



DATA STRUCTURES AND ITS APPLICATIONS

Shylaja S S & Kusuma K V
Department of Computer Science
& Engineering



DATA STRUCTURES AND ITS APPLICATIONS

Iterative BST Traversal

Shylaja S S

Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
s = emptyStack
current = root
do {
    while(current != null)
    {
        /* Travel down left branches as far as possible
           saving pointers to nodes passed in the stack*/
        push(s, current)
        current = current->left
    } //At this point, the left subtree is empty
    poppedNode = pop(s)
    print poppedNode ->info      //visit the node
    current = poppedNode ->right //traverse right subtree
} while(!isEmpty(s) or current != null)
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
s = emptyStack
```



```
current = root
```



```
do {
```

```
    while(current != null)
```



```
{
```

```
    push(s, current)
```



```
    current = current->left
```

```
}
```

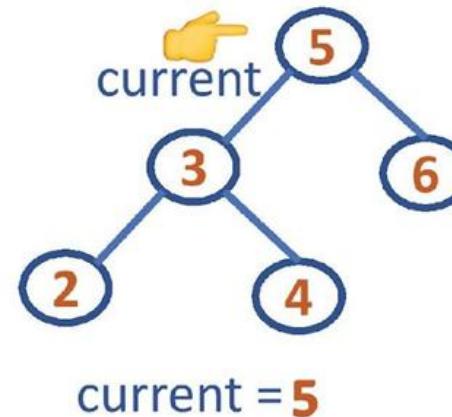
```
poppedNode = pop(s)
```

```
print poppedNode ->info
```

```
current = poppedNode ->right
```

```
} while(!isEmpty(s) or current != null)
```

Note: Stack has Address of Nodes Pushed In



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
s = emptyStack
```

```
current = root
```

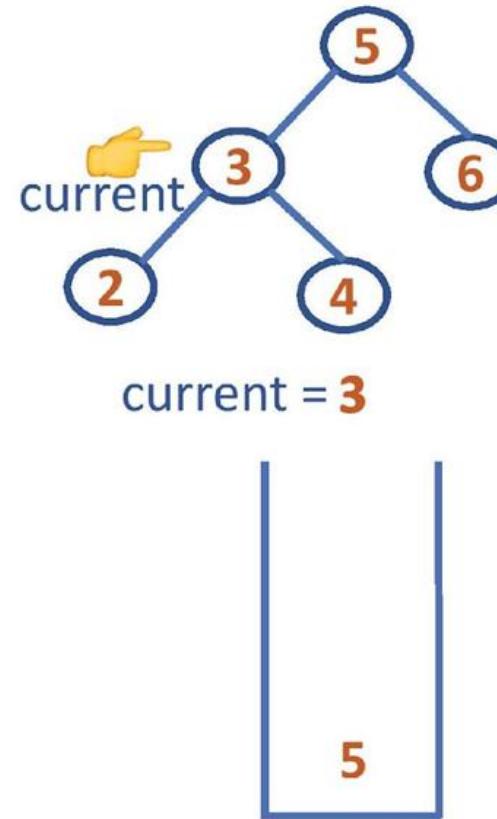
```
do {
```

```
    while(current != null)      ↗
```

```
    {  
        push(s, current)      ↗  
        current = current->left      ↗  
    }
```

```
    poppedNode = pop(s)  
    print poppedNode ->info  
    current = poppedNode ->right
```

```
} while(!isEmpty(s) or current != null)
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
    s = emptyStack
```

```
    current = root
```

```
    do {
```

```
        while(current != null)      ↗
```

```
        {
```

```
            push(s, current)      ↗
```



```
            current      ↗
```



```
            current = 2
```

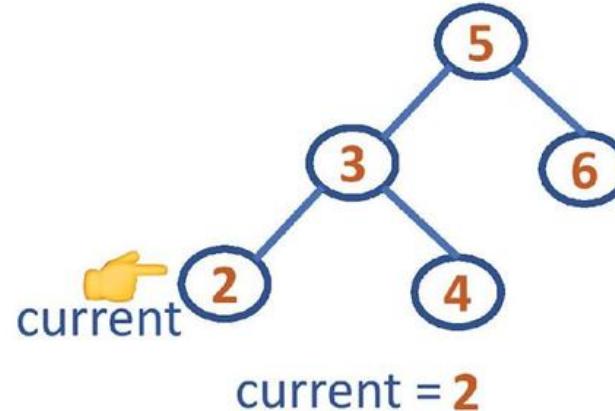
```
        }
```

```
        poppedNode = pop(s)
```

```
        print poppedNode ->info
```

```
        current = poppedNode ->right
```

```
} while(!isEmpty(s) or current != null)
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
s = emptyStack
```

```
current = root
```

```
do {
```

```
    while(current != null) 
```

```
{
```

```
    push(s, current)
```

```
    current = current->left 
```

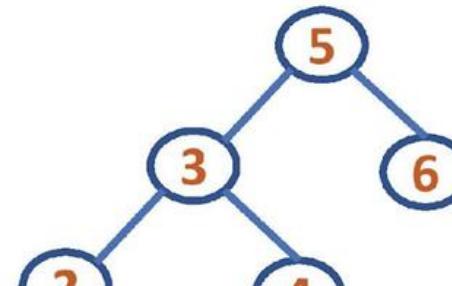
```
}
```

```
    poppedNode = pop(s)
```

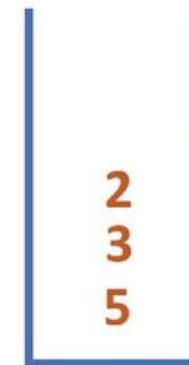
```
    print poppedNode ->info
```

```
    current = poppedNode ->right
```

```
} while(!isEmpty(s) or current != null)
```



current = N



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
s = emptyStack
```

```
current = root
```

```
do {
```

```
    while(current != null)
```

```
{
```

```
    push(s, current)
```

```
    current = current->left
```

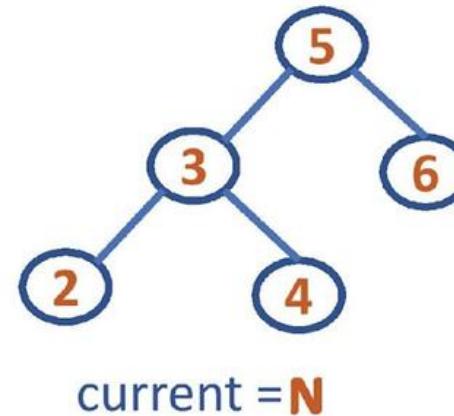
```
}
```

```
poppedNode = pop(s) ➔ poppedNode =
```

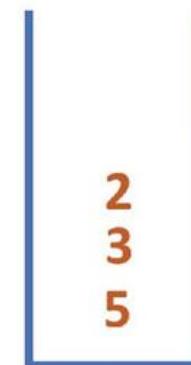
```
print poppedNode ->info ➔
```

```
current = poppedNode ->right ➔
```

```
} while(!isEmpty(s) or current != null) ➔
```



current = N



Inorder Traversal:

2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
    s = emptyStack
```

```
    current = root
```

```
    do { ➔
```

```
        while(current != null) ➔
```

```
        {
```

```
            push(s, current)
```

```
            current = current->left
```

```
        }
```

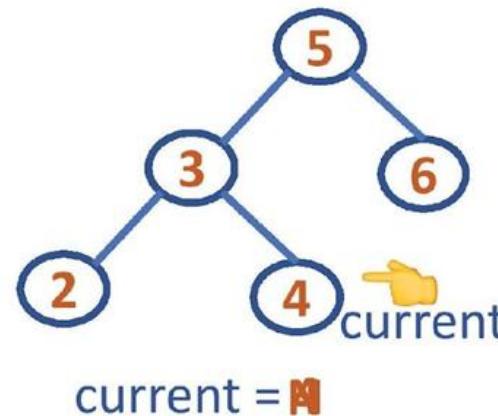
```
        poppedNode = pop(s) ➔
```

```
        poppedNode = 2
```

```
        print poppedNode ->info ➔
```

```
        current = poppedNode ->right ➔
```

```
    } while(!isEmpty(s) or current != null) ➔
```



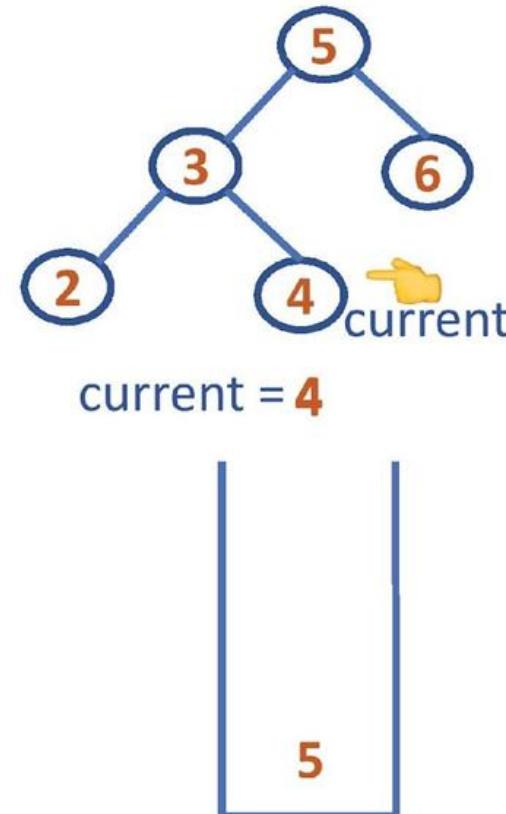
Inorder Traversal:

2 3

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
s = emptyStack
current = root
do {    ↗
        while(current != null) ↗
        {
            push(s, current) ↗
            current = current->left
        }
        poppedNode = pop(s)
        print poppedNode ->info
        current = poppedNode ->right
    } while(!isEmpty(s) or current != null)
```



Inorder Traversal:

23

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
s = emptyStack
```

```
current = root
```

```
do {
```

```
    while(current != null) ➔
```

```
{
```

```
    push(s, current)
```

```
    current = current->left ➔
```

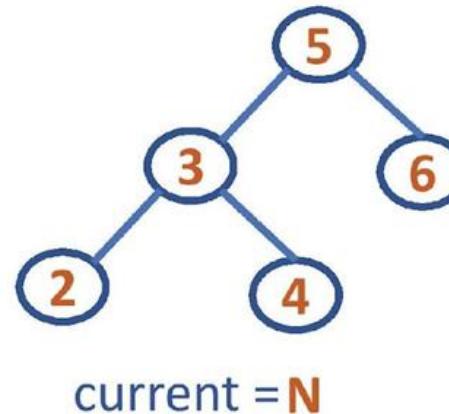
```
}
```

```
    poppedNode = pop(s) ➔
```

```
    print poppedNode ->info ➔
```

```
    current = poppedNode ->right ➔
```

```
} while(!isEmpty(s) or current != null) ➔
```



Inorder Traversal:

2 3 4

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
s = emptyStack
```

```
current = root
```

```
do { ➔
```

```
    while(current != null) ➔
```

```
{
```

```
    push(s, current)
```

```
    current = current->left
```

```
}
```

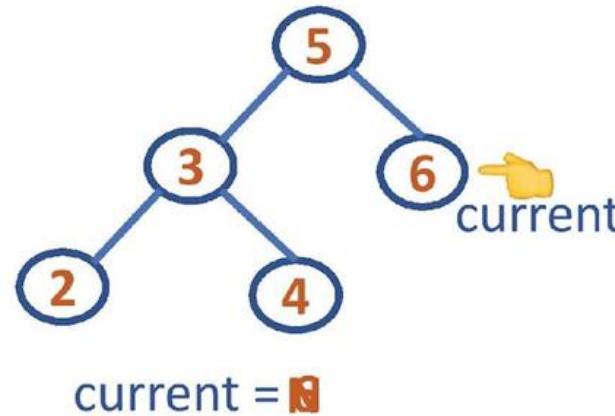
```
    poppedNode = pop(s) ➔
```

```
    poppedNode = 4
```

```
    print poppedNode ->info ➔
```

```
    current = poppedNode ->right ➔
```

```
} while(!isEmpty(s) or current != null) ➔
```



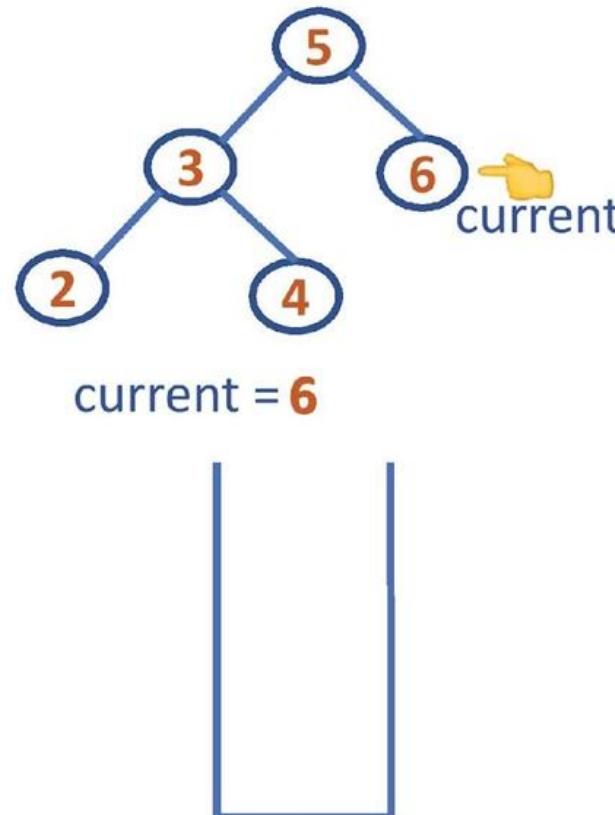
Inorder Traversal:

2 3 4 5

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
s = emptyStack
current = root
do { ➔
    while(current != null) ➔
    {
        push(s, current) ➔
        current = current->left
    }
    poppedNode = pop(s)
    print poppedNode ->info
    current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```



Inorder Traversal:

2 3 4 5

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

```
iterativeInorder(root)
```

```
s = emptyStack
```

```
current = root
```

```
do {
```

```
    while(current != null) ➔
```

```
{
```

```
    push(s, current)
```

```
    current = current->left ➔
```

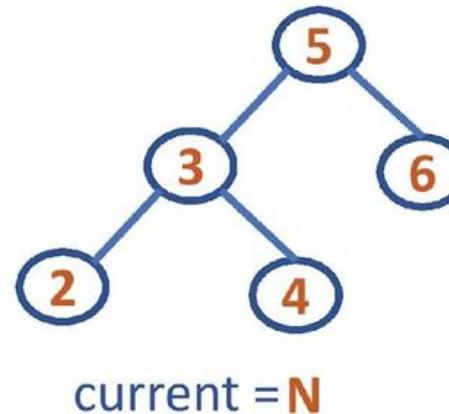
```
}
```

```
    poppedNode = pop(s) ➔ poppedNode =
```

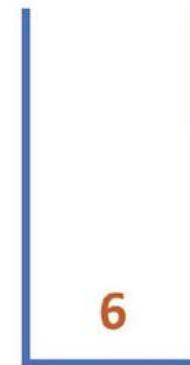
```
    print poppedNode ->info ➔
```

```
    current = poppedNode ->right ➔
```

```
} while(!isEmpty(s) or current != null) ➔
```



current = N



Inorder Traversal:

2 3 4 5 6

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
current=root
if (current == null)
    return
s = emptyStack
push(s, current)
while(!isEmpty(s)) {
    current = pop(s)
    print current->info
    //right child is pushed first so that left is processed first
    if(current->right !=NULL)
        push(s, current->right)
    if(current->left !=NULL)
        push(s, current->left)
}
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
    current=root
```



```
    if (current == null)
```



```
        return
```

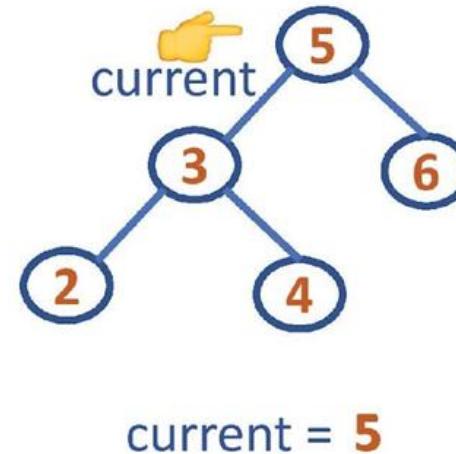
```
    s = emptyStack
```



```
    push(s, current)
```



```
    ...
```



Note: Stack has Address
of Nodes Pushed In



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
...  
...  
...  
...
```

```
while (!isEmpty(s))
```



```
{
```

```
    current = pop(s)
```



```
    print current ->info
```



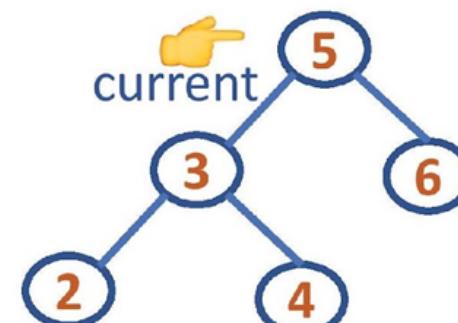
```
    if(current->right != null)
```

```
        push(s, current->right)
```

```
    if(current->left != null)
```

```
        push(s, current->left)
```

```
}
```



current = 5



Preorder Traversal:
5



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
while (!isEmpty(s))
```

```
{
```

```
    current = pop(s)
```

```
    print current ->info
```

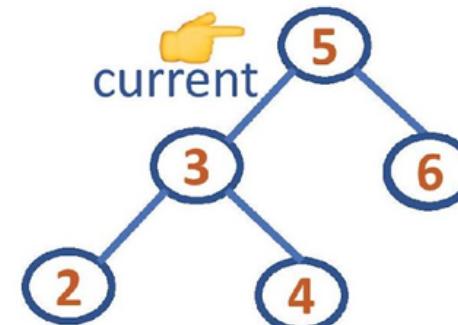
```
    if(current->right != null) 
```

```
        push(s, current->right) 
```

```
    if(current->left != null)
```

```
        push(s, current->left)
```

```
}
```



current = 5

current->right = 6



Preorder Traversal:
5

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
while (!isEmpty(s))
```

```
{
```

```
    current = pop(s)
```

```
    print current ->info
```

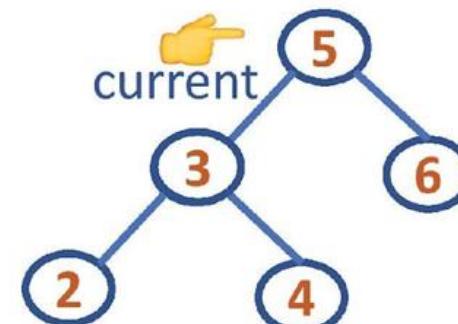
```
    if(current->right != null)
```

```
        push(s, current->right)
```

```
    if(current->left != null) 
```

```
        push(s, current->left) 
```

```
}
```



current = 5

current->left = 3



Preorder Traversal:
5



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

iterativePreorder(root)

11

```
while (!isEmpty(s))
```

{

current = pop(s)

print current ->info

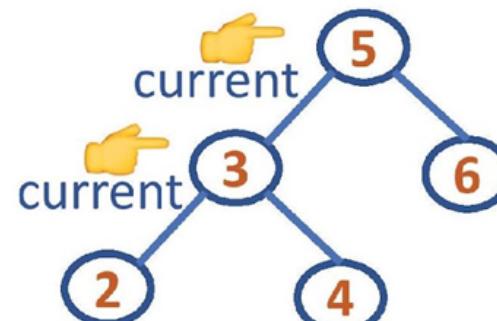
fo

```
if(current->right != null)
```

`push(s, current->right)`

```
if(current->left != null)
```

`push(s, current->left)`



current = 3



Preorder Traversal:

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
while (!isEmpty(s))
```

```
{
```

```
    current = pop(s)
```

```
    print current ->info
```

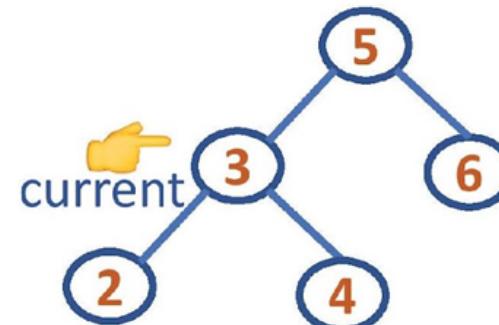
```
    if(current->right != null) ➔
```

```
        push(s, current->right) ➔
```

```
    if(current->left != null)
```

```
        push(s, current->left)
```

```
}
```



current = 3

current->right = 4



Preorder Traversal:
5 3



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
...  
while (!isEmpty(s))
```

```
{
```

```
    current = pop(s)
```

```
    print current ->info
```

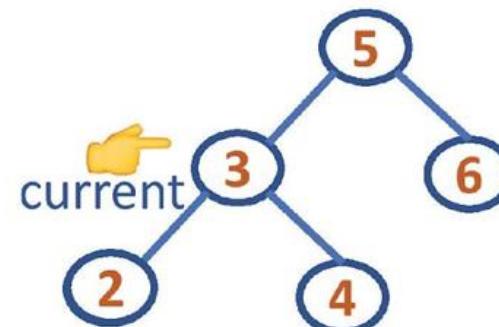
```
    if(current->right != null)
```

```
        push(s, current->right)
```

```
    if(current->left != null) 
```

```
        push(s, current->left) 
```

```
}
```



current = 3

current->left = 2



Preorder Traversal:
5 3



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

iterativePreorder(root)

20

```
while (!isEmpty(s))
```

{

current = pop(s) 

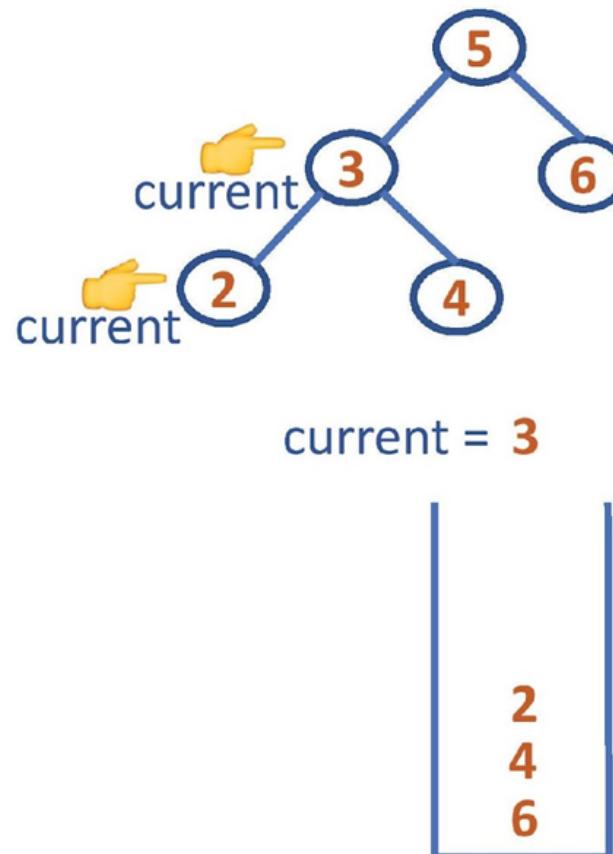
print current ->info

```
if(current->right != null)
```

`push(s, current->right)`

```
if(current->left != null)
```

`push(s, current->left)`



Preorder Traversal:

532

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
while (!isEmpty(s))
```

```
{
```

```
    current = pop(s)
```

```
    print current ->info
```

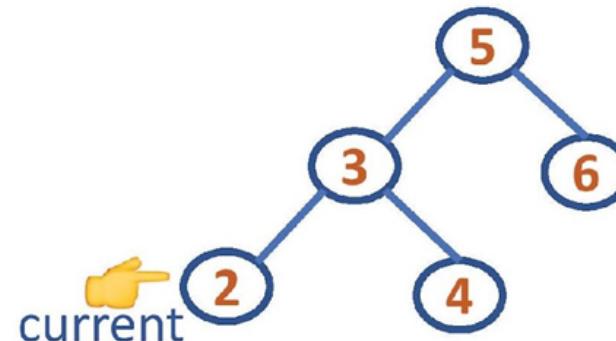
```
    if(current->right != null) ➔
```

```
        push(s, current->right)
```

```
    if(current->left != null)
```

```
        push(s, current->left)
```

```
}
```



current = 2
current->right = N



Preorder Traversal:
5 3 2



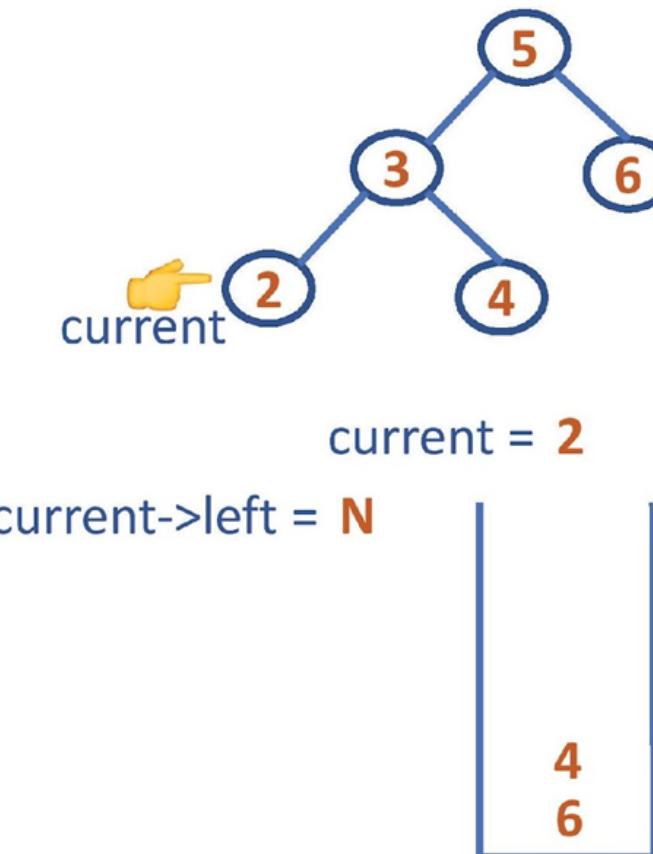
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
while (!isEmpty(s))
```

```
{  
    current = pop(s)  
    print current ->info  
    if(current->right != null)  
        push(s, current->right)  
    if(current->left != null)   
        push(s, current->left)  
}
```



Preorder Traversal:
5 3 2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
while (!isEmpty(s))
```

```
{
```

```
    current = pop(s)
```

```
    print current ->info
```

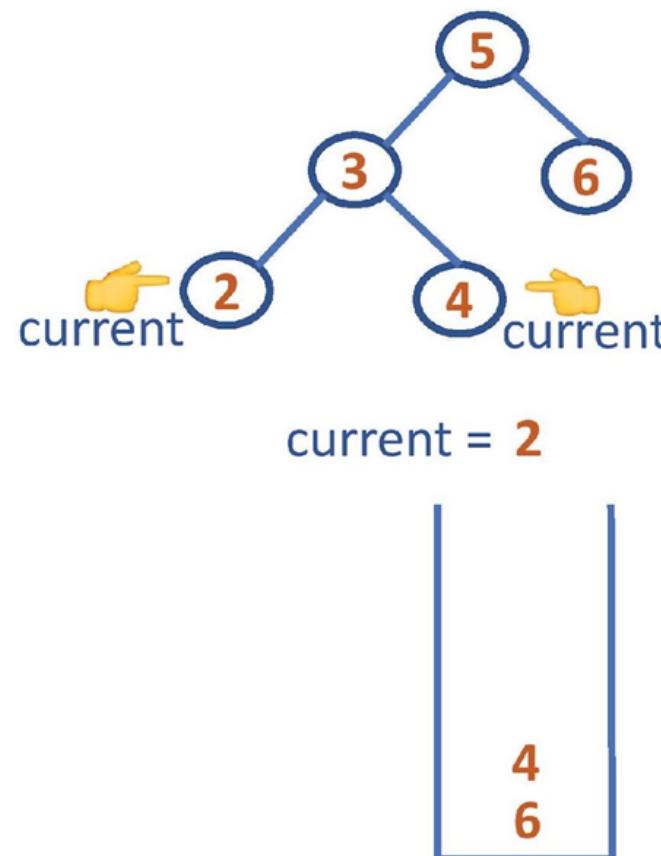
```
    if(current->right != null)
```

```
        push(s, current->right)
```

```
    if(current->left != null)
```

```
        push(s, current->left)
```

```
}
```



Preorder Traversal:
5 3 2 4

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
...  
while (!isEmpty(s))
```

```
{
```

```
    current = pop(s)
```

```
    print current ->info
```

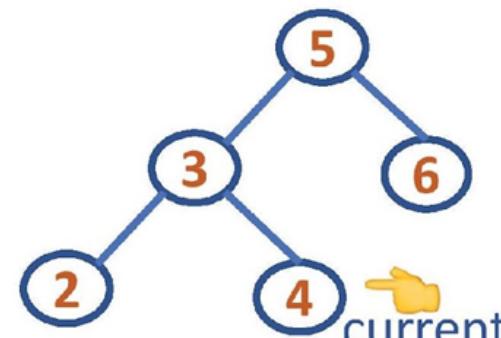
```
    if(current->right != null) ➔
```

```
        push(s, current->right)
```

```
    if(current->left != null) ➔
```

```
        push(s, current->left)
```

```
}
```



current = 4

current->right = N

current->left = N



Preorder Traversal:
5 3 2 4

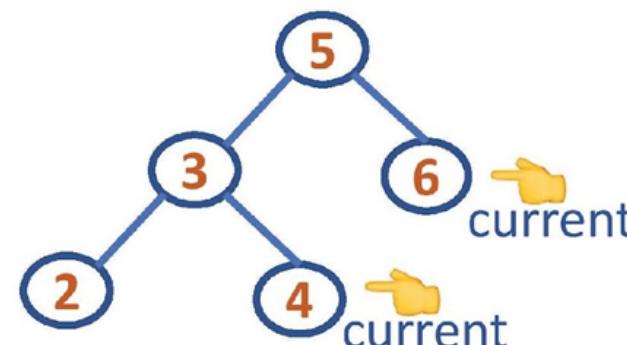


DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
```

```
  ...  
  while (!isEmpty(s)) {  
    current = pop(s)  
    print current ->info  
    if(current->right != null) push(s, current->right)  
    if(current->left != null) push(s, current->left)  
  }
```



current = 4

current->right = N
current->left = N



Preorder Traversal:
5 3 2 4 6



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal



```
iterativePostorder(root)
s1 = emptyStack ; s2 = emptyStack ; push(s1, root)
while(!isEmpty(s1)) {
    current = pop(s1)
    push(s2,current)
    if(current->left !=NULL)
        push(s1, current->left)
    if(current->right !=NULL)
        push(s1, current->right)
}
while(!isEmpty(s2)) { //Print all the elements of stack2
    current = pop(s2)
    print current->info
}
```

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

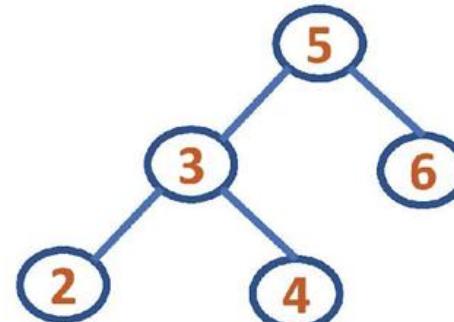
iterativePostorder(root)

s1 = emptyStack ➔

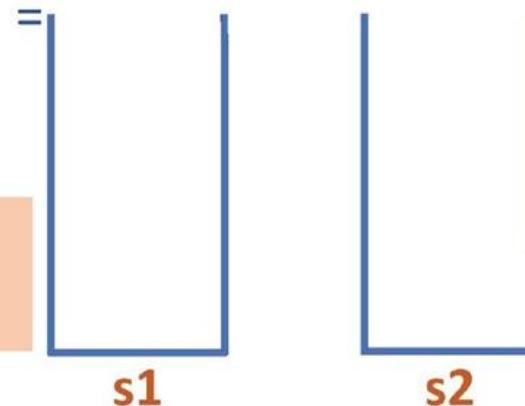
s2 = emptyStack ➔

push(s1, root) ➔

⋮
⋮



root 5



Note: Stacks have Address
of Nodes Pushed In



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))
```

```
{
```

```
    current = pop(s1)
```

```
    push(s2, current)
```

```
    if(current->left !=NULL)
```

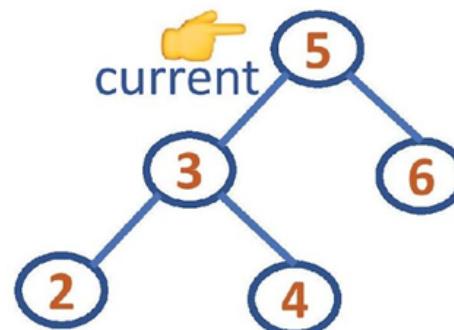
```
        push(s1, current->left)
```

```
    if(current->right !=NULL)
```

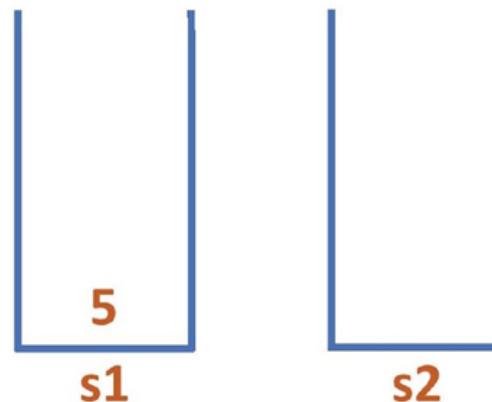
```
        push(s1, current->right)
```

```
}
```

```
  :::::
```



```
current =
```

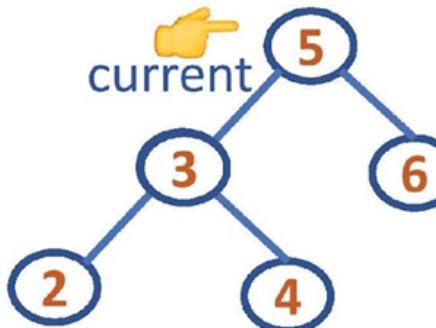


DATA STRUCTURES AND ITS APPLICATIONS

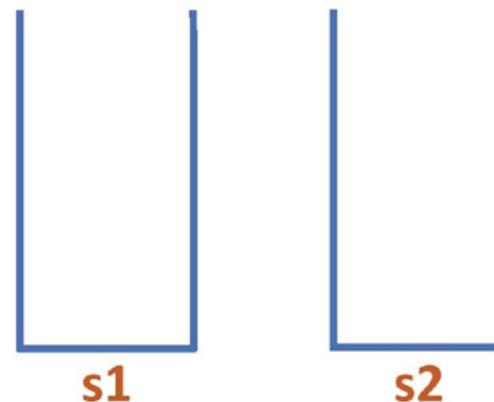
Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current)  ↗  
    if(current->left !=NULL) ↗  
      push(s1, current->left) ↗  
    if(current->right !=NULL)  
      push(s1, current->right)  
  }  
  :::::  
          current->left = 3
```



current = 5

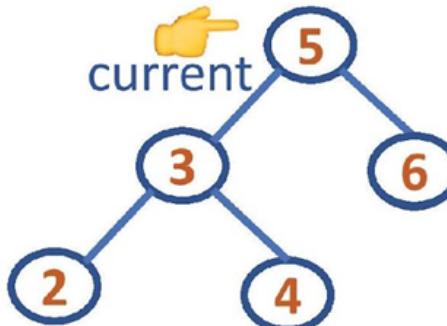


DATA STRUCTURES AND ITS APPLICATIONS

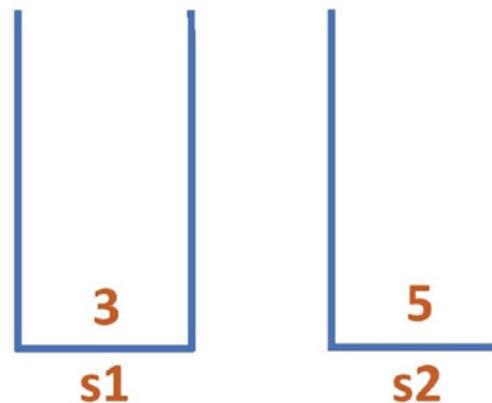
Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current)  
    if(current->left !=NULL)  
      push(s1, current->left)  
    if(current->right !=NULL)    
      push(s1, current->right)  
  }  
  :::::
```



current->right = 6



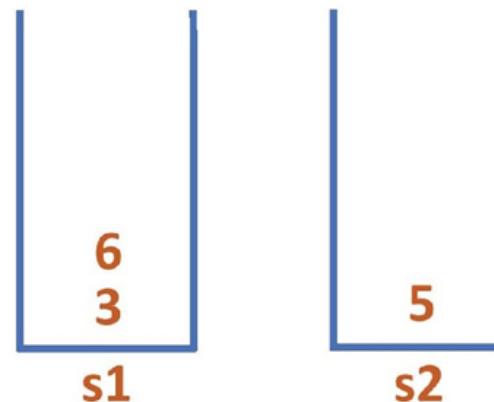
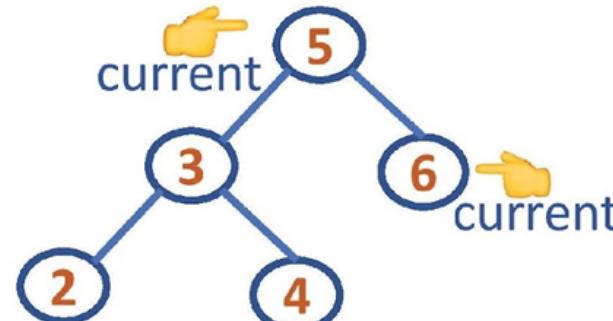
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))
```

```
  {  
    current = pop(s1)  
    push(s2,current)  
    if(current->left !=NULL)  
      push(s1, current->left)  
    if(current->right !=NULL)  
      push(s1, current->right)  
  }  
  :::::
```



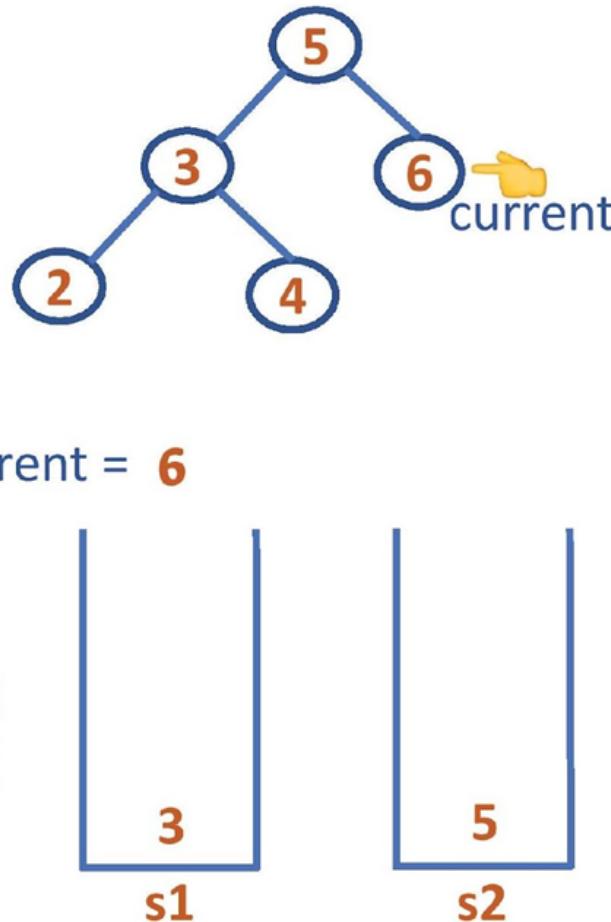
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current)  ↗  
    if(current->left !=NULL) ↗  
      push(s1, current->left)  
    if(current->right !=NULL) ↗  
      push(s1, current->right)  
  }  
  :::::
```

current->left = N
current->right = N



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))
```

```
{
```

```
    current = pop(s1)
```



```
    push(s2,current)
```

```
    if(current->left !=NULL)
```

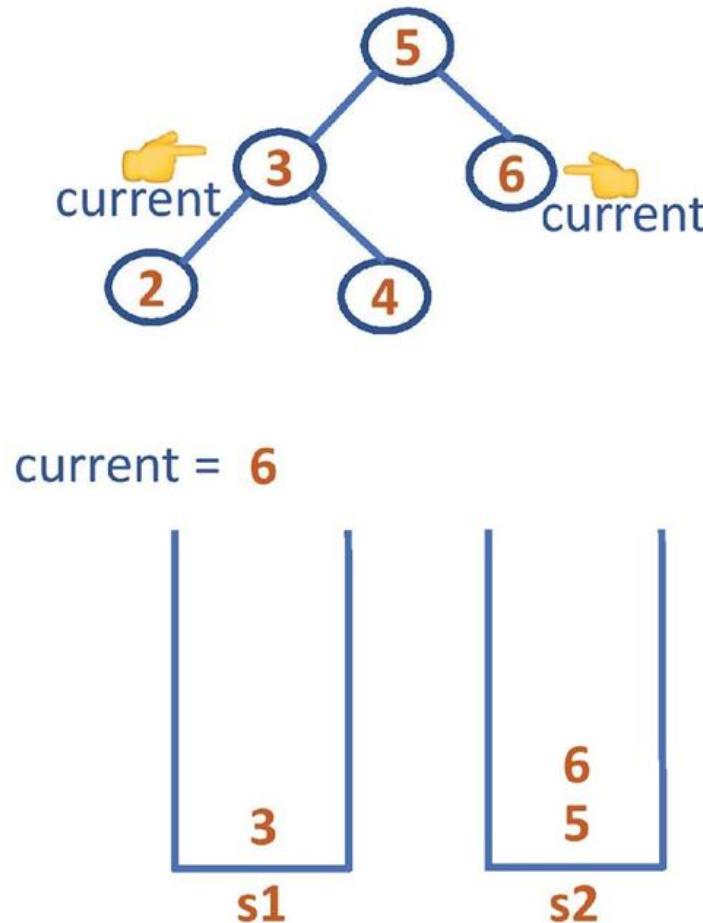
```
        push(s1, current->left)
```

```
    if(current->right !=NULL)
```

```
        push(s1, current->right)
```

```
}
```

```
  :::::
```

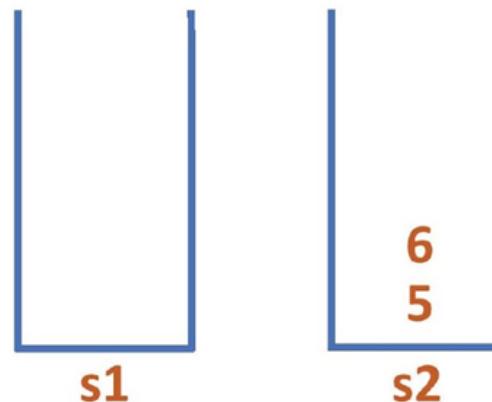
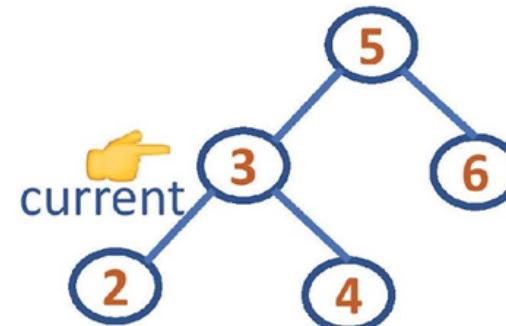


DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current)  ↗  
    if(current->left !=NULL)  
      push(s1, current->left)  
    if(current->right !=NULL)  
      push(s1, current->right)  
  }  
  :::::
```

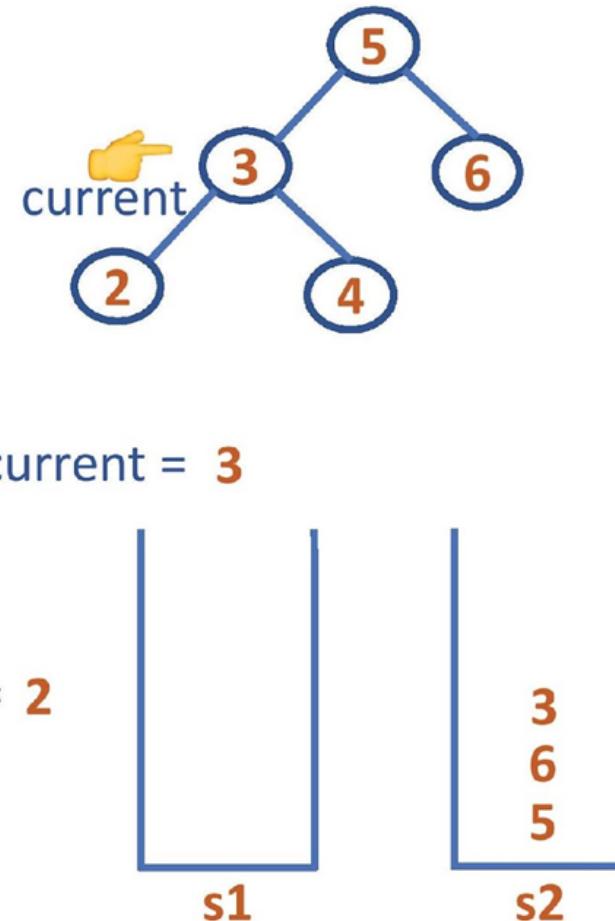


DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current)  
    if(current->left !=NULL)    
      push(s1, current->left)    
    if(current->right !=NULL)  
      push(s1, current->right)  
  }  
  :::::
```

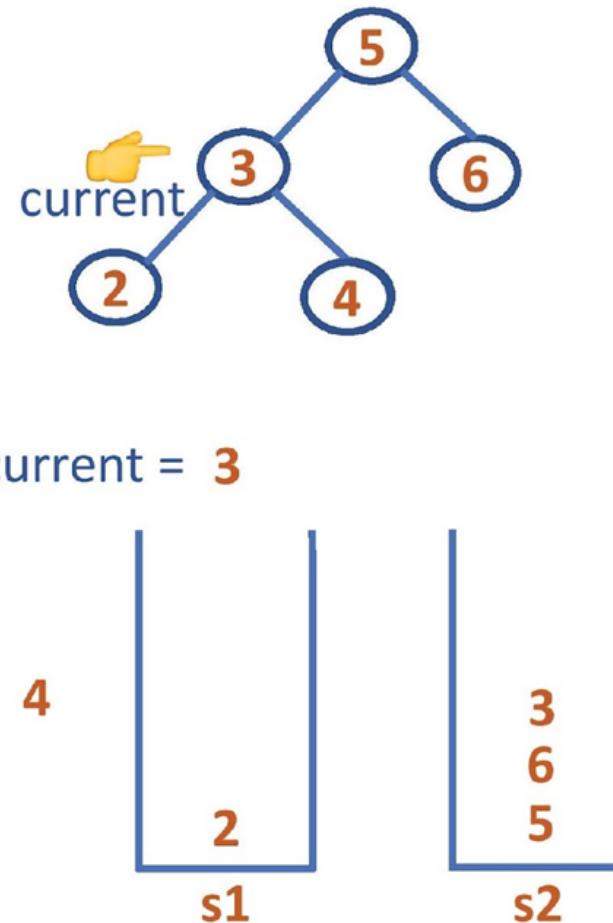


DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current)  
    if(current->left !=NULL)  
      push(s1, current->left)  
    if(current->right !=NULL)   
      push(s1, current->right)  
  }  
  :::::
```



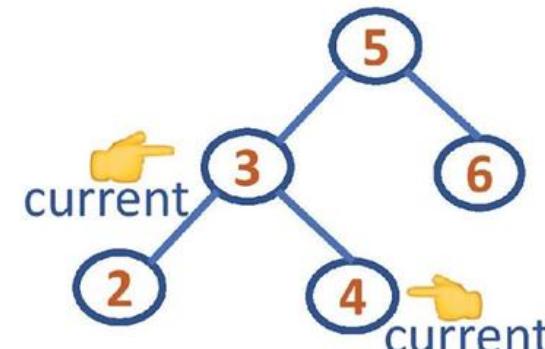
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

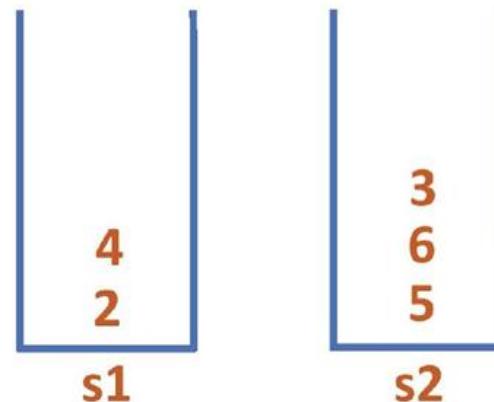
```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))
```

```
{  
    current = pop(s1)  
    push(s2,current)  
    if(current->left !=NULL)  
        push(s1, current->left)  
    if(current->right !=NULL)  
        push(s1, current->right)  
}  
  :::::
```



current = 3

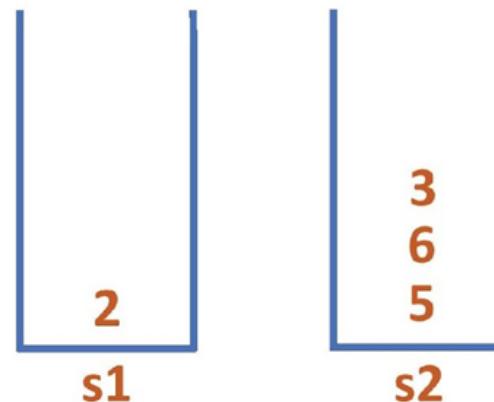
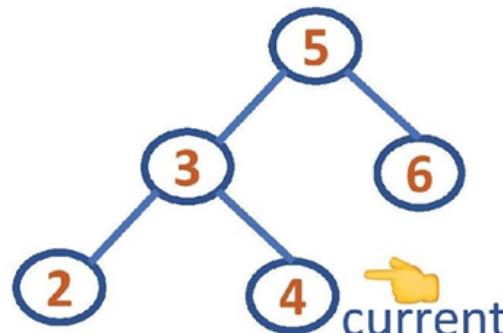


DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current) ➔  
    if(current->left !=NULL) ➔  
      push(s1, current->left)  
    if(current->right !=NULL) ➔  
      push(s1, current->right)  
  }  
  :::::  
          current->left = N  
          current->right = N
```



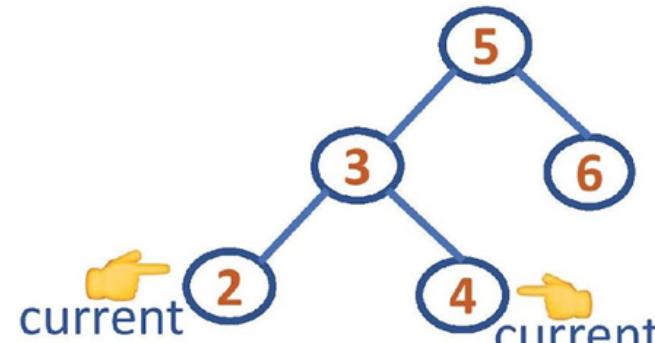
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

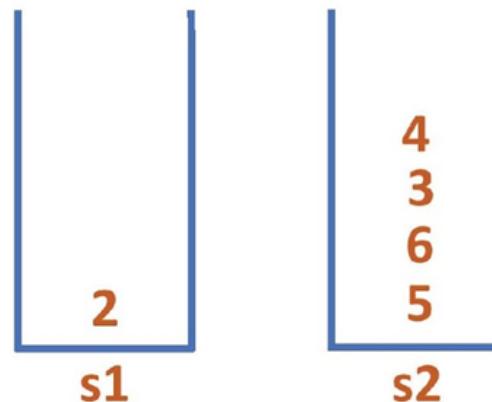
```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))
```

```
{  
    current = pop(s1)  
    push(s2,current)  
    if(current->left !=NULL)  
        push(s1, current->left)  
    if(current->right !=NULL)  
        push(s1, current->right)  
}  
  :::::
```



current = 4



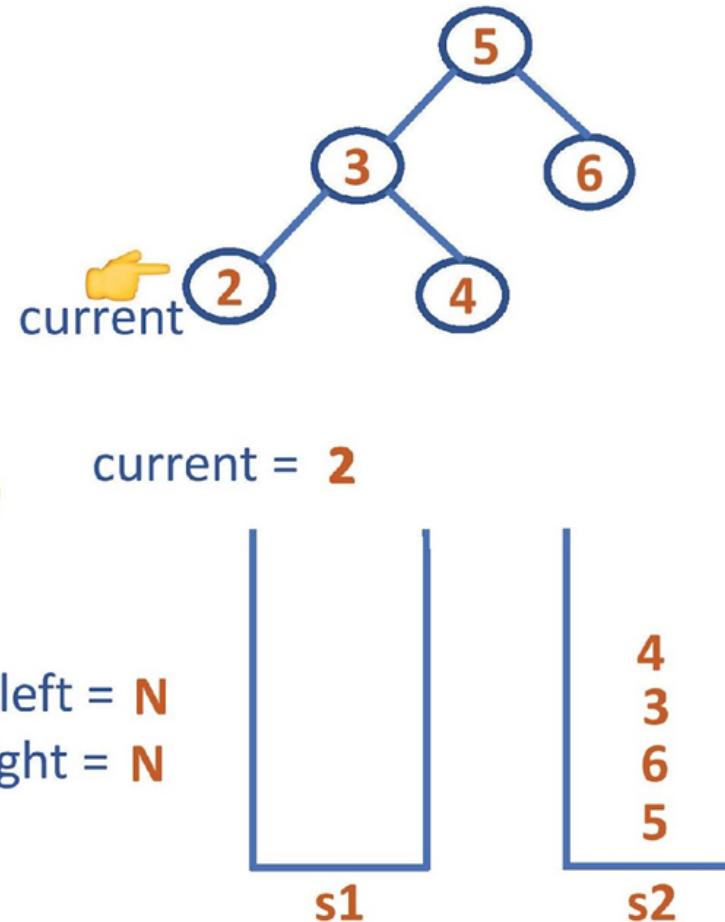
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal



```
iterativePostorder(root)
```

```
  :::::  
  while(!isEmpty(s1))  
  {  
    current = pop(s1)  
    push(s2,current) ➔  
    if(current->left !=NULL) ➔  
      push(s1, current->left)  
    if(current->right !=NULL) ➔  
      push(s1, current->right)  
  }  
  :::::  
  current->left = N  
  current->right = N
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
:::
```

```
while(!isEmpty(s1)) ➔
```

```
{
```

```
    current = pop(s1)
```

```
    push(s2,current)
```

```
    if(current->left !=NULL)
```

```
        push(s1, current->left)
```

```
    if(current->right !=NULL)
```

```
        push(s1, current->right)
```

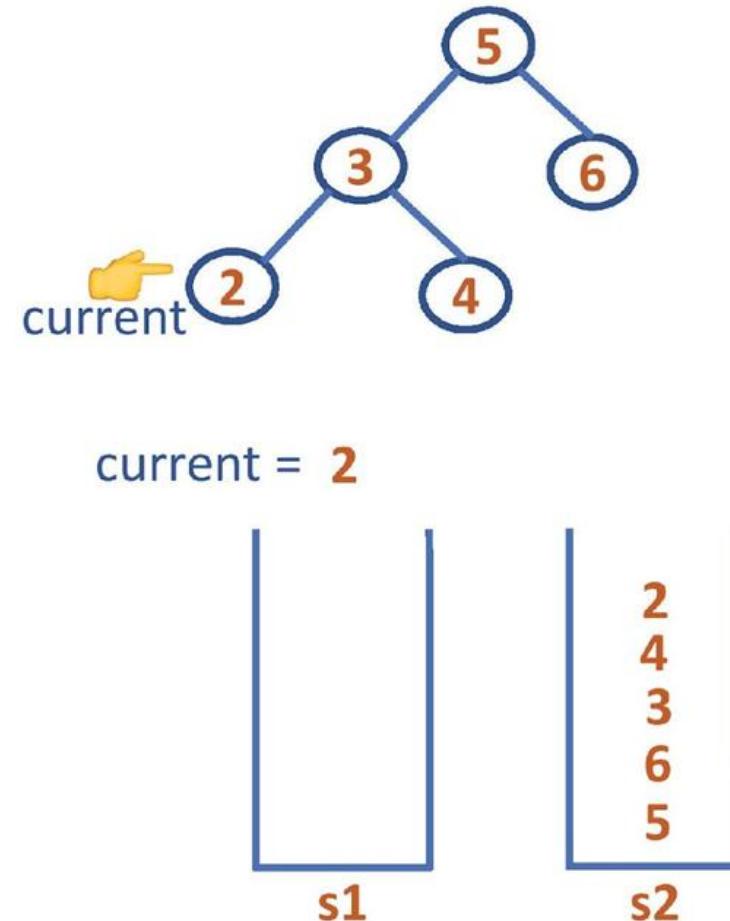
```
}
```

```
while(!isEmpty(s2)) { ➔
```

```
    current = pop(s2)
```

```
    print current->info
```

```
}
```



DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
    :::::  
while(!isEmpty(s1))
```

```
{  
    current = pop(s1)  
    push(s2,current)
```

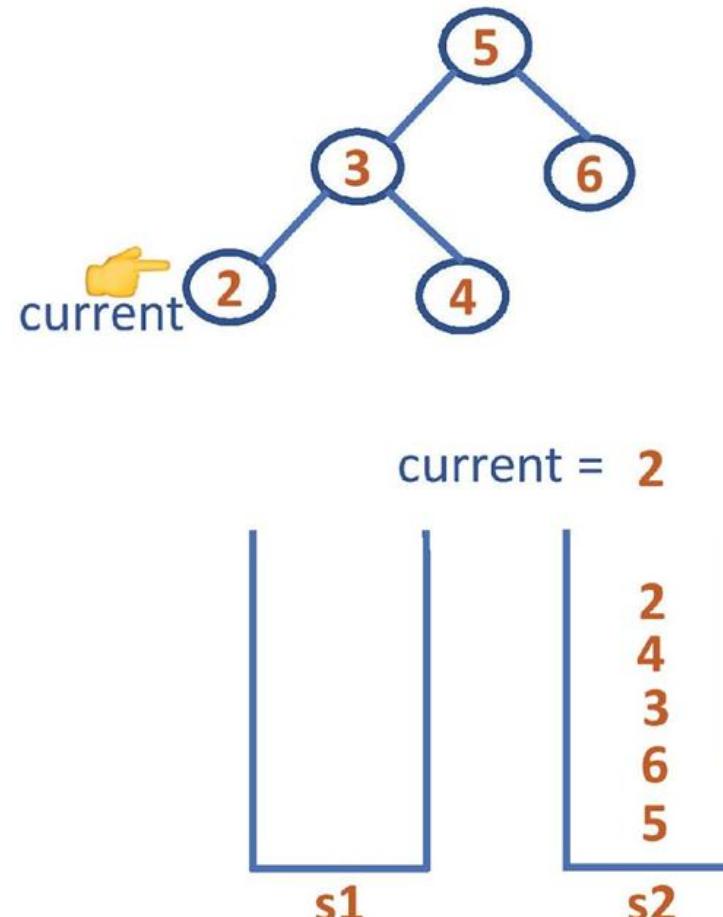
```
    if(current->left !=NULL)  
        push(s1, current->left)
```

```
    if(current->right !=NULL)  
        push(s1, current->right)
```

```
}
```

```
while(!isEmpty(s2)) {  
    current = pop(s2)   
    print current->info 
```

```
}
```



Postorder Traversal:

2

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
    :::::  
while(!isEmpty(s1))
```

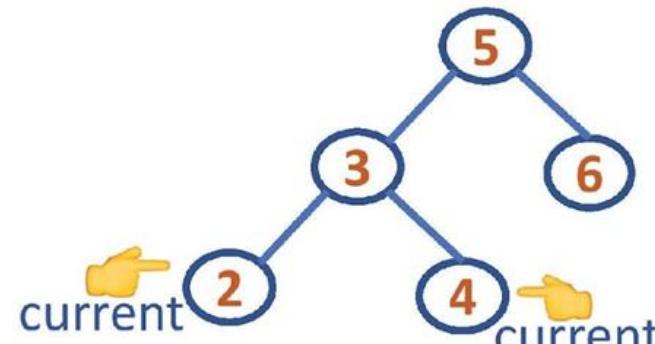
```
{  
    current = pop(s1)  
    push(s2,current)
```

```
    if(current->left !=NULL)  
        push(s1, current->left)
```

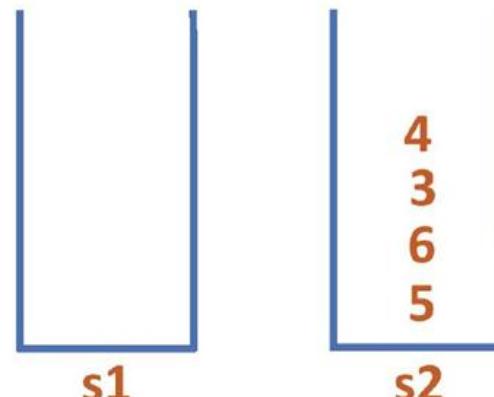
```
    if(current->right !=NULL)  
        push(s1, current->right)
```

```
}  
while(!isEmpty(s2)) {  
    current = pop(s2)  
    print current->info
```

```
}
```



current = 2



Postorder Traversal:

2 4

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

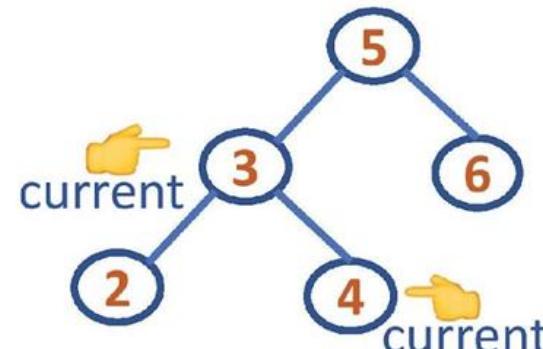
```
    :::::  
while(!isEmpty(s1))
```

```
{  
    current = pop(s1)  
    push(s2,current)}
```

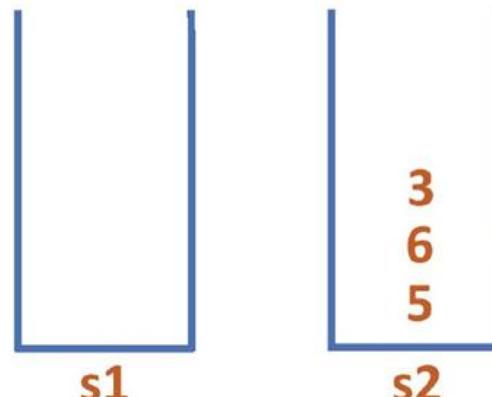
```
    if(current->left !=NULL)  
        push(s1, current->left)  
    if(current->right !=NULL)  
        push(s1, current->right)
```

```
}  
while(!isEmpty(s2)) {  
    current = pop(s2)  
    print current->info
```

```
}
```



current = 4



Postorder Traversal:

2 4 3

DATA STRUCTURES AND ITS APPLICATIONS

Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
    :::  
while(!isEmpty(s1))
```

```
{  
    current = pop(s1)  
    push(s2,current)
```

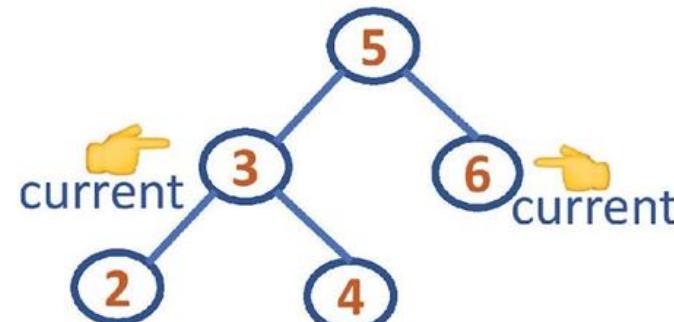
```
    if(current->left !=NULL)  
        push(s1, current->left)
```

```
    if(current->right !=NULL)  
        push(s1, current->right)
```

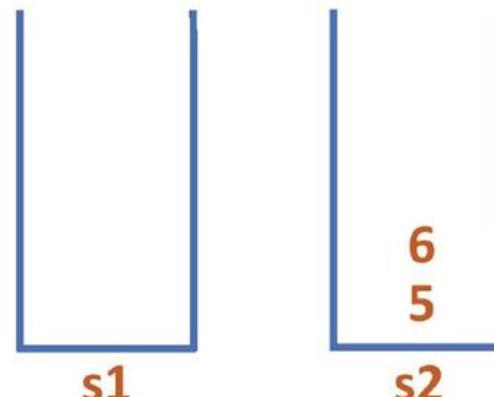
```
}
```

```
while(!isEmpty(s2)) {  
    current = pop(s2)  
    print current->info
```

```
}
```



current = 3



Postorder Traversal:

2 4 3 6

DATA STRUCTURES AND ITS APPLICATIONS

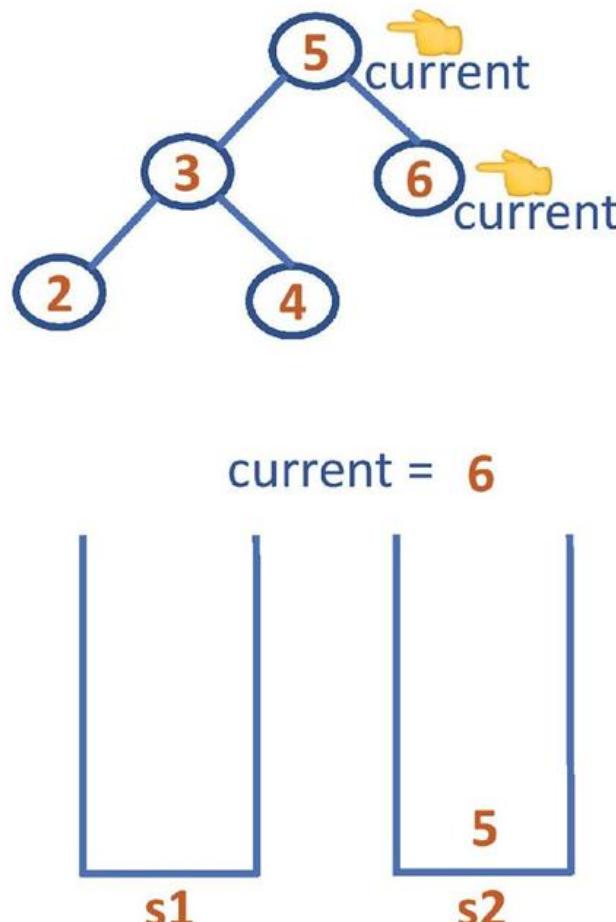
Iterative Postorder Traversal

```
iterativePostorder(root)
```

```
  :::::  
 while(!isEmpty(s1))
```

```
{  
    current = pop(s1)  
    push(s2,current)  
    if(current->left !=NULL)  
        push(s1, current->left)  
    if(current->right !=NULL)  
        push(s1, current->right)  
}
```

```
while(!isEmpty(s2)) {  
    current = pop(s2)  
    print current->info  
}
```



Postorder Traversal:

2 4 3 6 5

1. Which is the correct method for iterative postorder traversal using two stacks?

- A) Push root to stack1, pop from stack1, push children to stack2, then output stack2.
- B) Push root to stack1, pop from stack1, push to stack2, and push its children into stack1.
- C) Perform inorder traversal and reverse the order.
- D) Always push left child only into stack2.

1. Which is the correct method for iterative postorder traversal using two stacks?

- A) Push root to stack1, pop from stack1, push children to stack2, then output stack2.
- B) Push root to stack1, pop from stack1, push to stack2, and push its children into stack1.**
- C) Perform inorder traversal and reverse the order.
- D) Always push left child only into stack2.

2. Which traversal is most suitable for evaluating an expression tree using a stack?

- A) Inorder
- B) Preorder
- C) Postorder
- D) Level-order

2. Which traversal is most suitable for evaluating an expression tree using a stack?

- A) Inorder
- B) Preorder
- C) Postorder
- D) Level-order

3. Suppose you want to print all values of a BST in ascending order without recursion. Which traversal and data structure would you use?

- A) Preorder traversal with stack
- B) Inorder traversal with stack
- C) Postorder traversal with queue
- D) Level-order traversal with queue

3. Suppose you want to print all values of a BST in ascending order without recursion. Which traversal and data structure would you use?

- A) Preorder traversal with stack
- B) Inorder traversal with stack**
- C) Postorder traversal with queue
- D) Level-order traversal with queue

4. Which of the following is true about level-order traversal?

- A) It uses recursion only
- B) It visits nodes depth-first
- C) It visits nodes breadth-first
- D) It always requires a stack

4. Which of the following is true about level-order traversal?

- A) It uses recursion only
- B) It visits nodes depth-first
- C) It visits nodes breadth-first
- D) It always requires a stack

5. You want to print all paths from root to leaves. Which traversal is generally preferred?

- A) Pre-order
- B) In-order
- C) Post-order
- D) Level-order

5. You want to print all paths from root to leaves. Which traversal is generally preferred?

- A) Pre-order
- B) In-order
- C) Post-order
- D) Level-order

(Explanation: Pre-order allows keeping track of the current path from root to node.)



THANK YOU

Shylaja S S

Department of Computer Science
& Engineering

shylaja.sharath@pes.edu

+91 9449867804