DATA STRUCTURES AND ITS APPLICATIONS

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

DATA STRUCTURES AND ITS APPLICATIONS

Implementation & Traversal of Trees

Shylaja S S

Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversals

Important operation: Traversal

Traversal: Moving through all the nodes in a binary tree and visiting each one in turn

Trees: There are many orders possible since it is a nonlinear DS

Tasks: 1. Visiting a node denoted by V

2. Traversing the left subtree denoted by L

3. Traversing the right subtree denoted by R

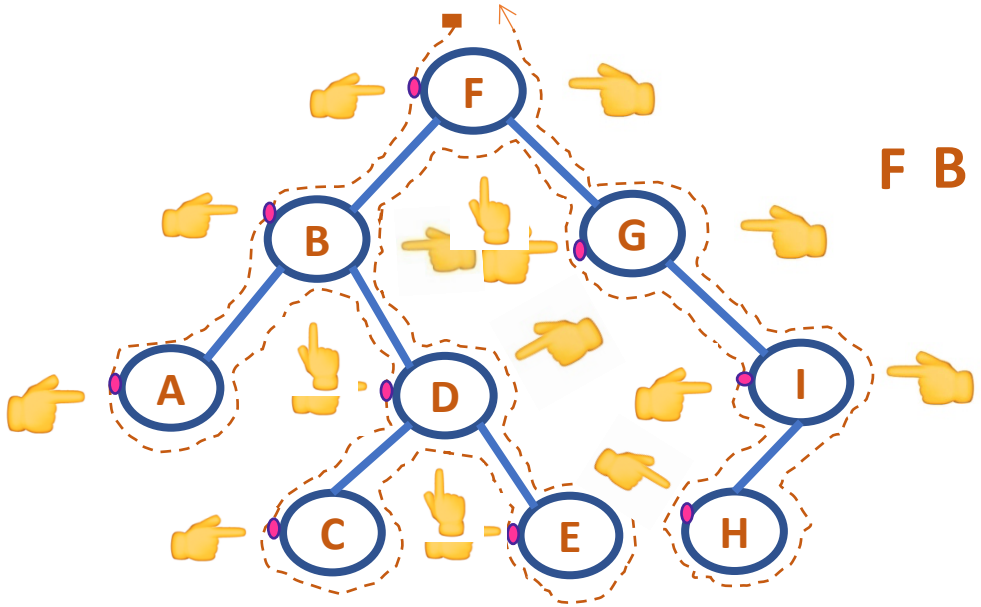
Six ways to arrange them: VLR, LVR, LRV, VRL, RVL, RLV

Standard Traversals include: VLR-Preorder, LVR-Inorder, LRV-Postorder

Binary Tree Traversal: Preorder

Steps:

- Root Node is visited before the subtrees
- Left subtree is traversed in preorder
- Right subtree is traversed in preorder



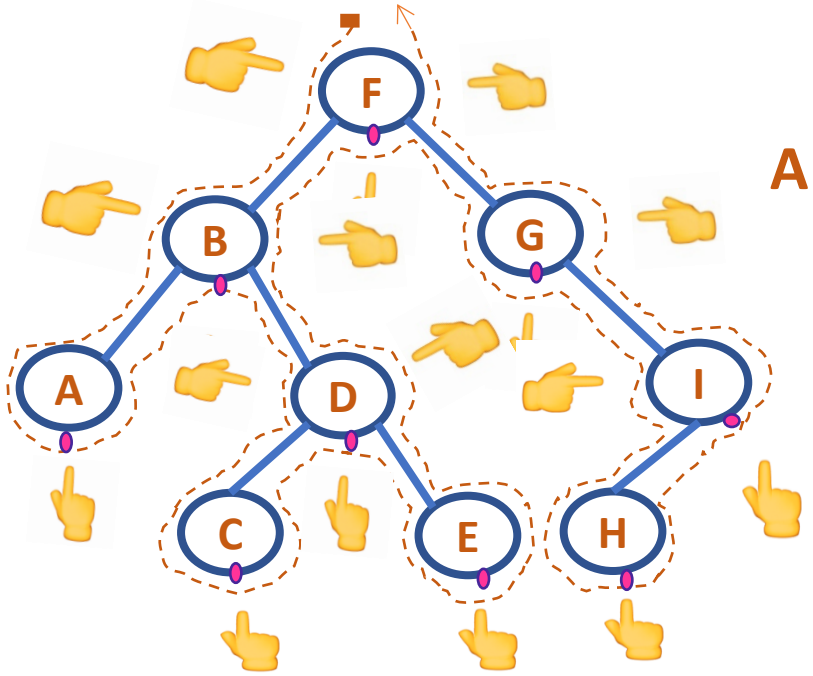
F B A D C E G I H

DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversal: Inorder

Steps:

- Left subtree is traversed in Inorder
- Root Node is visited
- Right subtree is traversed in Inorder



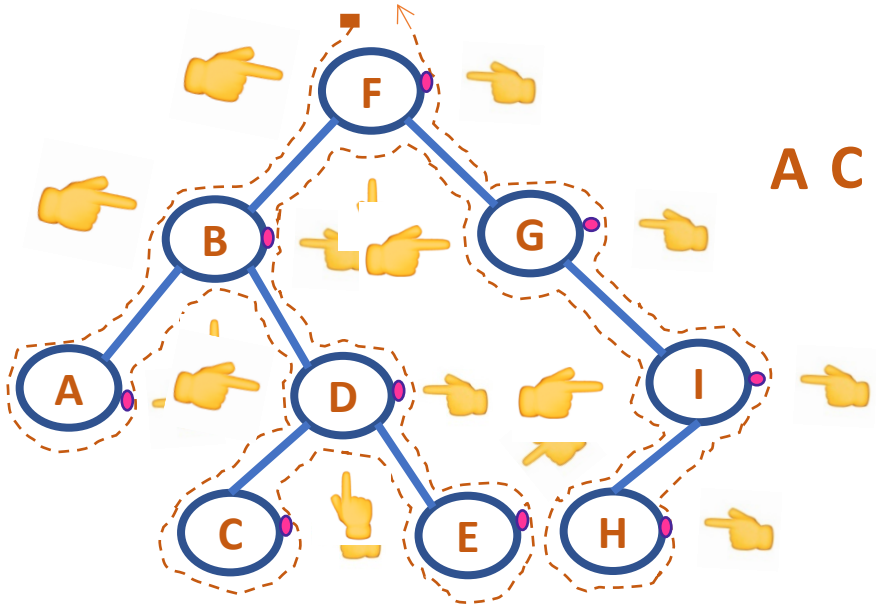
A B C D E F G H I

DATA STRUCTURES AND ITS APPLICATIONS

Binary Tree Traversal: Postorder

Steps:

- Left subtree is traversed in postorder
- Right subtree is traversed in postorder
- Root Node is visited



A C E D B H I G F

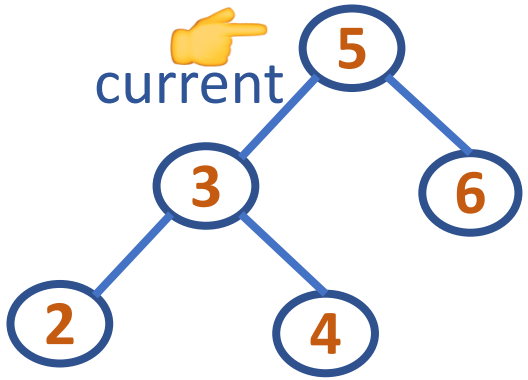
Iterative Inorder Traversal

```
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)
```

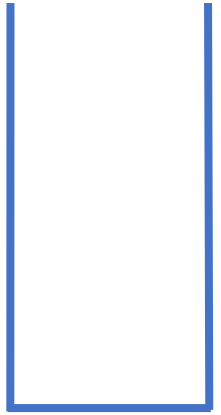
DATA STRUCTURES AND ITS APPLICATIONS

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)
```



current = **5**

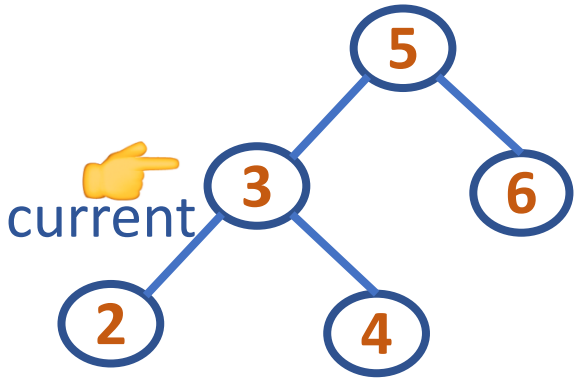


Note: Stack has Address of Nodes Pushed In

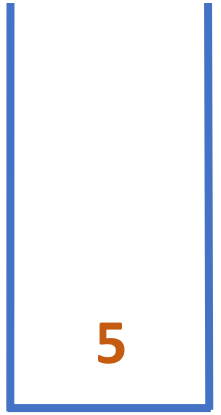
DATA STRUCTURES AND ITS APPLICATIONS

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)
```

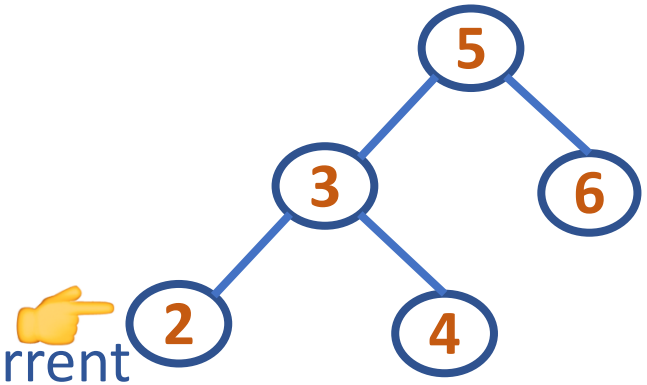


current = **3**

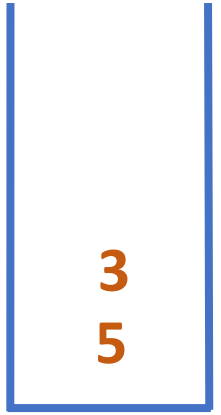


Iterative Inorder Traversal

```
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)
```



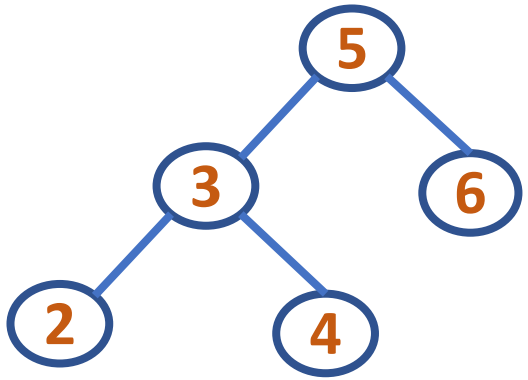
current = **2**



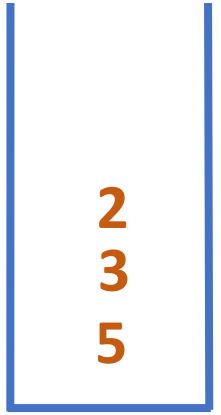
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Inorder Traversal

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

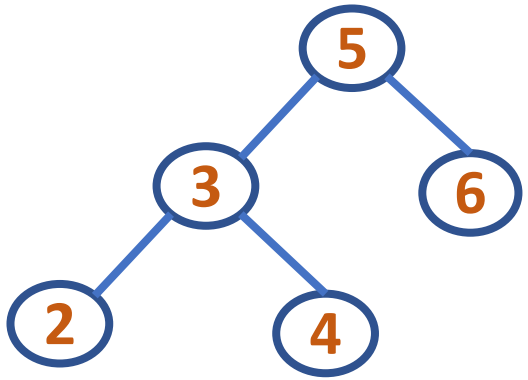


current = **N**



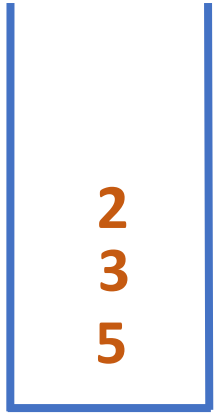
Iterative Inorder Traversal

```
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**

poppedNode =

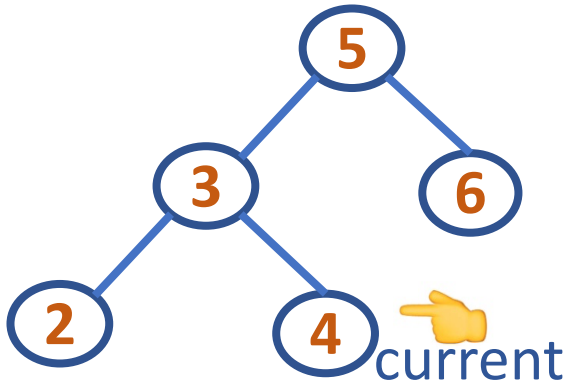


Inorder Traversal:

2

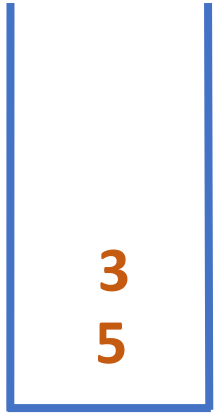
Iterative Inorder Traversal

```
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 = Null

poppedNode = 2

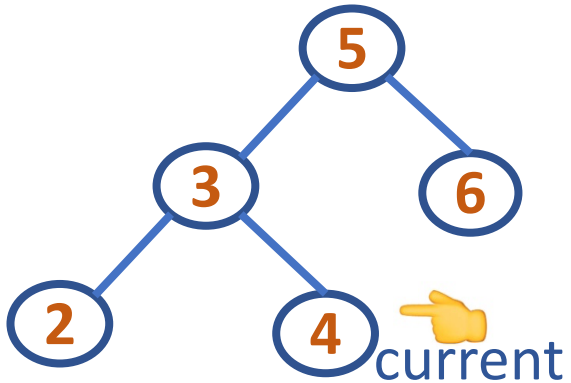


Inorder Traversal:

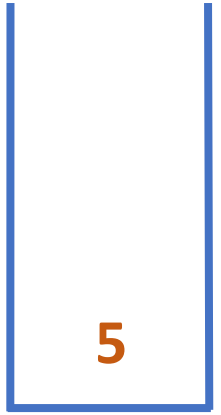
2 3

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 = **4**

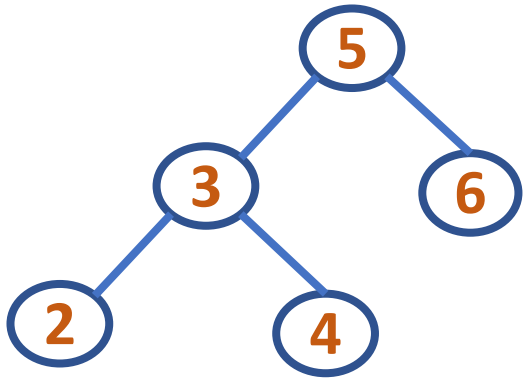


Inorder Traversal:

2 3

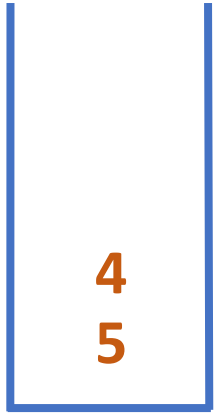
Iterative Inorder Traversal

```
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**

poppedNode = **3**



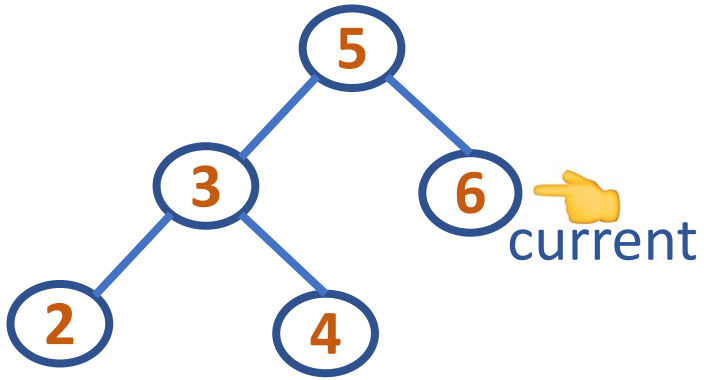
Inorder Traversal:

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)
      print poppedNode ->info
      current = poppedNode ->right
} while(!isEmpty(s) or current != null)
```

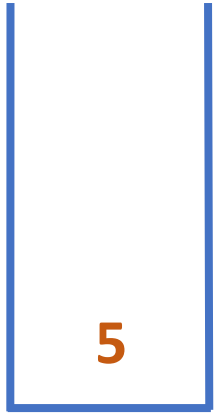


Inorder Traversal:

2 3 4 5

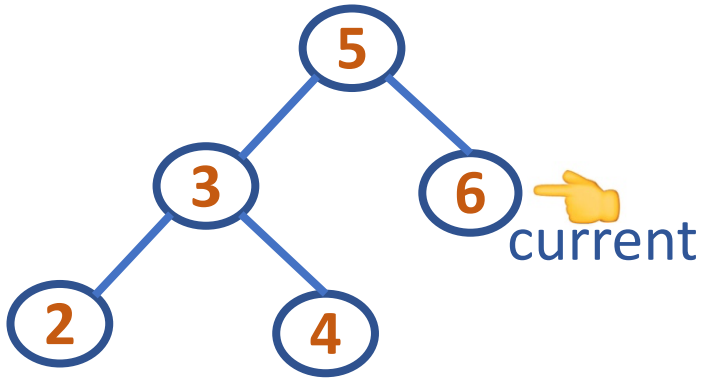
current = 6

poppedNode = 4

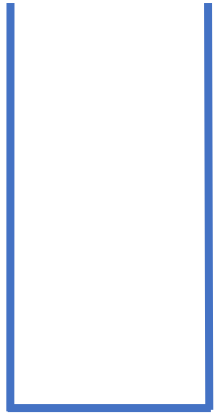


Iterative Inorder Traversal

```
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 = **6**



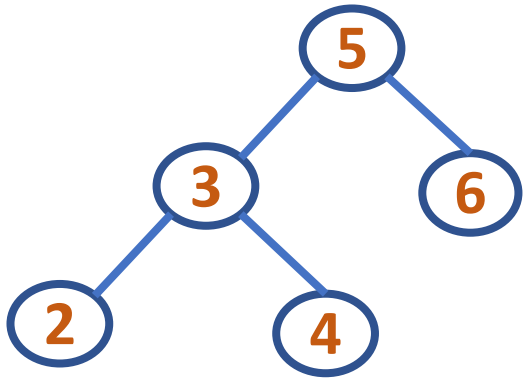
Inorder Traversal:

2 3 4 5

DATA STRUCTURES AND ITS APPLICATIONS

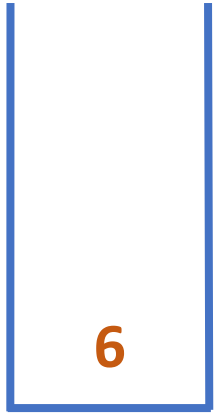
Iterative Inorder Traversal

```
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)
```



current = **N**

poppedNode =



Inorder Traversal:

2 3 4 5 6

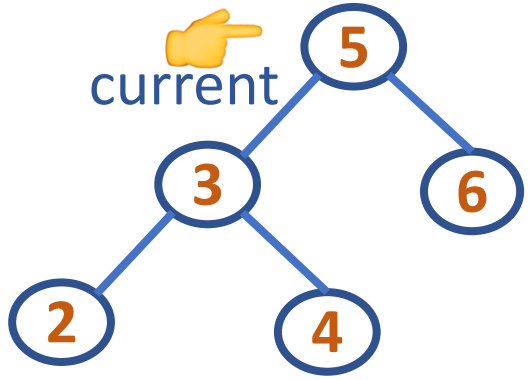
Iterative Preorder Traversal

```
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)
}
```

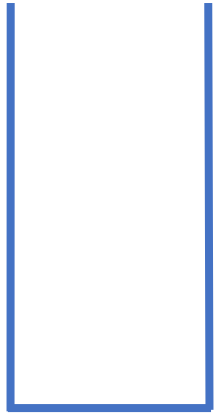
DATA STRUCTURES AND ITS APPLICATIONS

Iterative Preorder Traversal

```
iterativePreorder(root)
current=root
if (current == null)
    return
s = emptyStack
push(s, current)
    ⋮
    ⋮
```



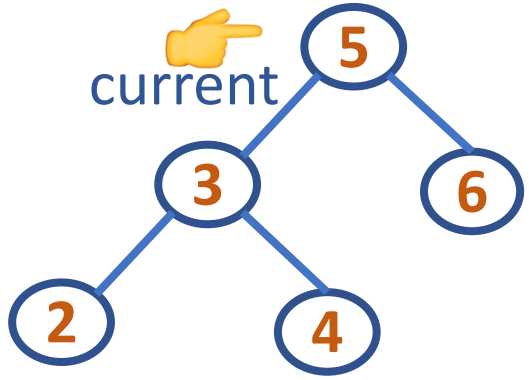
current = **5**



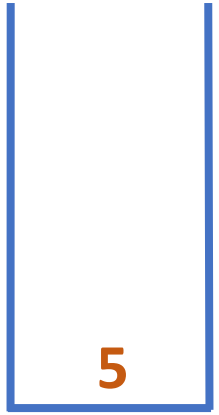
Note: Stack has Address of Nodes Pushed In

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

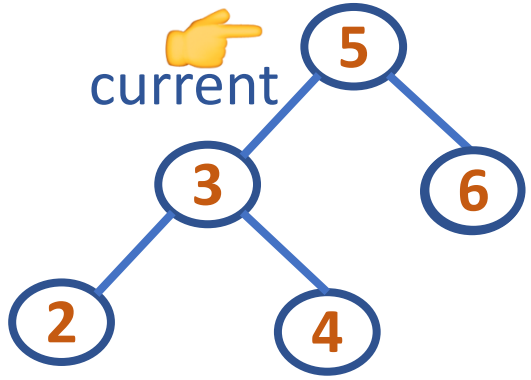


Preorder Traversal:

5

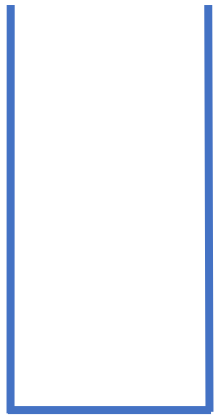
Iterative Preorder Traversal

```
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

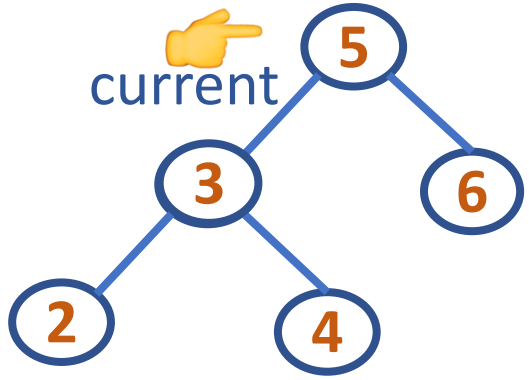


Preorder Traversal:

5

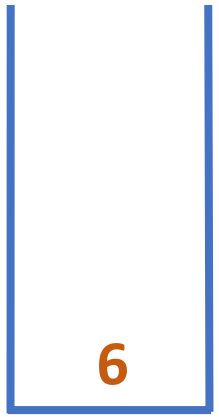
Iterative Preorder Traversal

```
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

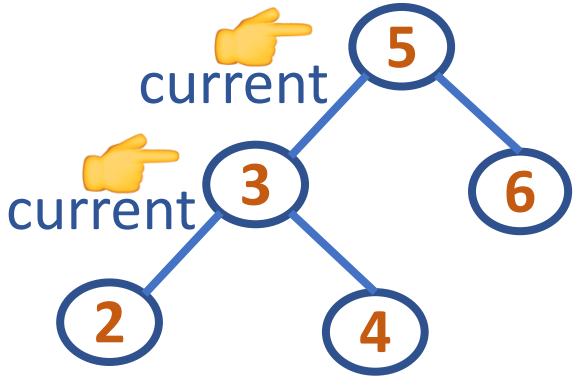


Preorder Traversal:

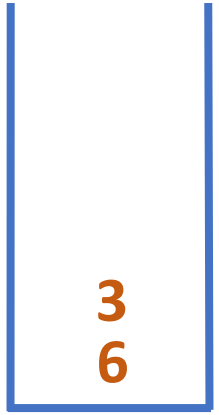
5

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

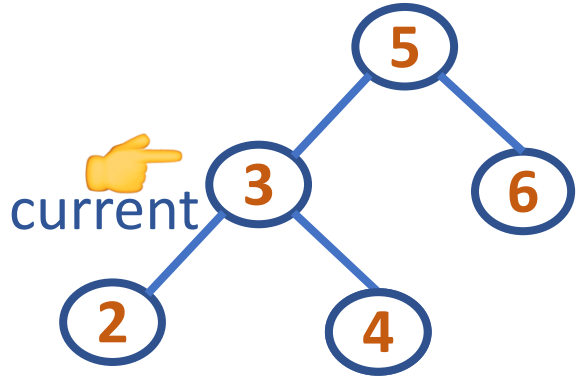


Preorder Traversal:

5 3

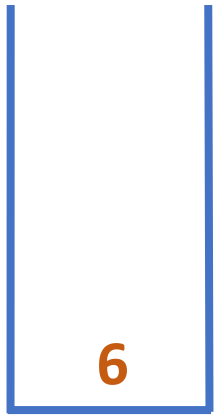
Iterative Preorder Traversal

```
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

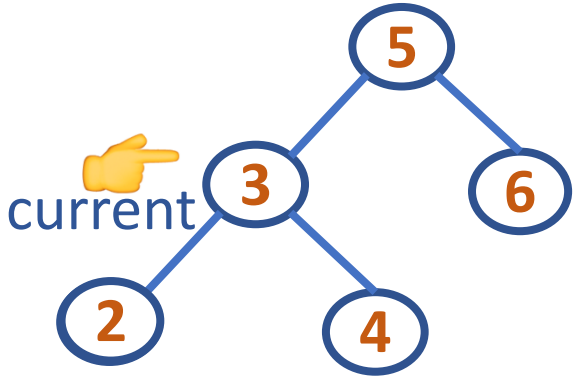


Preorder Traversal:

5 3

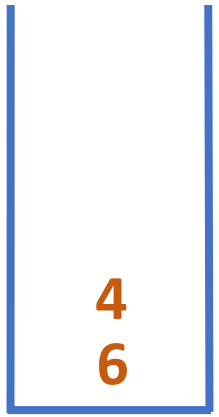
Iterative Preorder Traversal

```
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

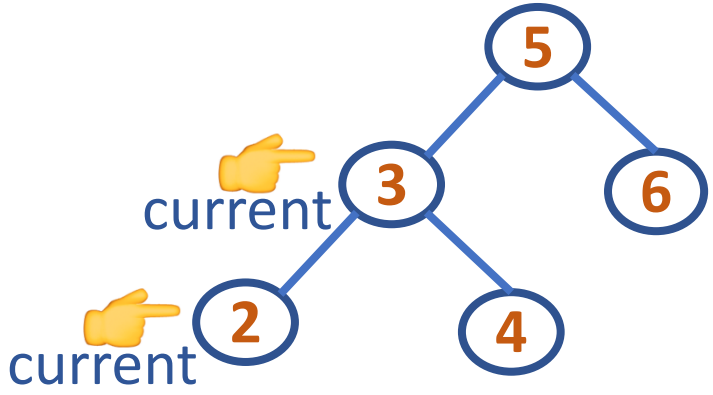


Preorder Traversal:

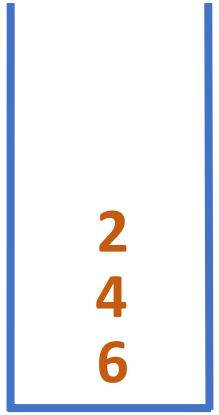
5 3

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

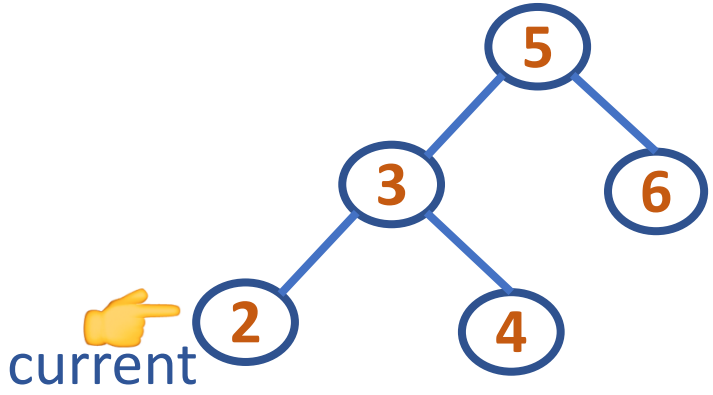


Preorder Traversal:

5 3 2

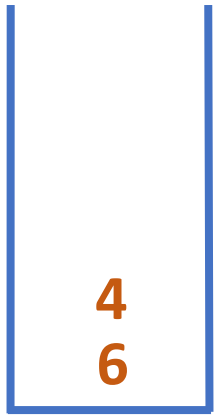
Iterative Preorder Traversal

```
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

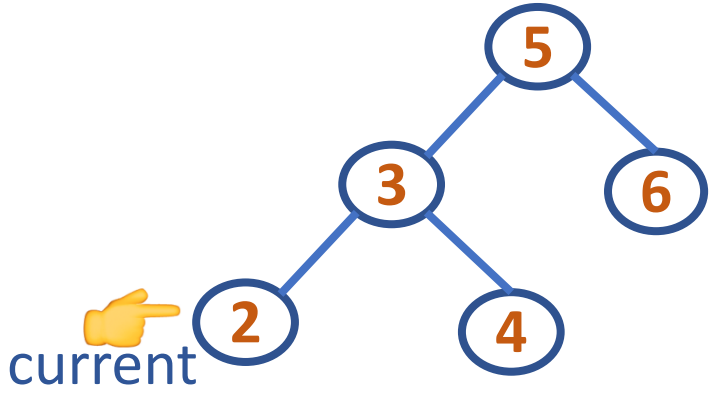


Preorder Traversal:

5 3 2

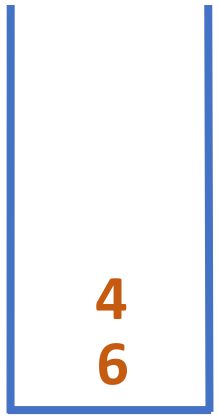
Iterative Preorder Traversal

```
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->left = N

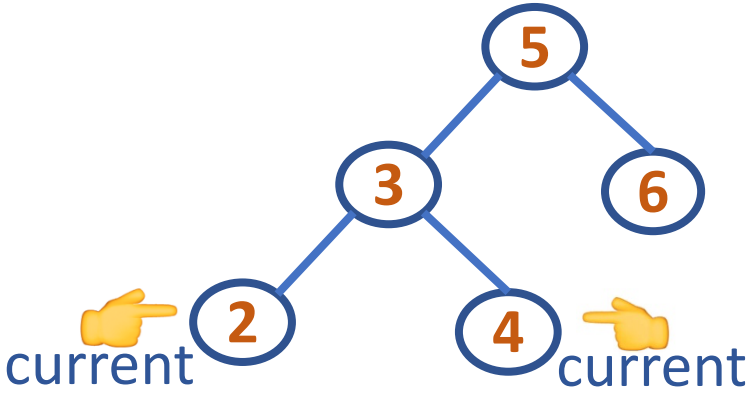


Preorder Traversal:

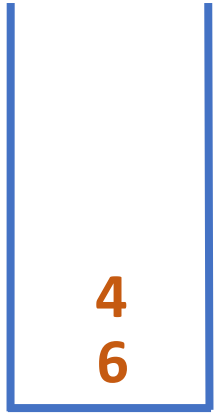
5 3 2

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

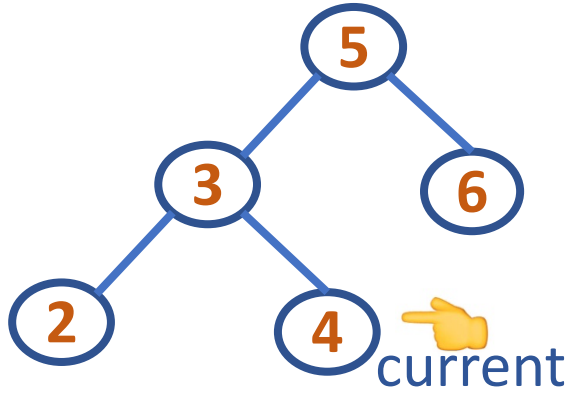


Preorder Traversal:

5 3 2 4

Iterative Preorder Traversal

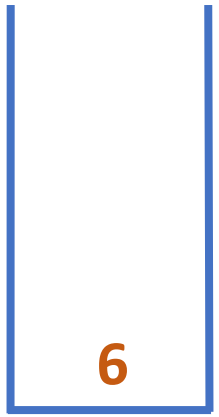
```
iterativePreorder(root)
      ⋮
while (!isEmpty(s))
{
   current = pop(s)
   print current ->info
   if(current->right != null)
      push(s, current->right)
   if(current->left != null)
      push(s, current->left)
}
```



Preorder Traversal:
5 3 2 4

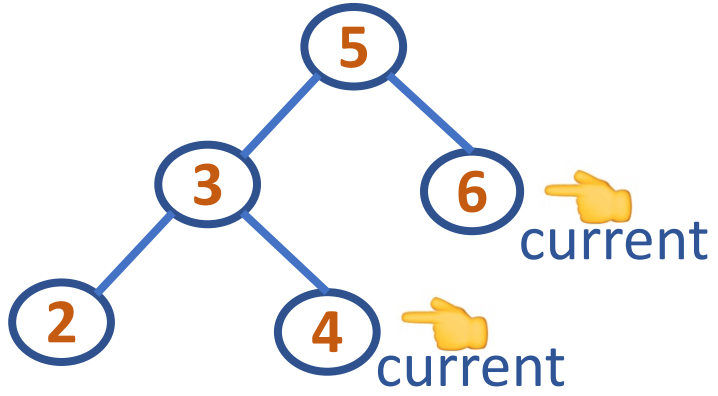
current = 4

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



Iterative Preorder Traversal

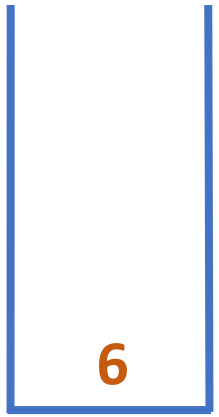
```
iterativePreorder(root)
      ⋮
while (!isEmpty(s))
{
    current = pop(s)
    print current ->info
    if(current->right != null)
        push(s, current->right)
    if(current->left != null)
        push(s, current->left)
}
```



Preorder Traversal:
5 3 2 4 6

current = 4

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

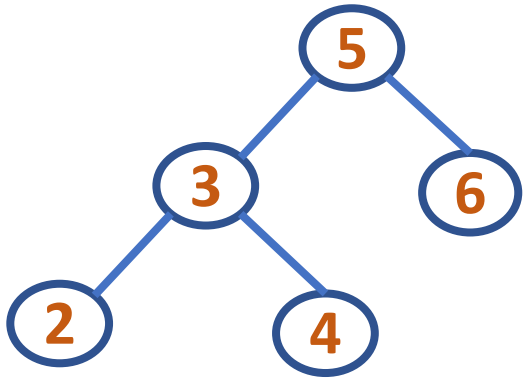


Iterative Postorder Traversal

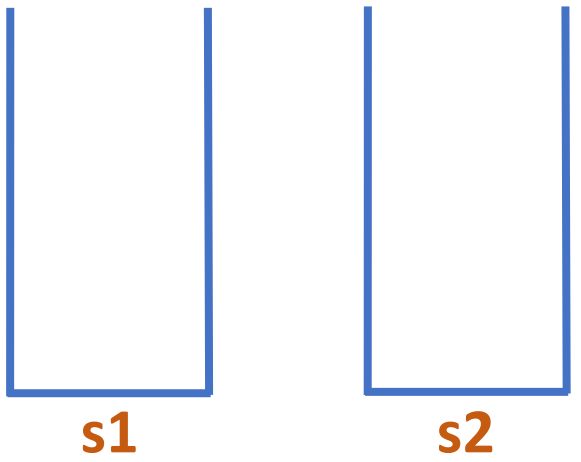
```
iterativePostorder(root)
s1 = emptyStack ; s2 = emptyStack ; push(s1, root)
while(!isEmpty(s1)) {
      current = pop(s1)
      push(s2,current)
      if(current->left !=NULL)
            push(s1, current->left)
      if(current->right !=NULL)
            push(s1, current->right)
}
while(!isEmpty(s2))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

Iterative Postorder Traversal

iterativePostorder(root)
s1 = emptyStack 👈
s2 = emptyStack 👈
push(s1, root) 👈
⋮



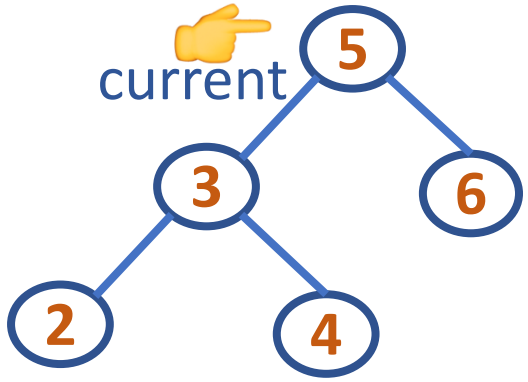
root = **5**



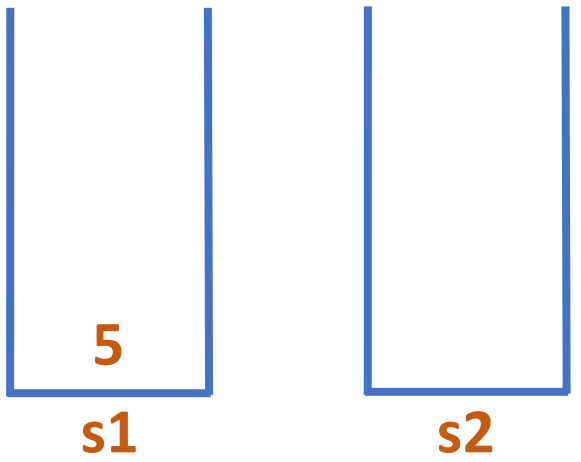
Note: Stacks have Address of Nodes Pushed In

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)
}
      ⁞⁞
```

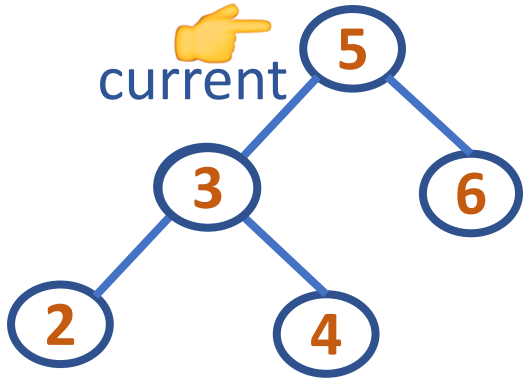


current =



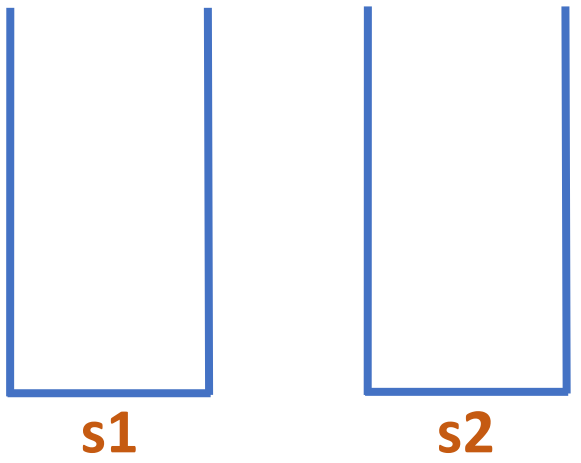
Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```



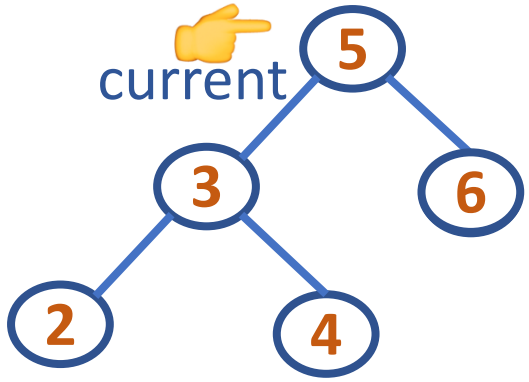
current = **5**

current->left = **3**



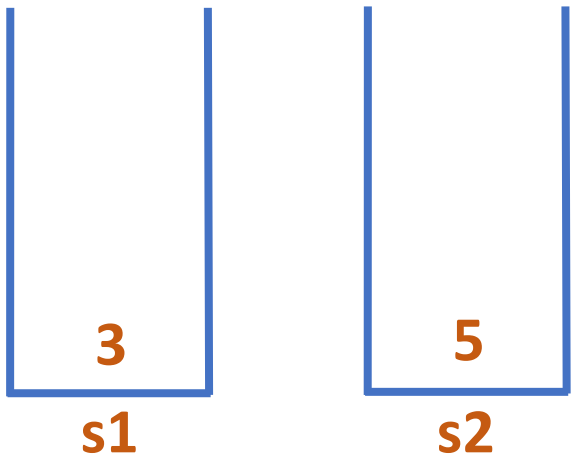
Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```



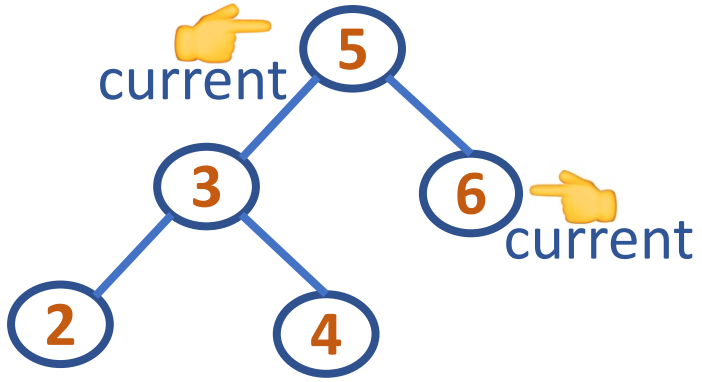
current = 5

current->right = 6

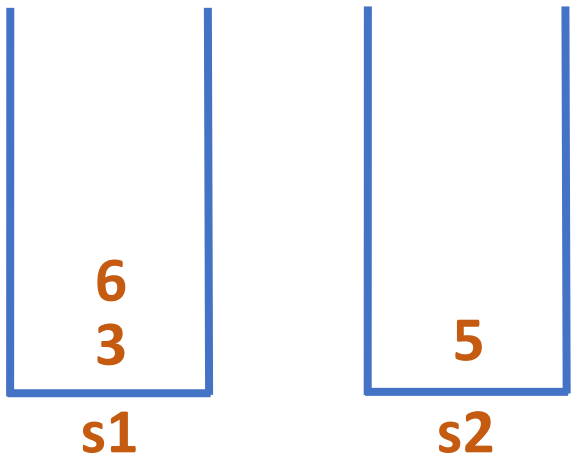


Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```

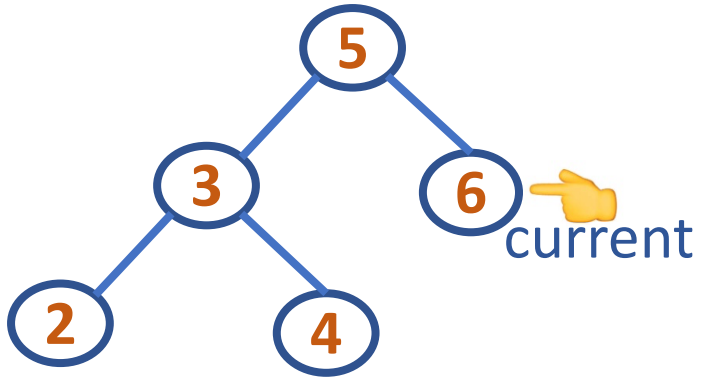


current = 5



Iterative Postorder Traversal

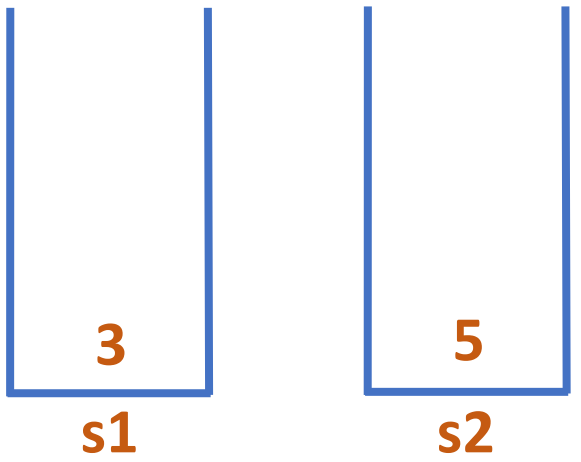
```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = **6**

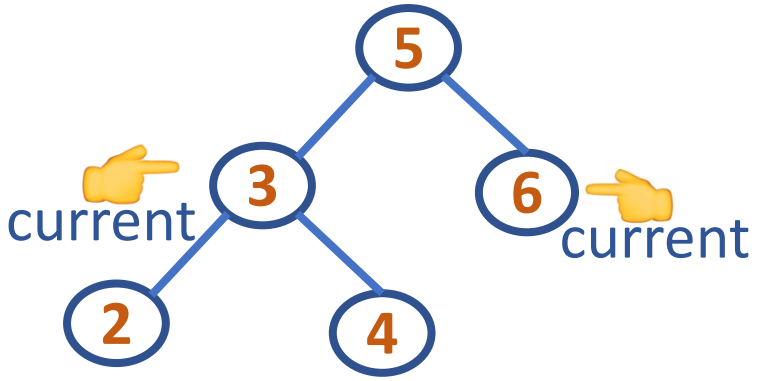
current->left = **N**

current->right = **N**

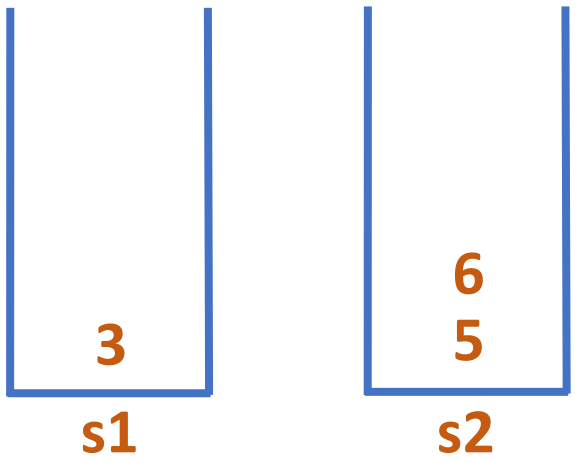


Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```

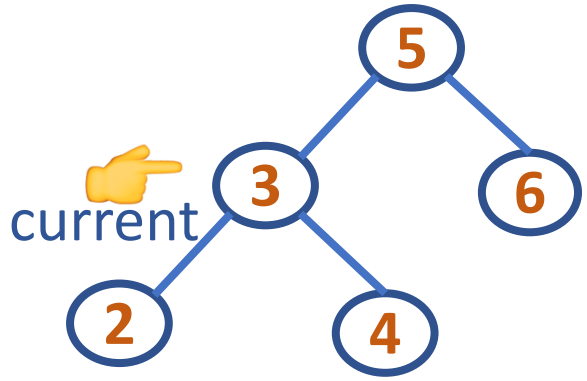


current = 6

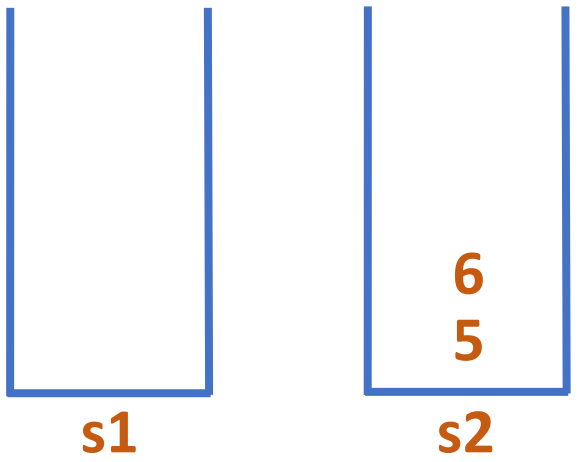


Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```

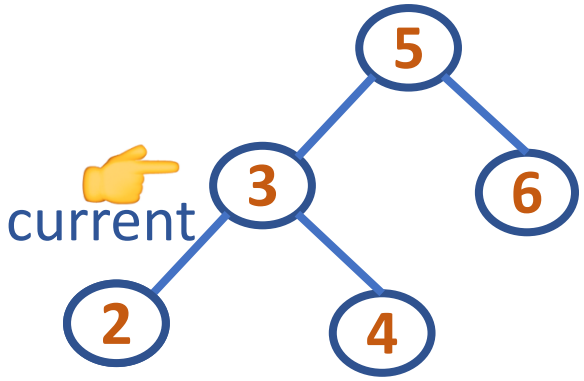


current = **3**



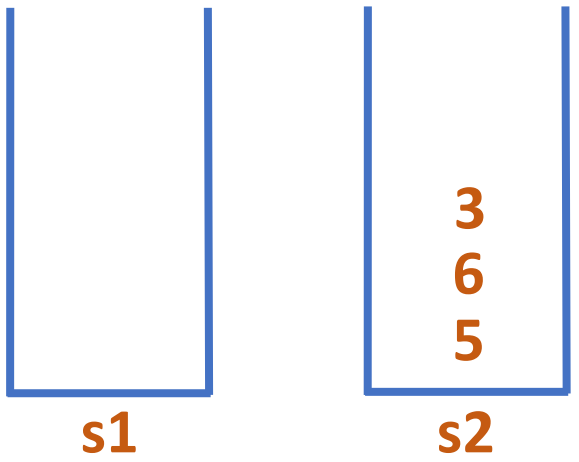
Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```



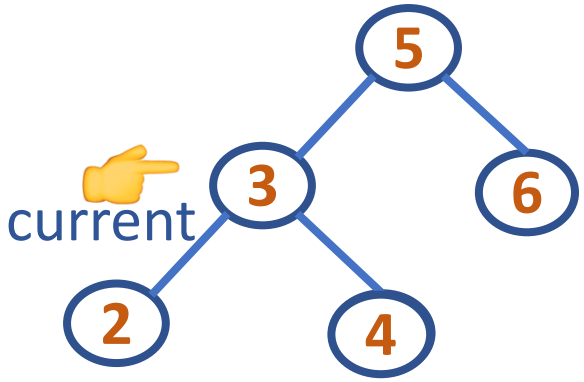
current = 3

current->left = 2



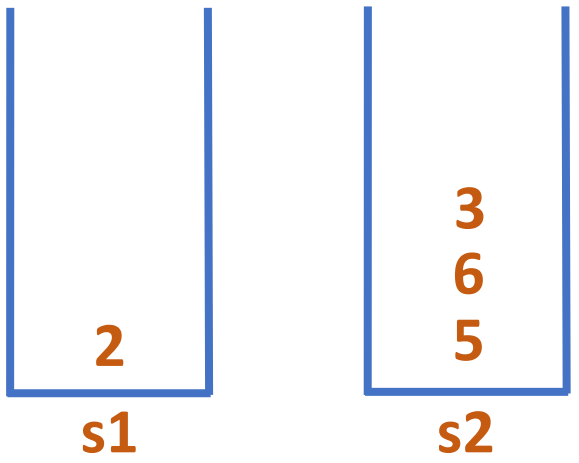
Iterative Postorder Traversal

```
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)
}
      ⋮⋮
```



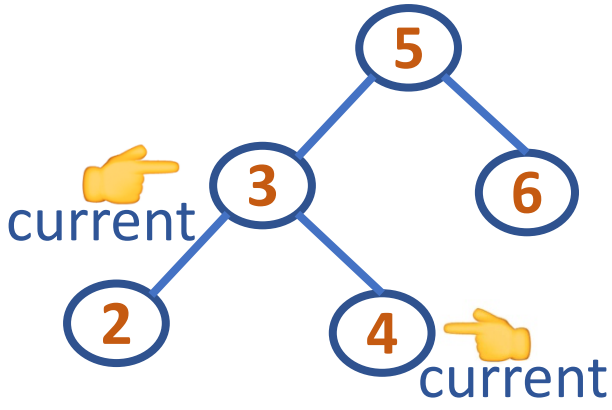
current = 3

current->right = 4

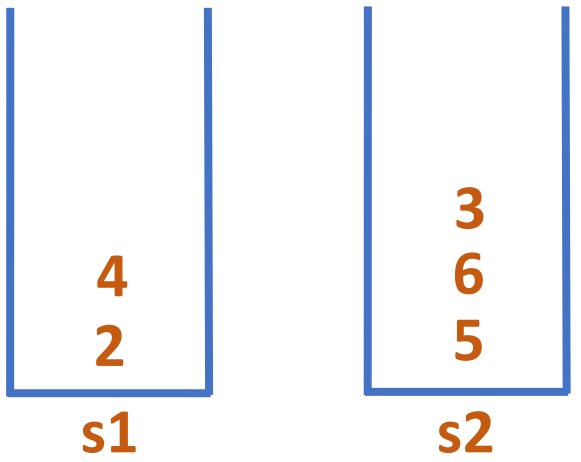


Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```

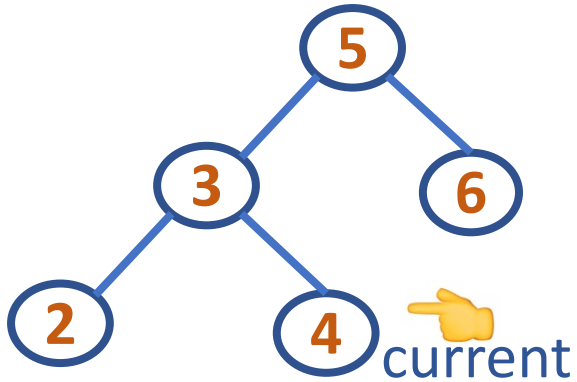


current = 3



Iterative Postorder Traversal

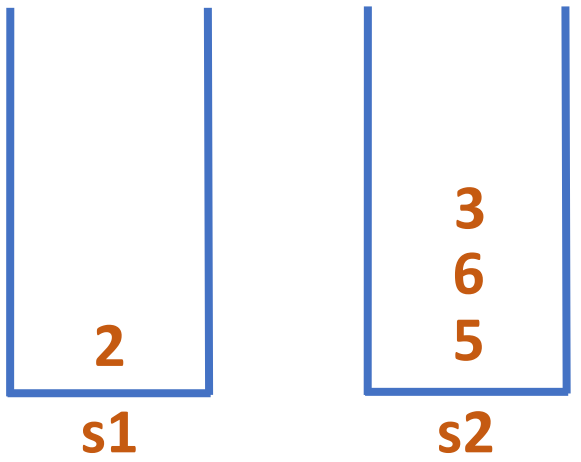
```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = **4**

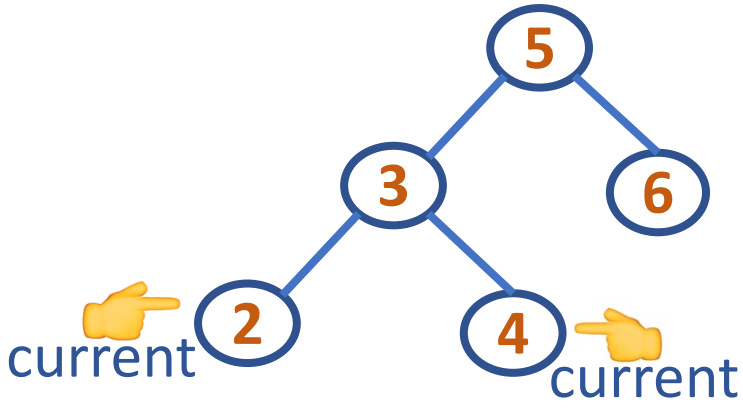
current->left = **N**

current->right = **N**

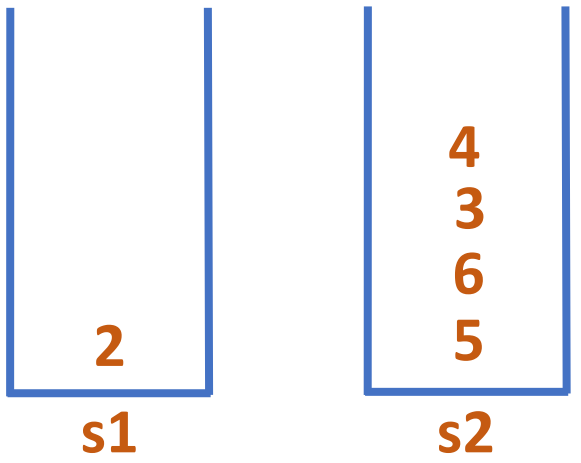


Iterative Postorder Traversal

```
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)
}
      ⁞⁞
```

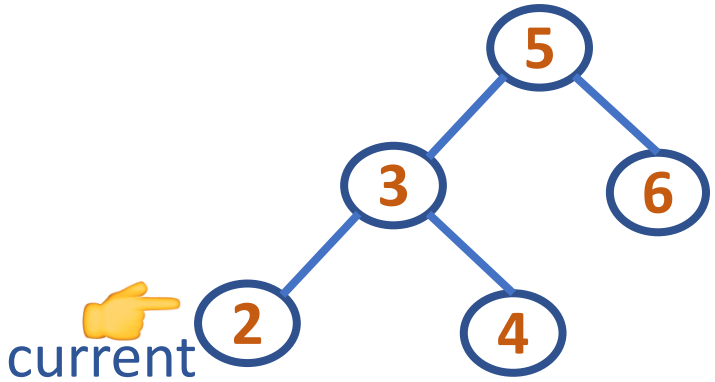


current = 4



Iterative Postorder Traversal

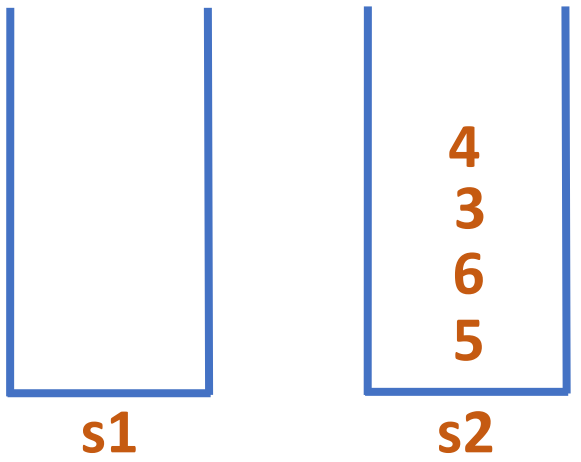
```
iterativePostorder(root)
      ⁞⁞
while(!isEmpty(s1))
{
   current = pop(s1)
   push(s2,current)
   if(current->left !=NULL)
     push(s1, current->left)
   if(current->right !=NULL)
     push(s1, current->right)
}
      ⁞⁞
```



current = **2**

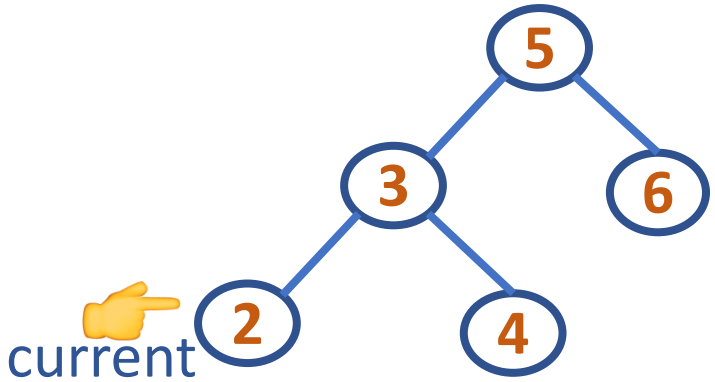
current->left = **N**

current->right = **N**

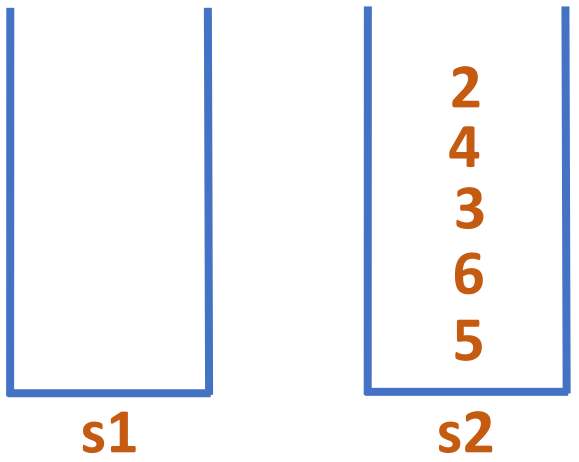


Iterative Postorder Traversal

```
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
}
```

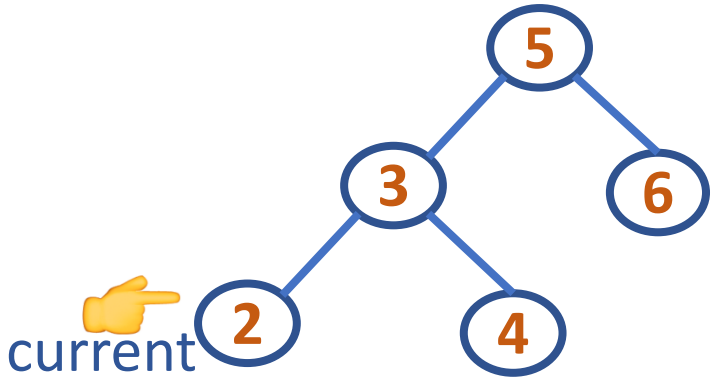


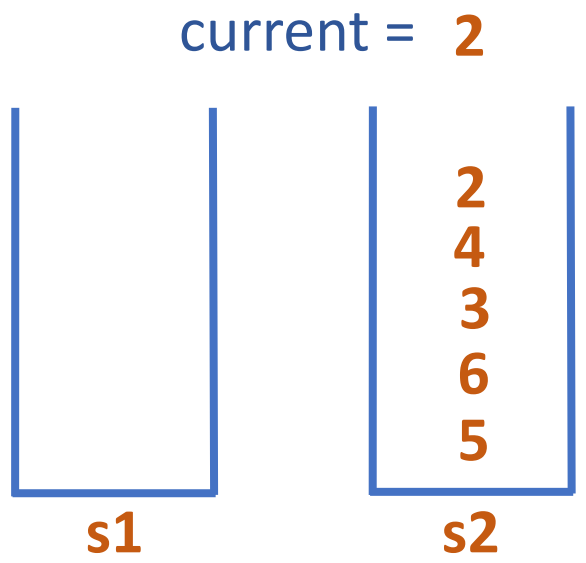
current = 2



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
}
```



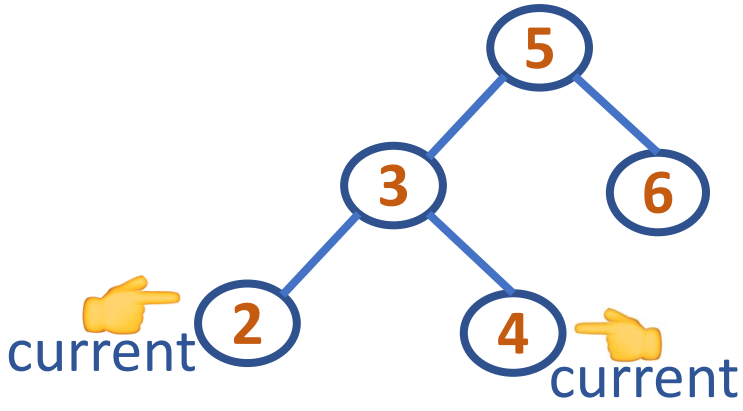


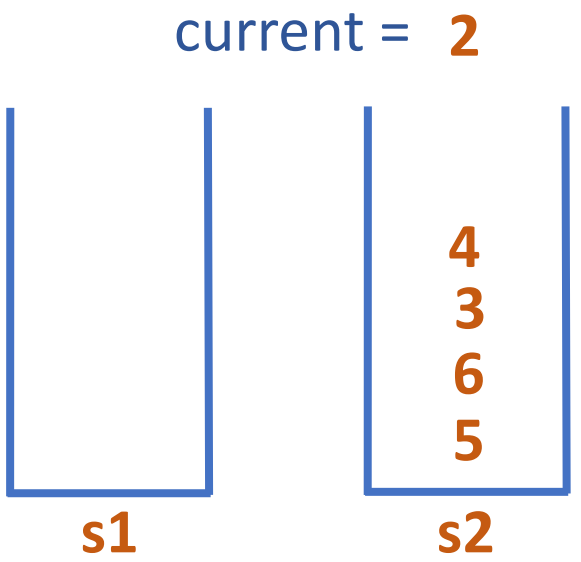
Postorder Traversal:

2

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
}
```



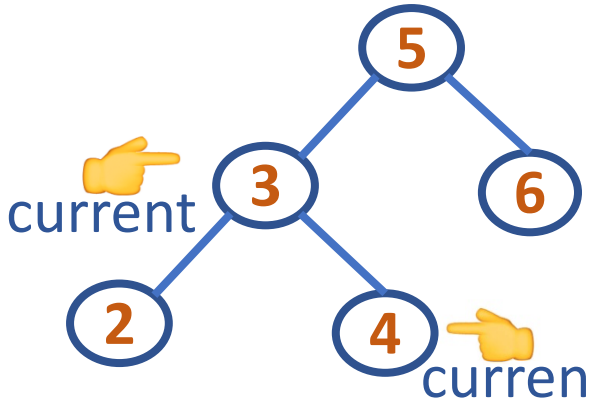


Postorder Traversal:

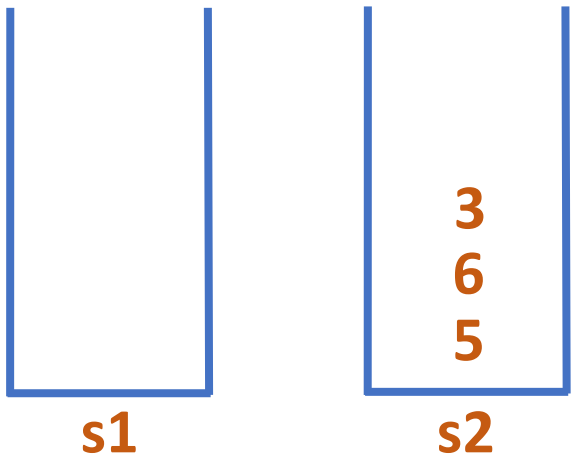
2 4

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

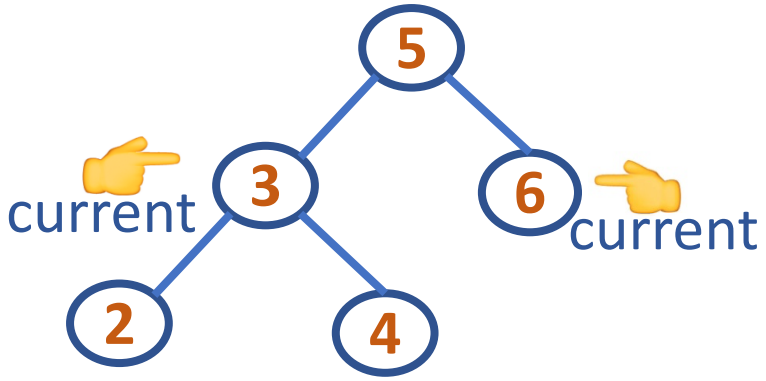


Postorder Traversal:

2 4 3

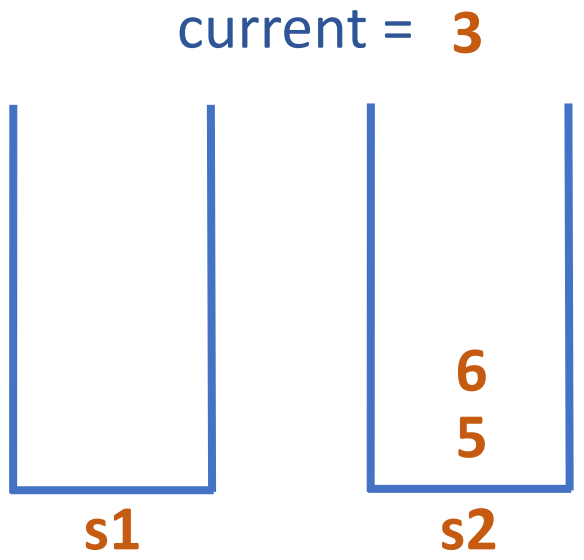
Iterative Postorder Traversal

```
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
}
```



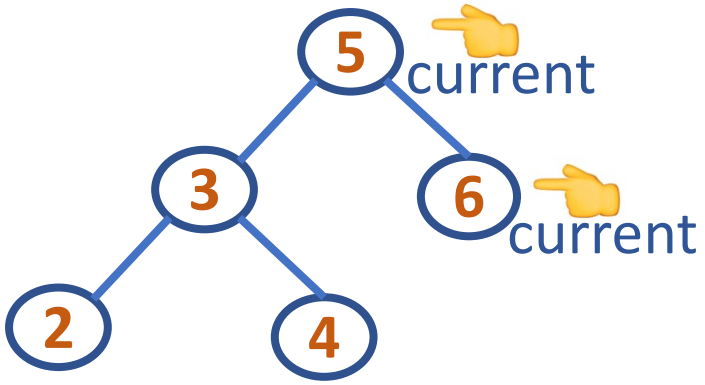
Postorder Traversal:

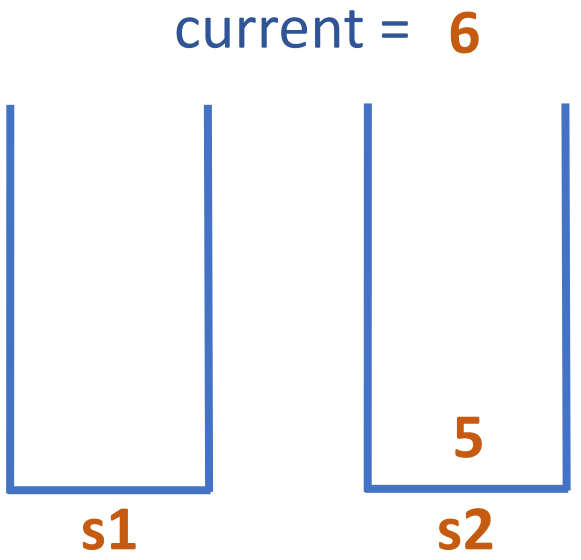
2 4 3 6



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

Tree Traversal

Structure of a treenode revisited

```
struct treenode{
        int info;
        struct treenode *child;
        struct treenode *sibling;
};
```

Tree Traversal

With the treenode implemented as having pointers to first child and immediate sibling, the traversal preorder, inorder and postorder for a tree are defined as below:

Preorder:

1. Visit the root of the first tree in the forest
2. Traverse in preorder the forest formed by the subtrees of the first tree, if any
3. Traverse in preorder the forest formed by the remaining trees in the forest, if any

Tree Traversal

```
void preorder(TREE *root)
{
    if(root!=NULL)
    {
      printf(" %d ",root->info);
      preorder(root->child);
      preorder(root->sibling);
    }
}
```

Tree Traversal

Inorder

1. Traverse in inorder the forest formed by the subtrees of the first tree, if any
2. Visit the root of the first tree in the forest
3. Traverse in inorder the forest formed by the remaining trees in the forest, if any

Tree Traversal


```
void inorder(TREE *root)
{
  if(root!=NULL)
   {
     inorder(root->child);
     printf(" %d ",root->info);
     inorder(root->sibling);
   }
}
```

Tree Traversal

Postorder

1. Traverse in postorder the forest formed by the subtrees of the first tree, if any
2. Traverse in postorder the forest formed by the remaining trees in the forest, if any
3. Visit the root of the first tree in the forest

Tree Traversal

```
void postorder(TREE *root)
{
  if(root!=NULL)
   {
     postorder(root->child);
     postorder(root->sibling);
     printf(" %d ", root->info);
   }
}
```

THANK YOU

Shylaja S S

Department of Computer Science & Engineering

shylaja.sharath@pes.edu

+91 9449867804