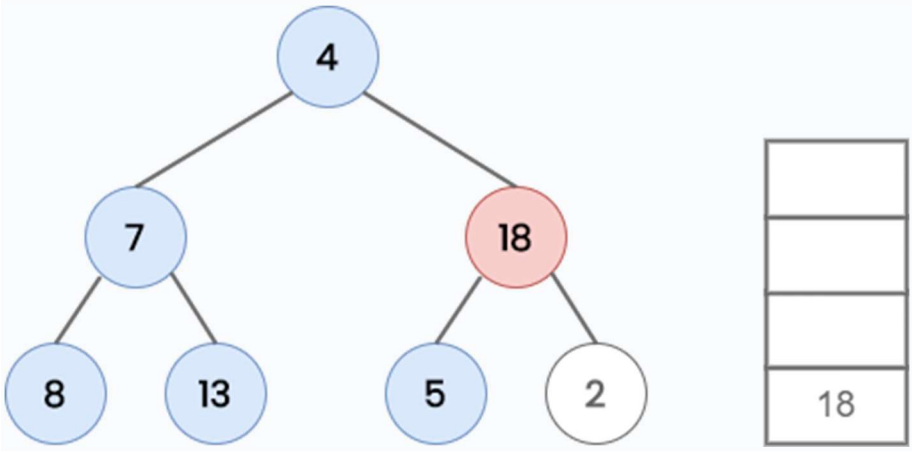
Move to the right of node 4 and put it on to the stack. Now PTR points to node 18.



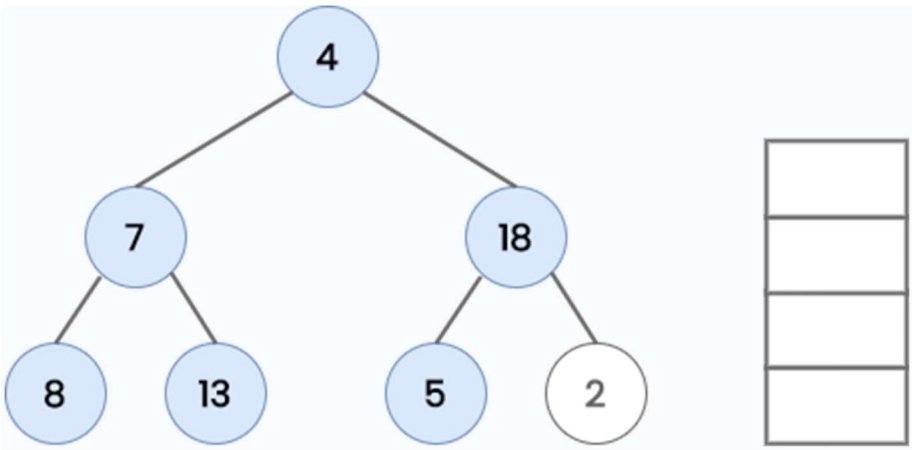
Move to the left(5) child of 18 and put it onto the stack. Now PTR points to node 5.



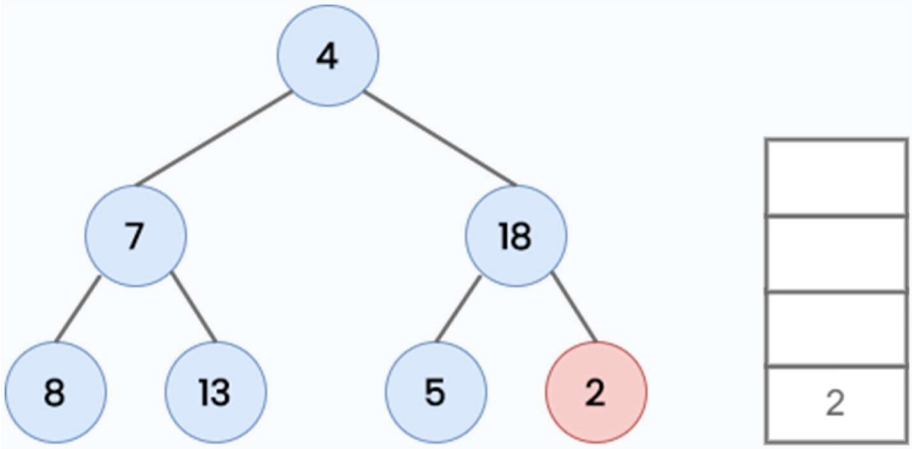
Move to the left of node 5, which is NULL. So POP 5 from the stack and process it.



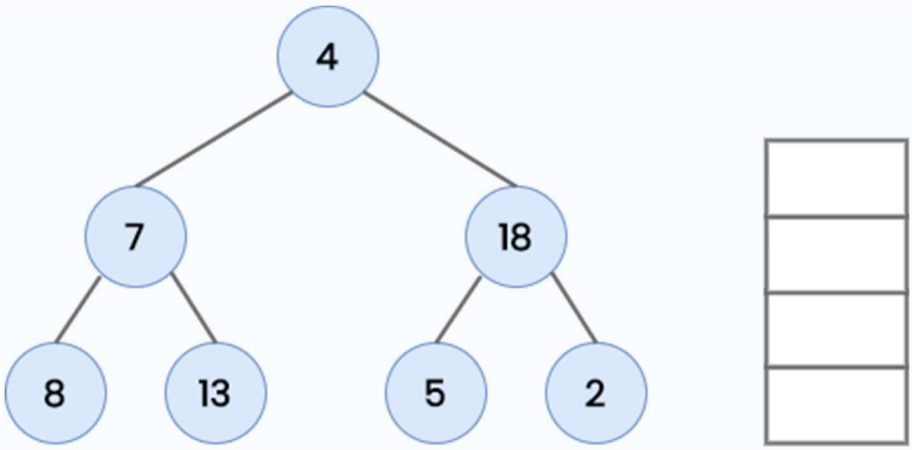
Now, move to the right of node 5, which is NULL. So POP 18 from the stack and process it.



Move to the right(2) of node 18 and PUSH it on to the stack.



Since node 2 has left child, POP 2 from the stack and process it. Now the stack is empty and node 2 has no right child. So stop traversing.



The nodes are processed in the order

```
[ 8, 7, 13, 4, 5, 18, 2 ]
```

This is the required in order traversal of the given tree.

3. Pre-order traversal with non-recursive :

```
1. Set TOP = 1. STACK[1] = NULL and PTR = ROOT.
2. Repeat Steps 3 to 5 while PTR ≠ NULL:
3.    Set TOP = TOP + 1 and STACK[TOP] = PTR.
4.    If PTR->RIGHT ≠ NULL. then: [Push on STACK. ]
         Set TOP = TOP + 1 and STACK[TOP] = PTR->-RIGHT.
      [End of If structure.]
5.    Set PTR = PTR->LEFT. [Updates pointer PTR.]
   [End of Step 2 loop.]
6. Set PTR = STACK[TOP] and TOP = TOP - 1.
7. Repeat while PTR > 0:
      (a) Apply PROCESS to PTR->INFO.
      (b) Set PTR = STACK[TOP] and TOP = TOP - 1.
   [End of loop.]
8. If PTR < 0, then:
      (a) Set PTR = -PTR.
      (b) Go to Step 2.
   [End of If structure]
9. Exit.
```