CIS 351-Data Structure-Non-recursive Tree Traversal
April 9, 2020

Dr. Farzana Rahman

Syracuse University

Nonrecursive Binary Tree Traversal Algorithms

- How to do nonrecursive inorder, preorder, and postorder traversal algorithms
- Using **Stack** is the obvious way to traverse tree without recursion

Nonrecursive Preorder Traversal

- For each node, first the node is visited, then the left subtree, and then the right subtree
- Must save a pointer to a node before visiting the left subtree, in order to visit the right subtree later

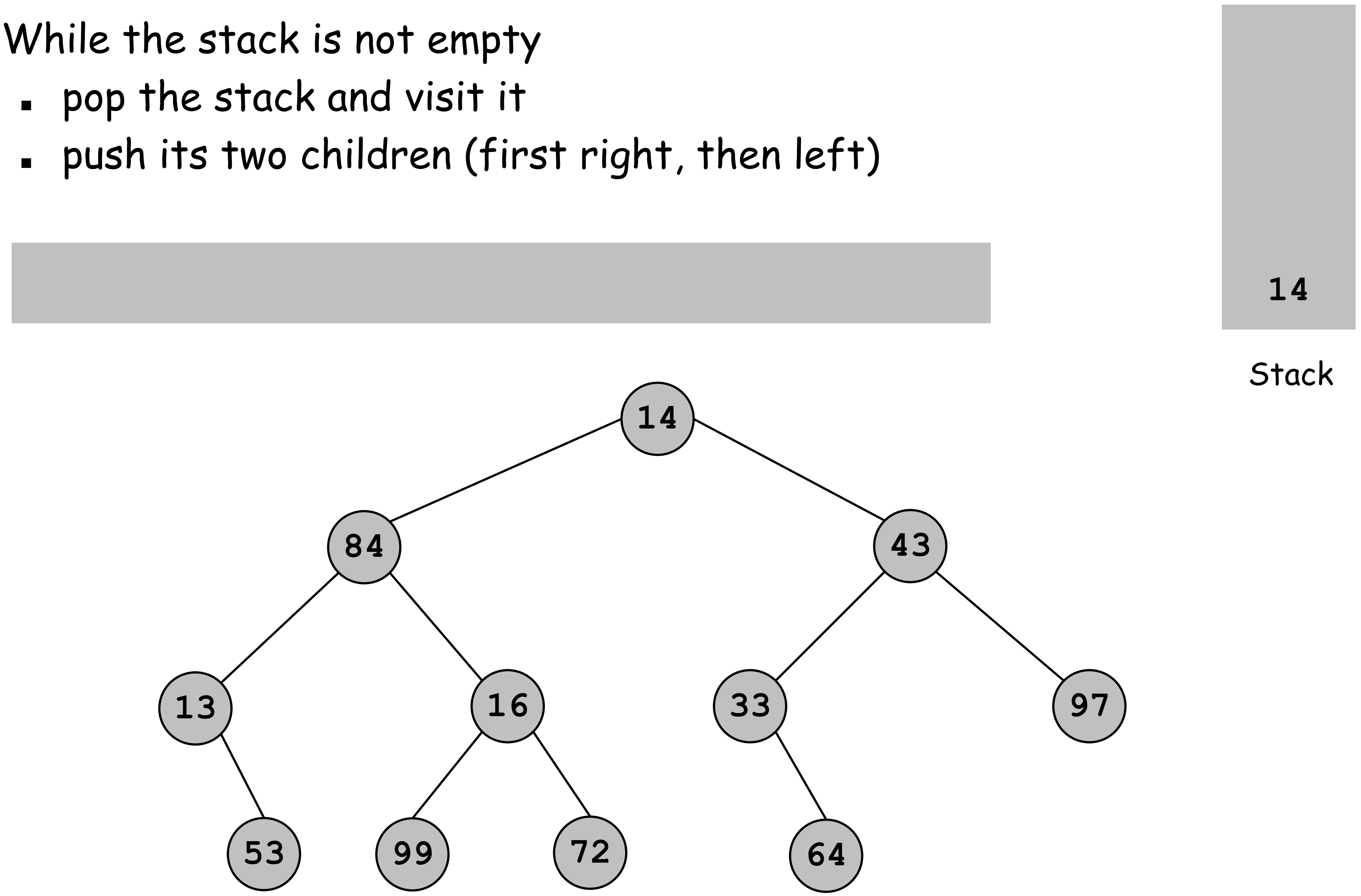
Nonrecursive Preorder Traversal

Preorder Traversal with a Stack

Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children (first right, then left)

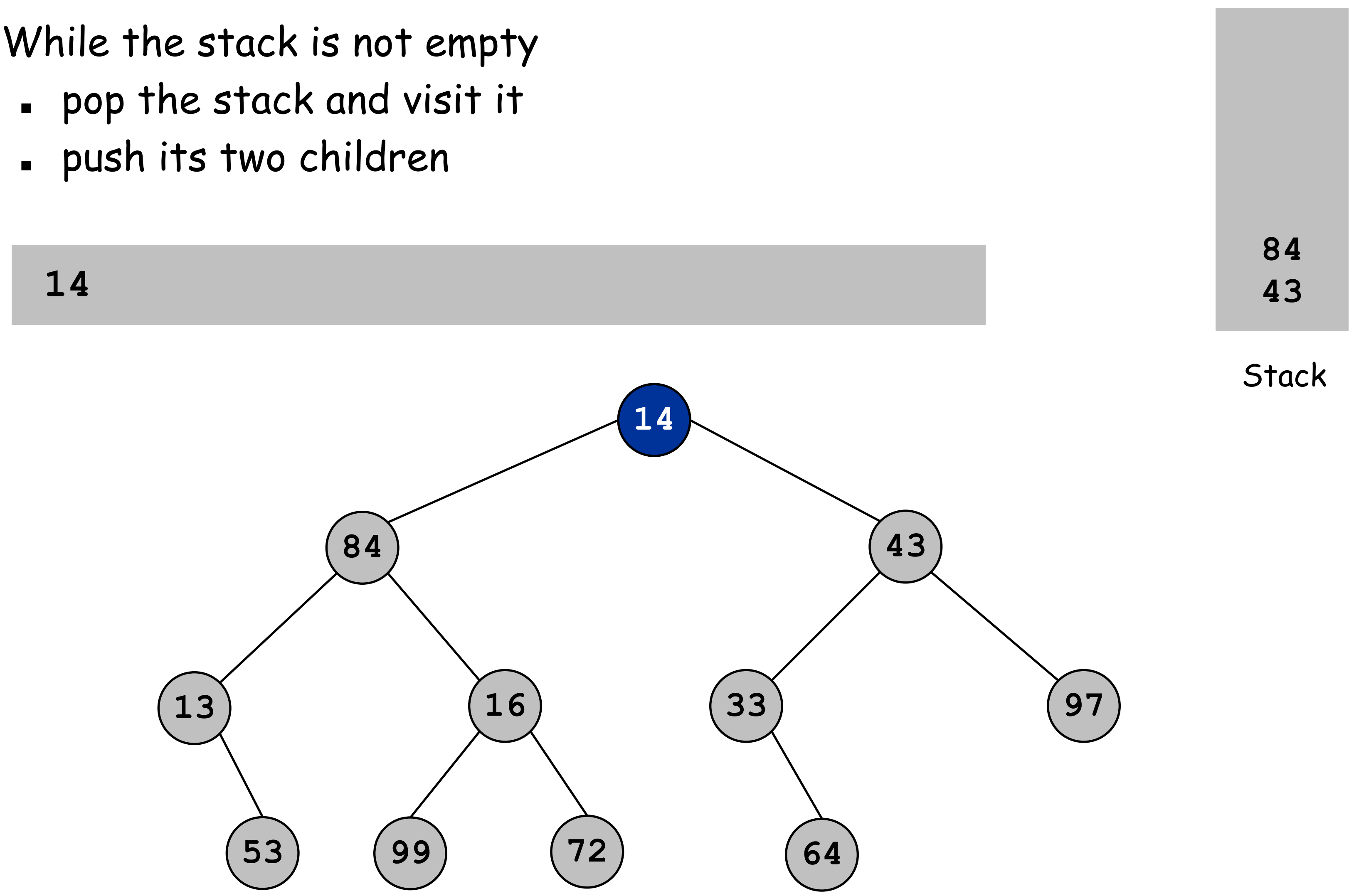


Preorder Traversal with a Stack

Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children



Preorder Traversal with a Stack

Push the root onto the stack.

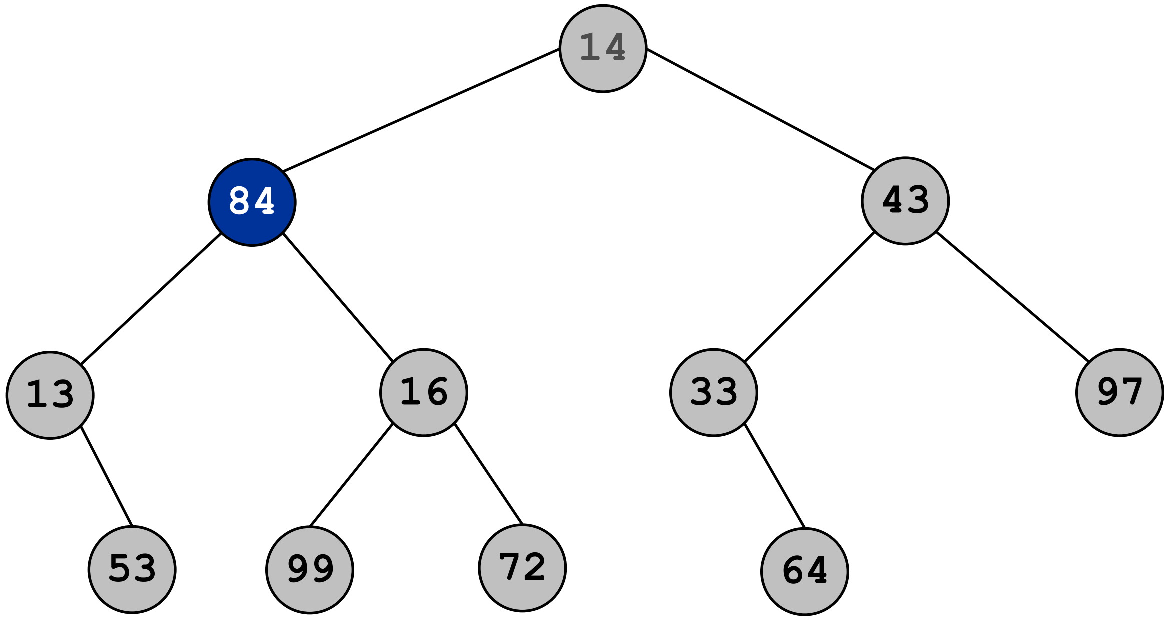
While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84
```

```
13
16
43
```

Stack



Preorder Traversal with a Stack

Push the root onto the stack.

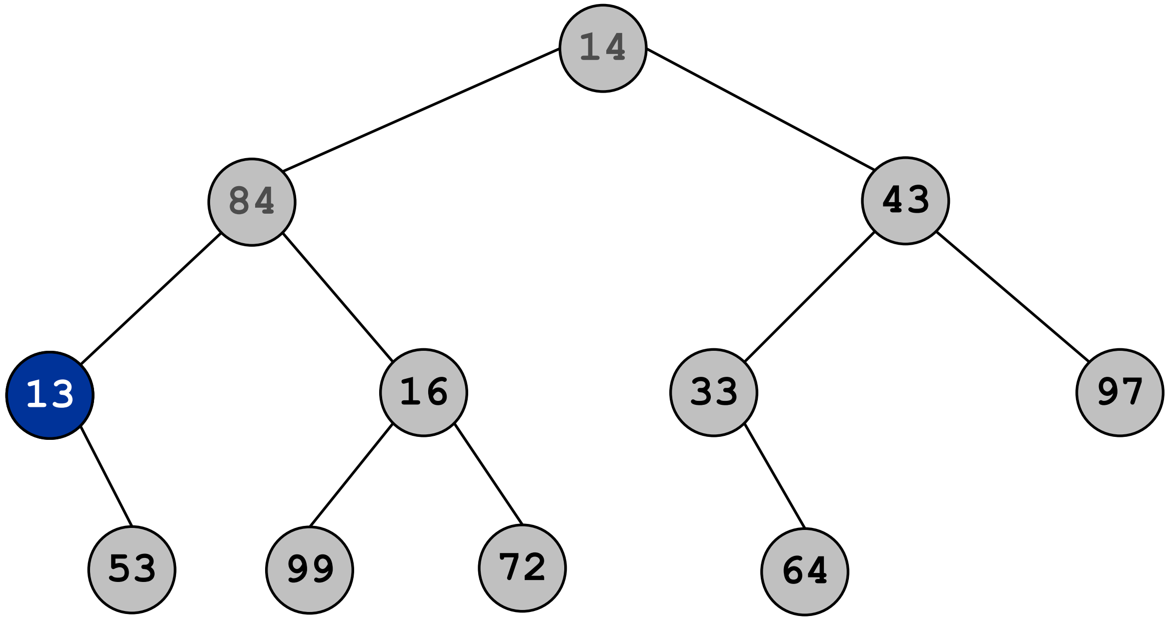
While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84 13
```

```
53
16
43
```

Stack



Preorder Traversal with a Stack

Push the root onto the stack.

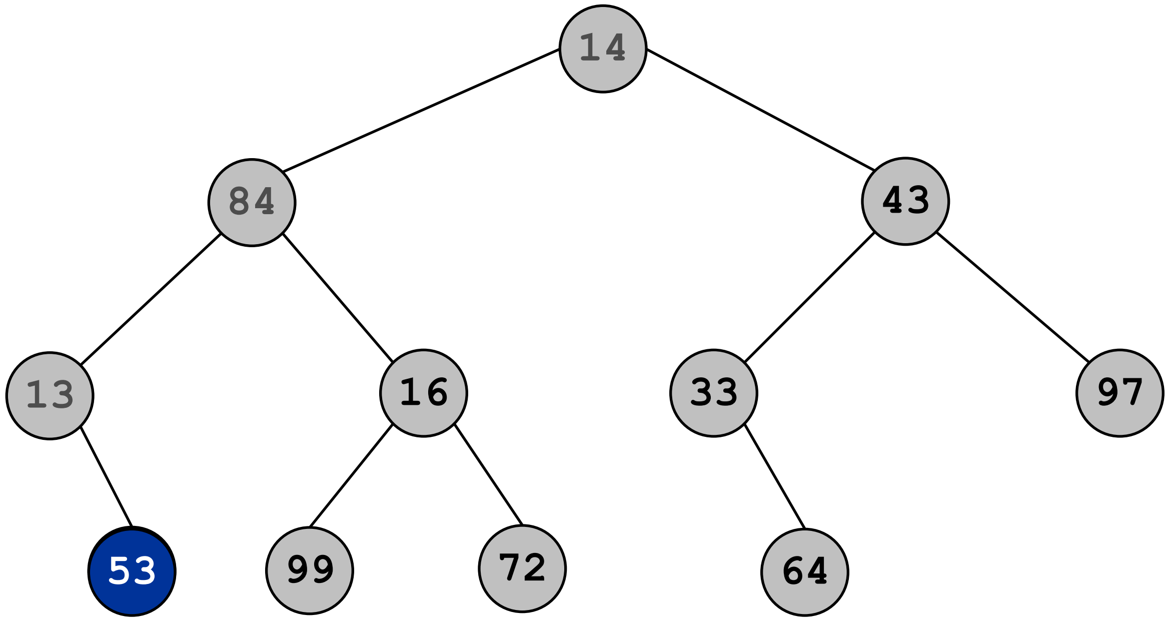
While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84 13 53
```

```
16
43
```

Stack



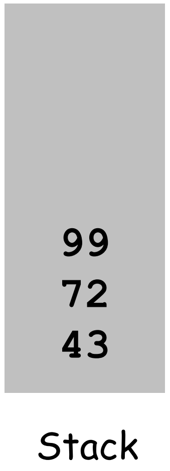
Preorder Traversal with a Stack

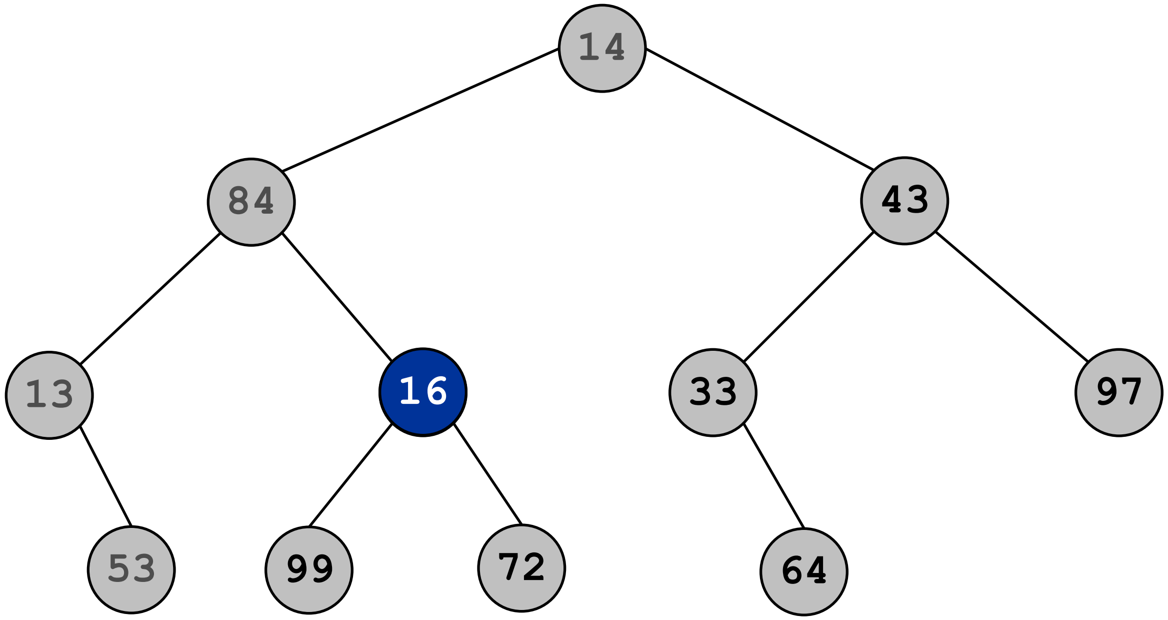
Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84 13 53 16
```





Preorder Traversal with a Stack

Push the root onto the stack.

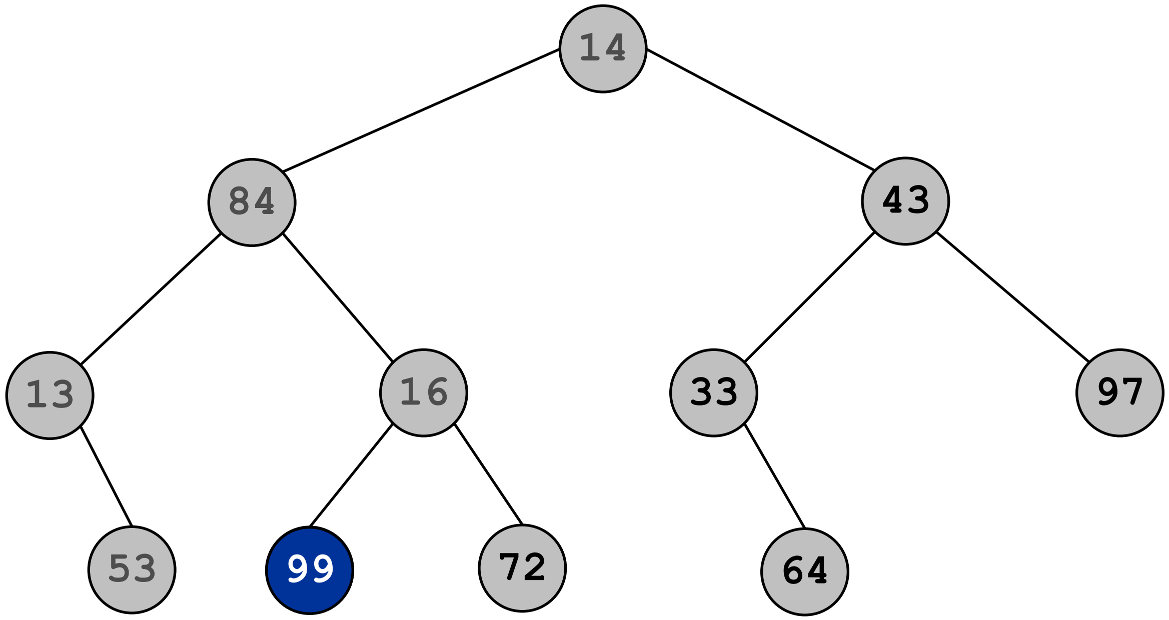
While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84 13 53 16 99
```

```
72
43
```

Stack



Preorder Traversal with a Stack

Push the root onto the stack.

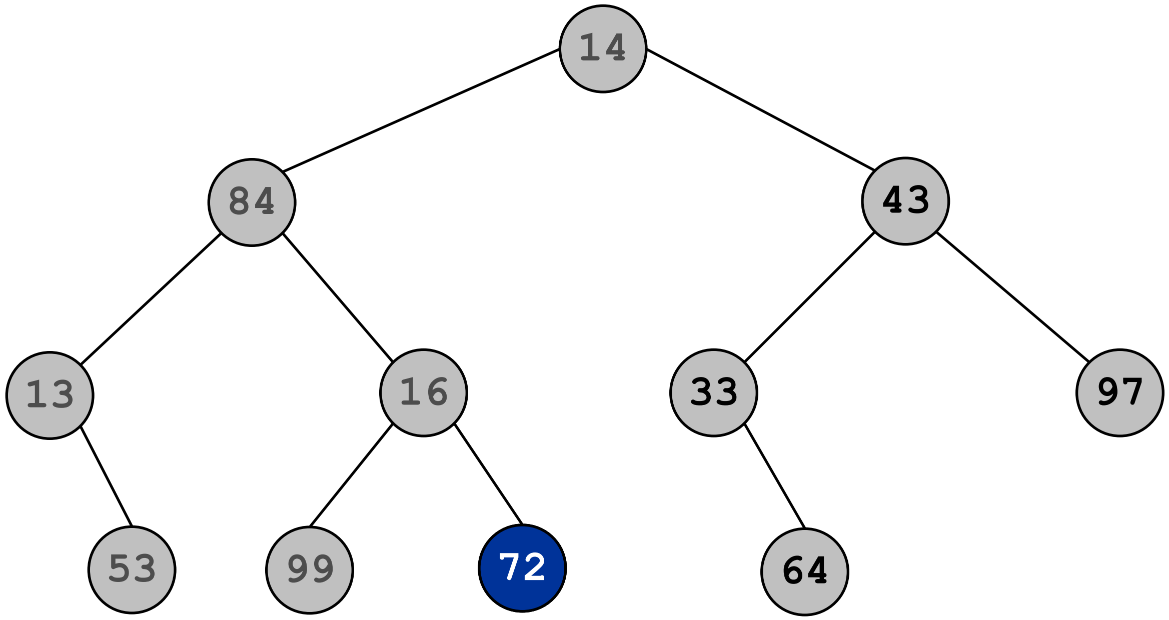
While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84 13 53 16 99 72
```

43

Stack



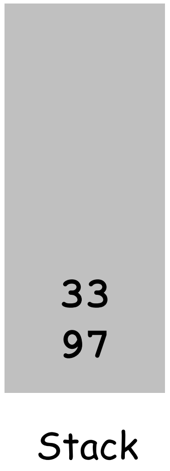
Preorder Traversal with a Stack

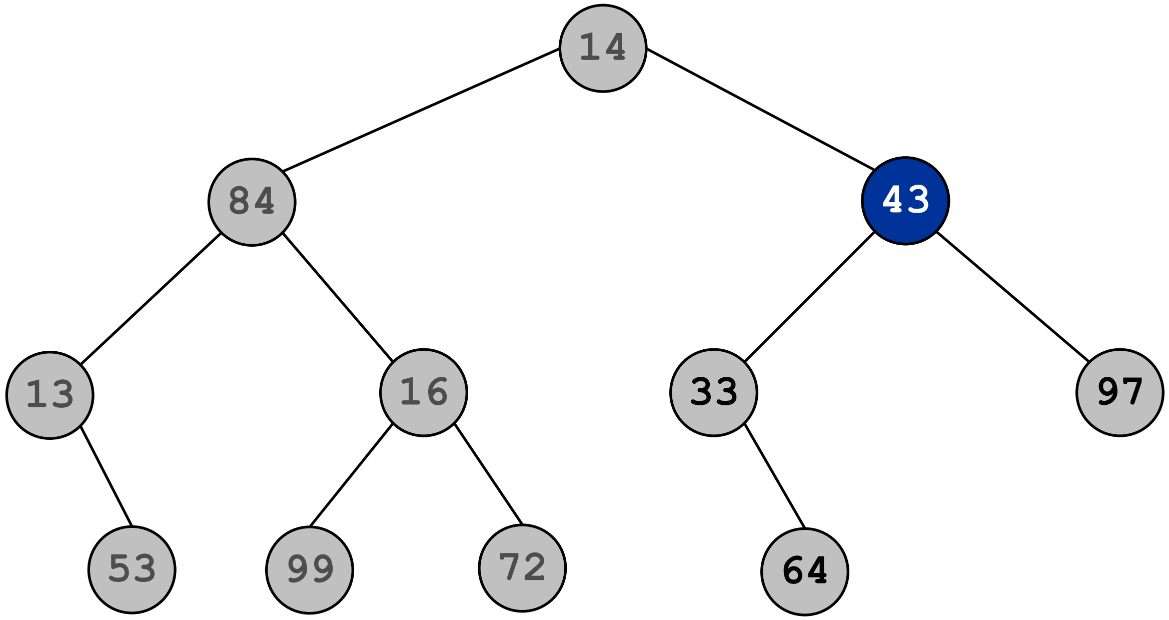
Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84 13 53 16 99 72 43
```





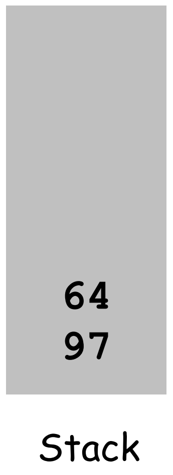
Preorder Traversal with a Stack

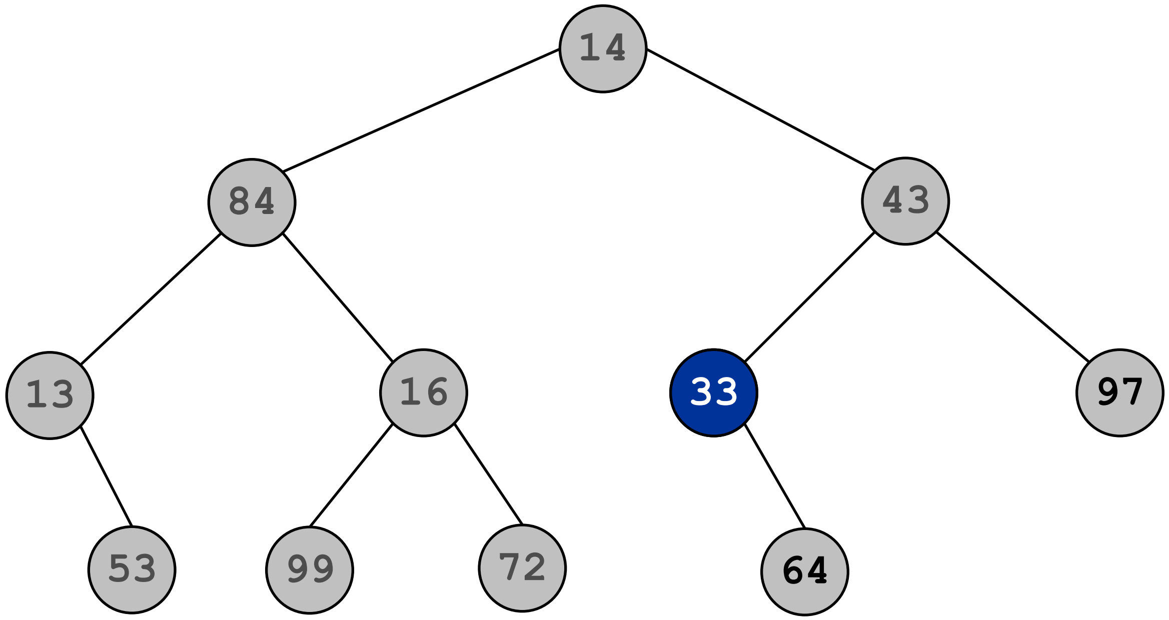
Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

14 84 13 53 16 99 72 43 33





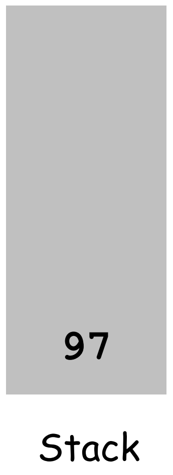
Preorder Traversal with a Stack

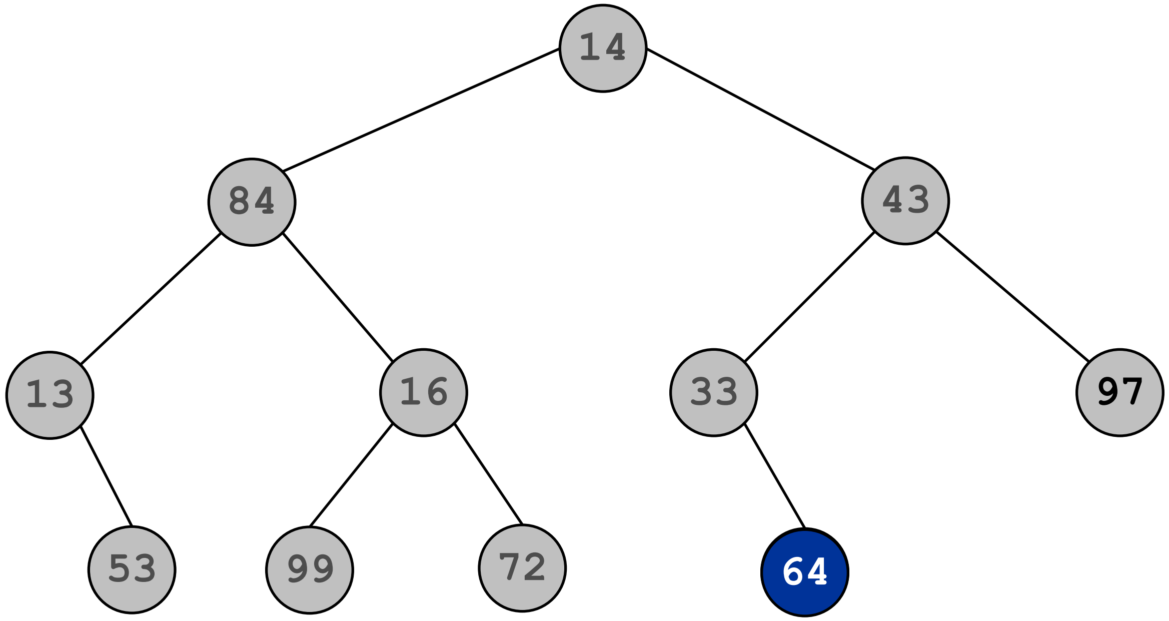
Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

14 84 13 53 16 99 72 43 33 64





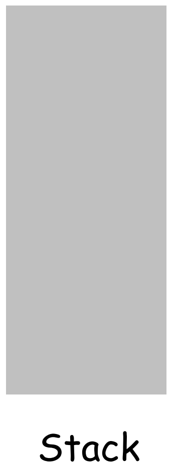
Preorder Traversal with a Stack

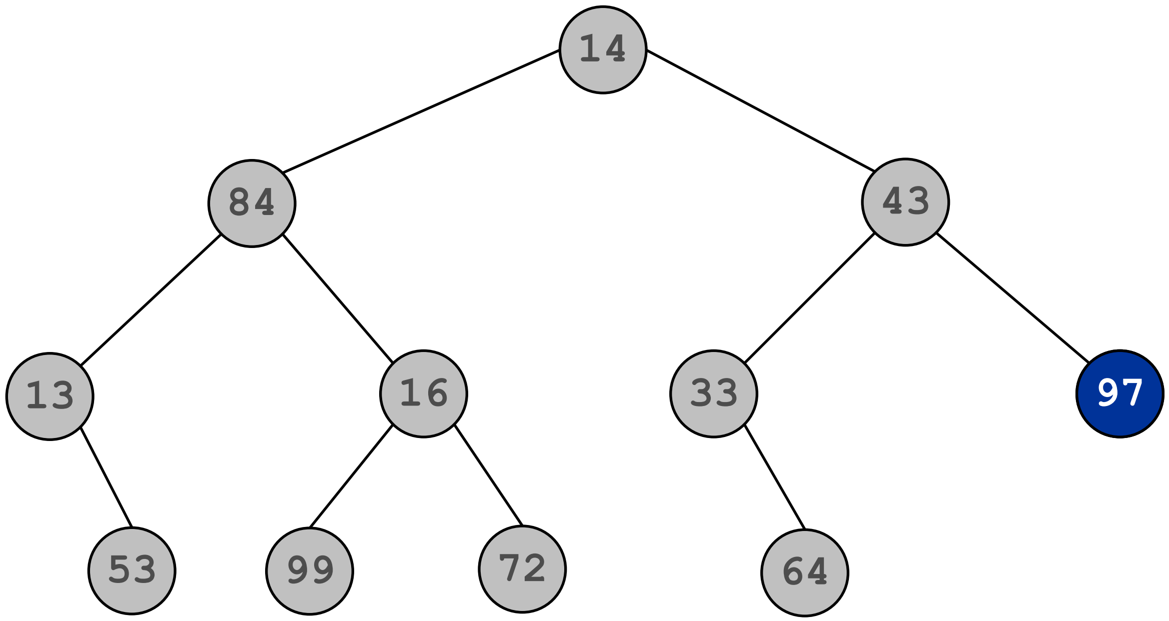
Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

```
14 84 13 53 16 99 72 43 33 64 97
```





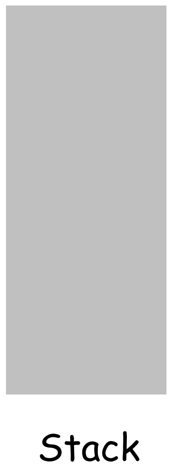
Preorder Traversal with a Stack

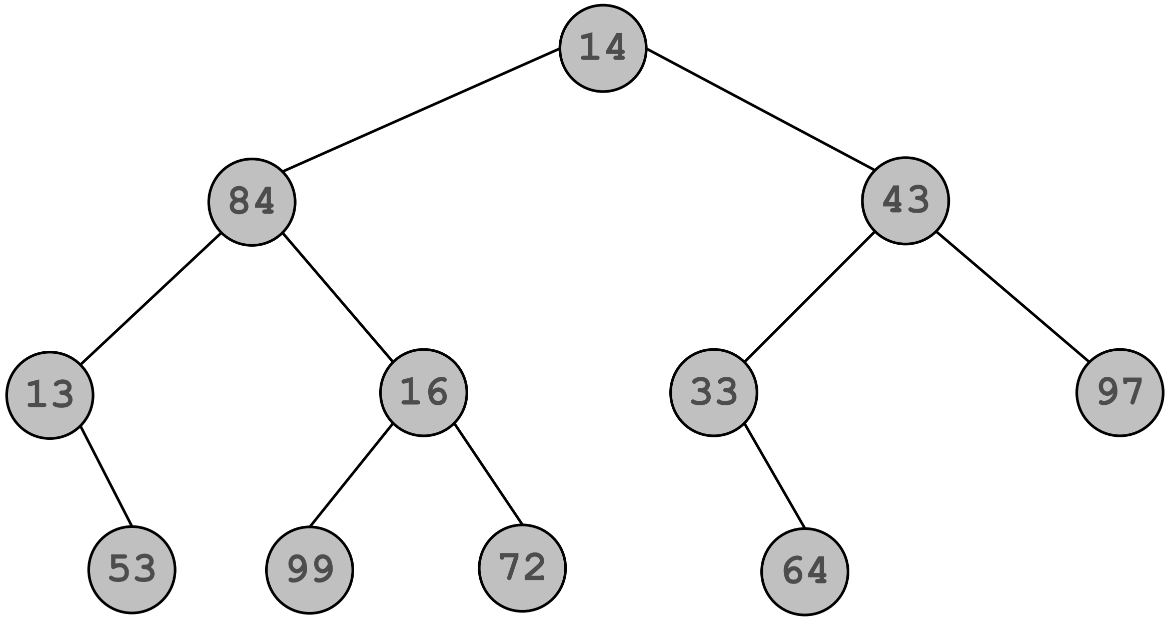
Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

14 84 13 53 16 99 72 43 33 64 97





Nonrecursive Inorder Traversal

- For each node, the left subtree is visited first, then the node, and then the right subtree

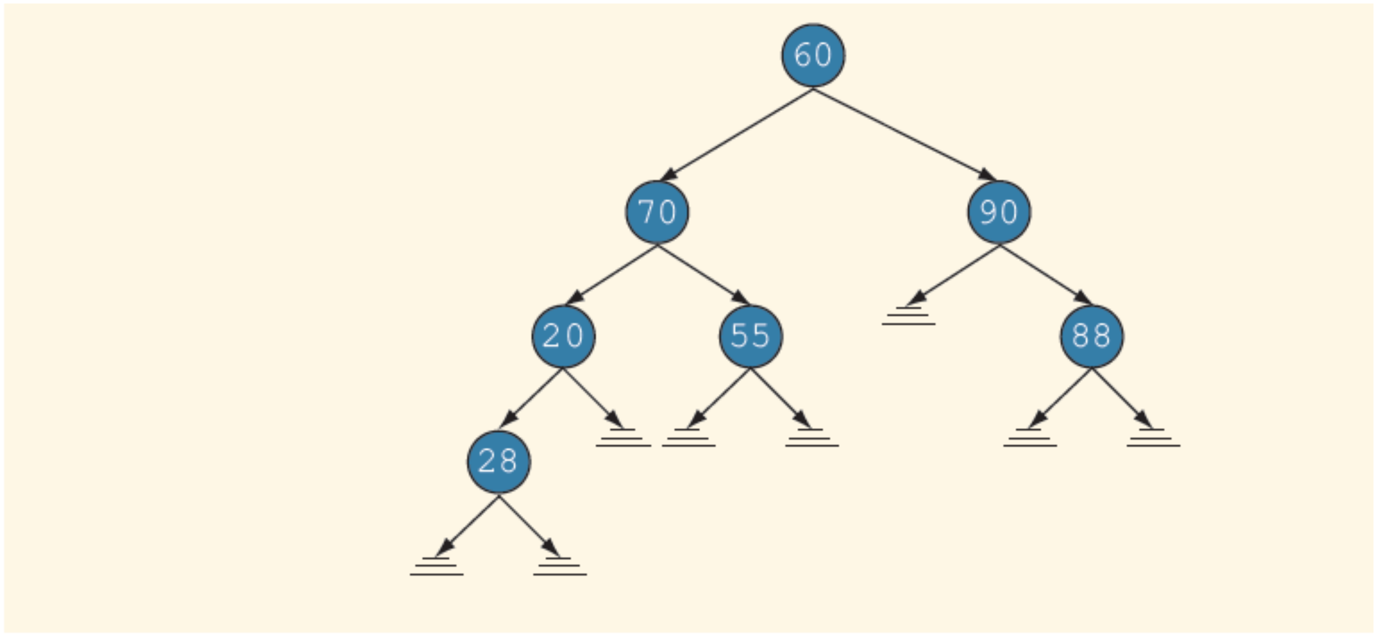


FIGURE 19-12 Binary tree; the leftmost node is 28

Non recursive Inorder

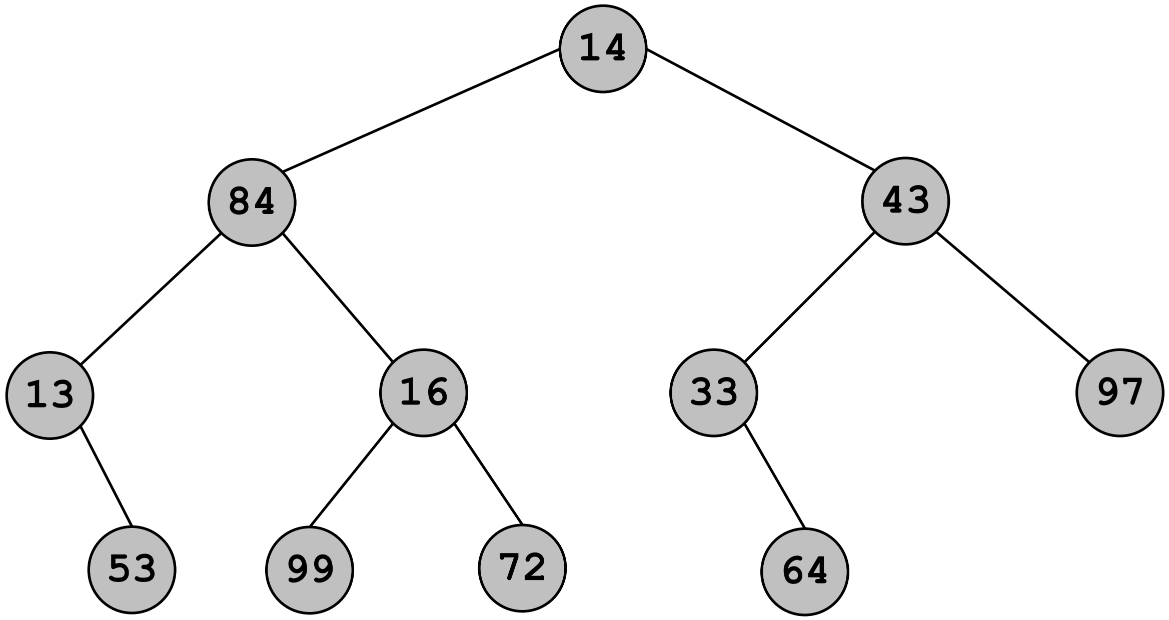
1) Create an empty stack S.
2) Push the root in stack
3) set current = current->left until current is NULL and push current
4) If current is NULL and stack is not empty then
    a) Pop the top item from stack and print
    b) Print the popped item, set current = current->right c) Go to step 3.
5) If current is NULL and stack is empty then we are done.

Inorder Traversal with a Stack

```
Push the root in stack
set current = current->left until current is NULL and push current
If current is NULL and stack is not empty then
    a) Pop the top item from stack and print
    b) set current = current->right c) Go to step 2
If current is NULL and stack is empty then we are done.
```

13
84
14

Stack



Inorder Traversal with a Stack

Push the root in stack

set current = current->left until current is NULL and push current

If current is NULL and stack is not empty then

  a) Pop the top item from stack and print

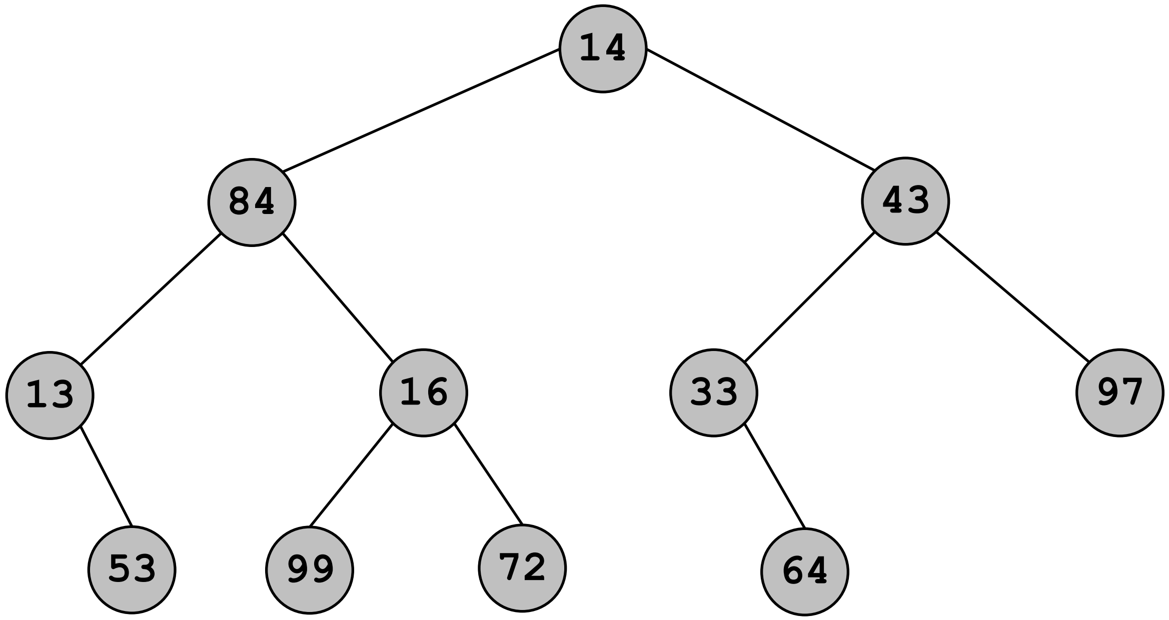
  b) set current = current->right c) Go to step 2

If current is NULL and stack is empty then we are done.

13

53
84
14

Stack



Inorder Traversal with a Stack

Push the root in stack

set current = current->left until current is NULL and push current

If current is NULL and stack is not empty then

    a) Pop the top item from stack and print

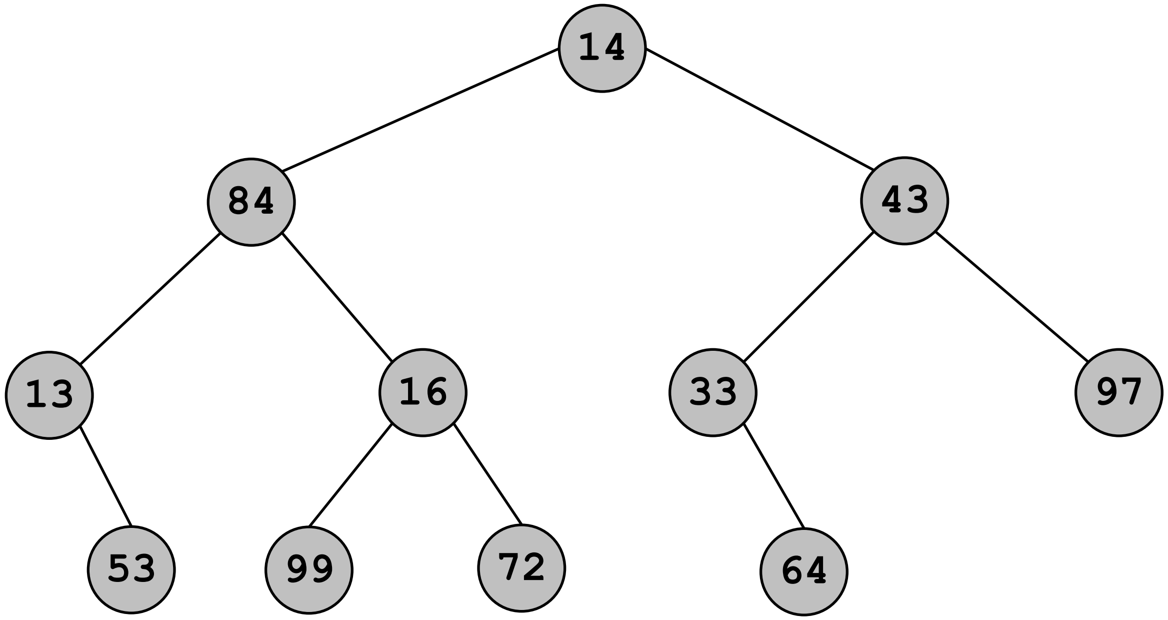
    b) set current = current->right c) Go to step 2

If current is NULL and stack is empty then we are done.

```
13 53
```

```
84
14
```

Stack



Inorder Traversal with a Stack

Push the root in stack

set current = current->left until current is NULL and push current

If current is NULL and stack is not empty then

a) Pop the top item from stack and print

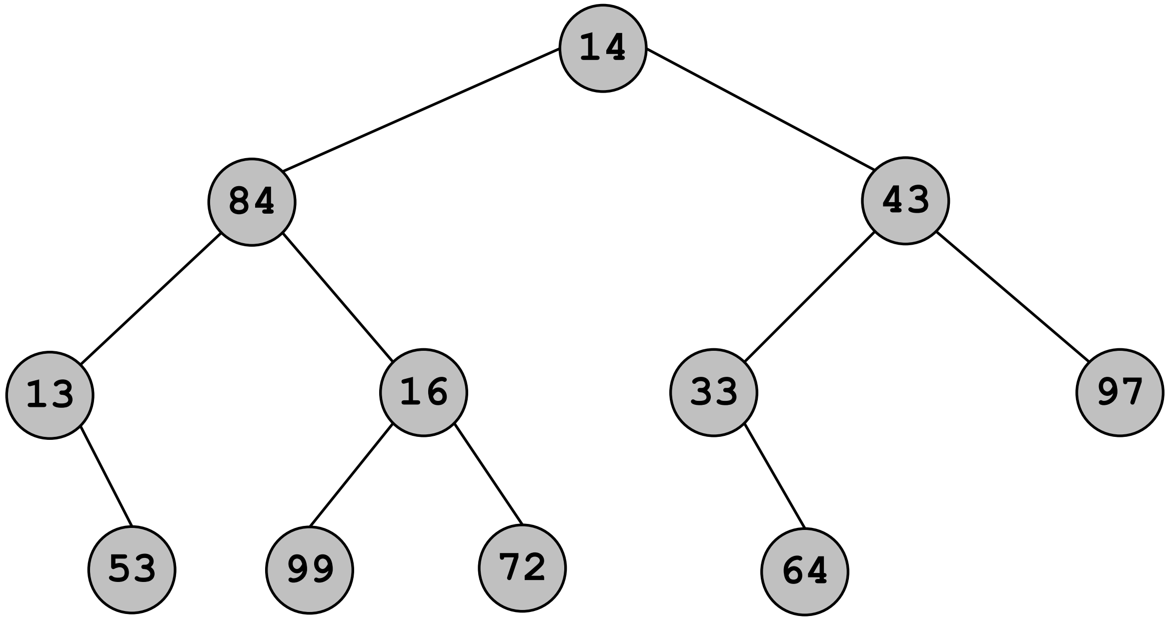
b) set current = current->right c) Go to step 2

If current is NULL and stack is empty then we are done.

```
13 53 84
```

```
99
16
14
```

Stack



Inorder Traversal with a Stack

Push the root in stack

set current = current->left until current is NULL and push current

If current is NULL and stack is not empty then

a) Pop the top item from stack and print

b) set current = current->right c) Go to step 2

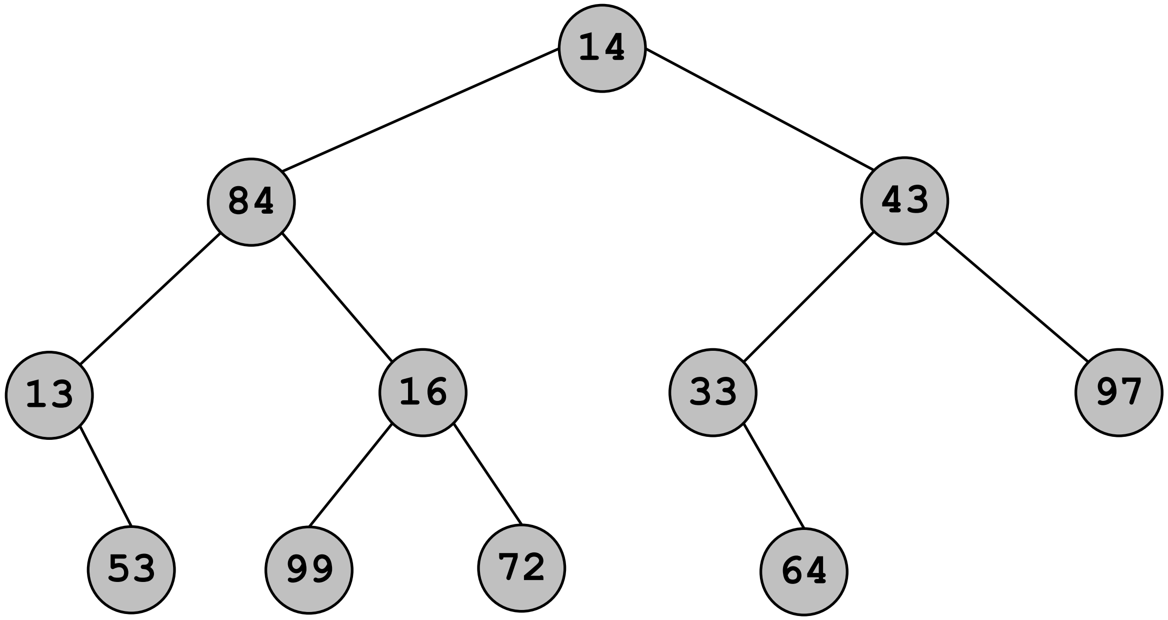
If current is NULL and stack is empty then we are done.

```
13 53 84 99 16
```

72

14

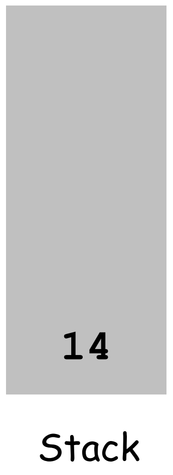
Stack

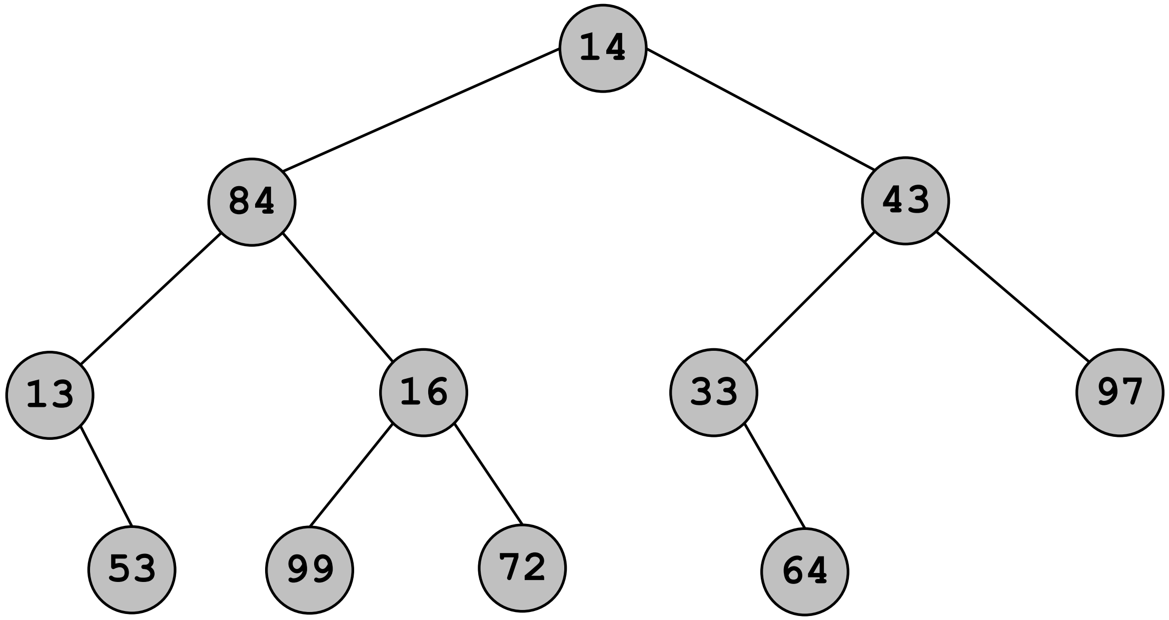


Inorder Traversal with a Stack

```
Push the root in stack
set current = current->left until current is NULL and push current
If current is NULL and stack is not empty then
    a) Pop the top item from stack and print
    b) set current = current->right c) Go to step 2
If current is NULL and stack is empty then we are done.
```

```
13 53 84 99 16 72
```

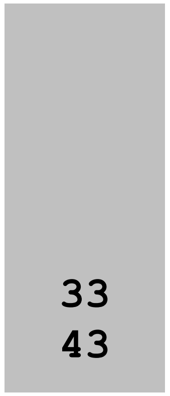




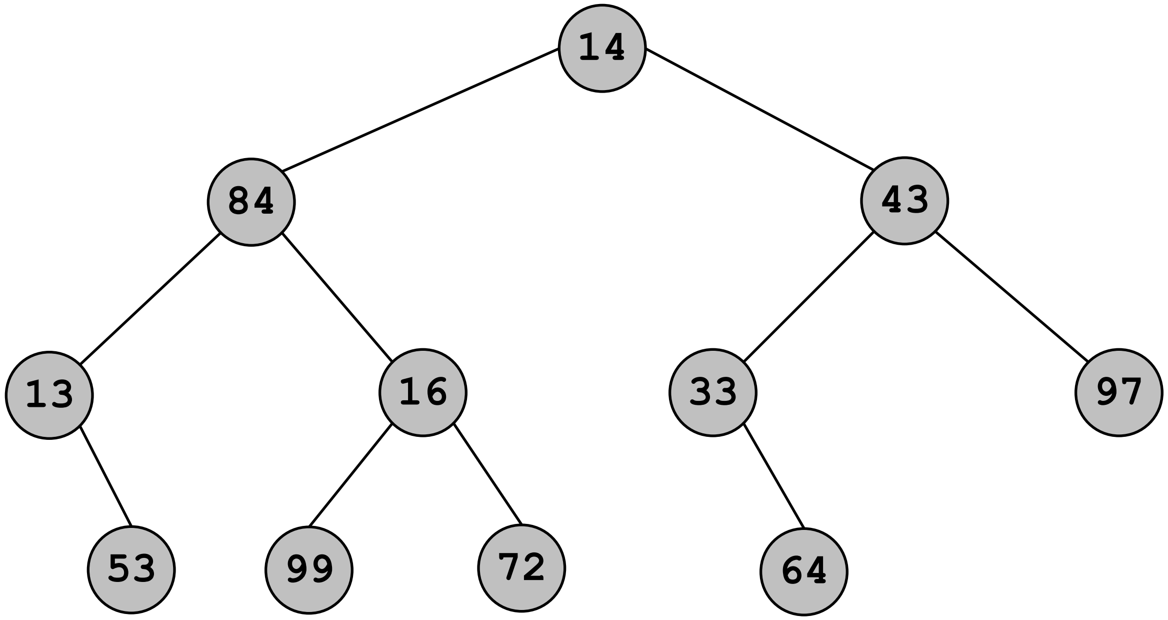
Inorder Traversal with a Stack

```
Push the root in stack
set current = current->left until current is NULL and push current
If current is NULL and stack is not empty then
    a) Pop the top item from stack and print
    b) set current = current->right c) Go to step 2
If current is NULL and stack is empty then we are done.
```

```
13 53 84 99 16 72 14
```



Stack



Preorder Traversal with a Stack

Push the root onto the stack.

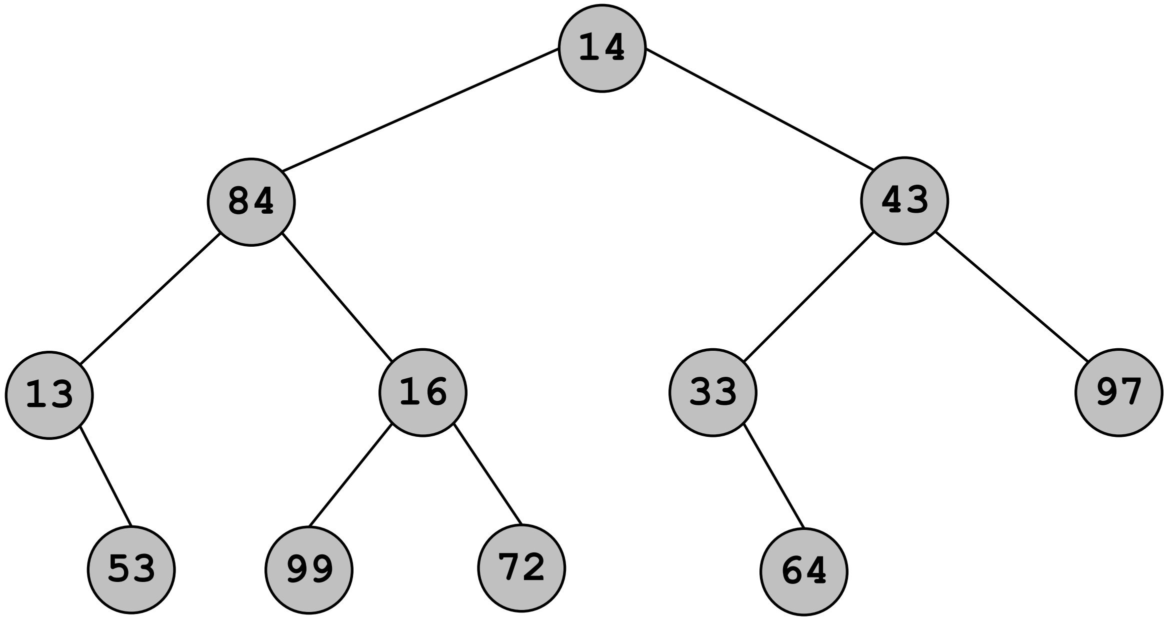
While the stack is not empty

- pop the stack and visit it
- push its two children (first right, then left)

13 53 84 99 16 72 14 33

64
43

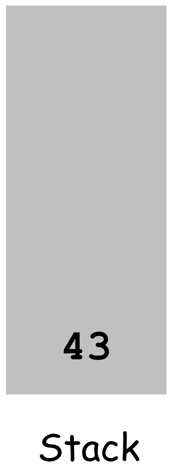
Stack

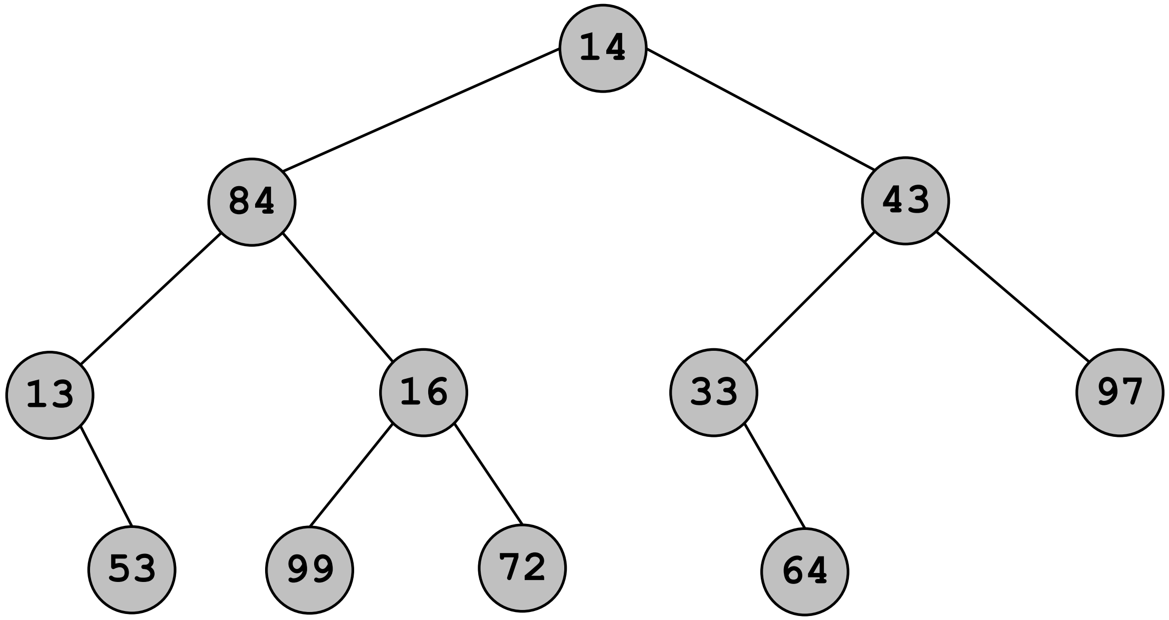


Inorder Traversal with a Stack

```
Push the root in stack
set current = current->left until current is NULL and push current
If current is NULL and stack is not empty then
    a) Pop the top item from stack and print
    b) set current = current->right c) Go to step 2
If current is NULL and stack is empty then we are done.
```

```
13 53 84 99 16 72 14 33 64
```





Inorder Traversal with a Stack

Push the root in stack

set current = current->left until current is NULL and push current

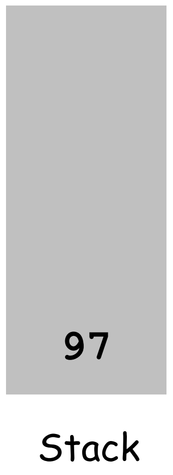
If current is NULL and stack is not empty then

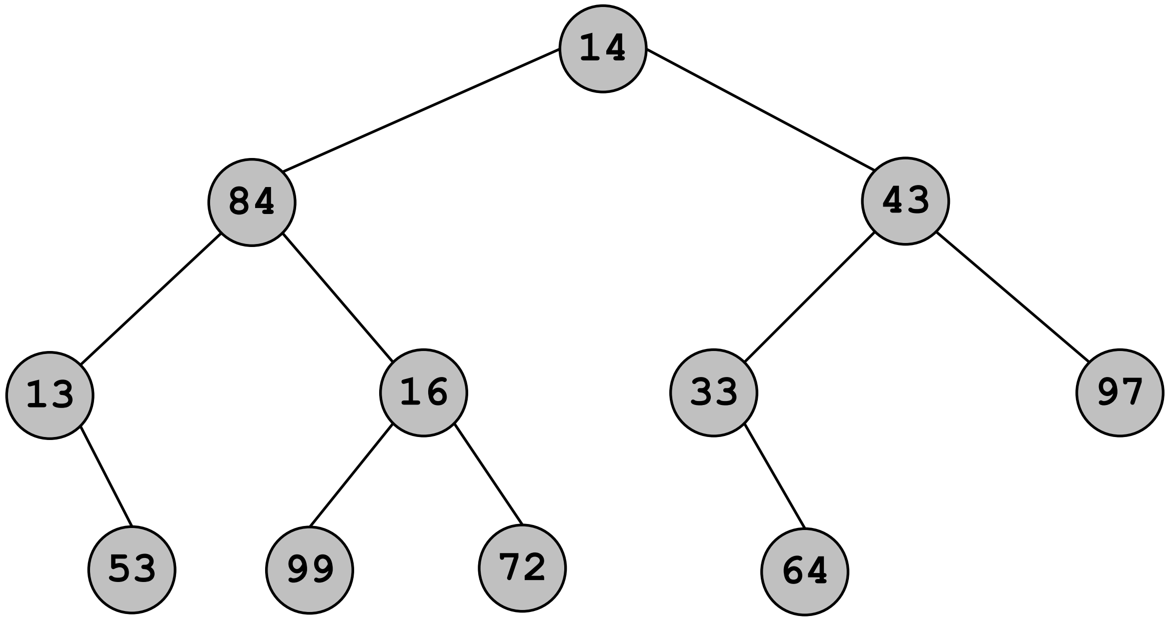
  a) Pop the top item from stack and print

  b) set current = current->right c) Go to step 2

If current is NULL and stack is empty then we are done.

```
13 53 84 99 16 72 14 33 64 43
```

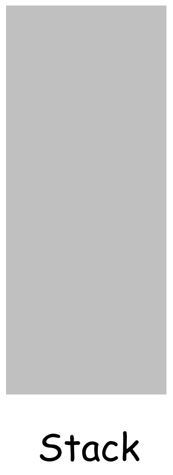


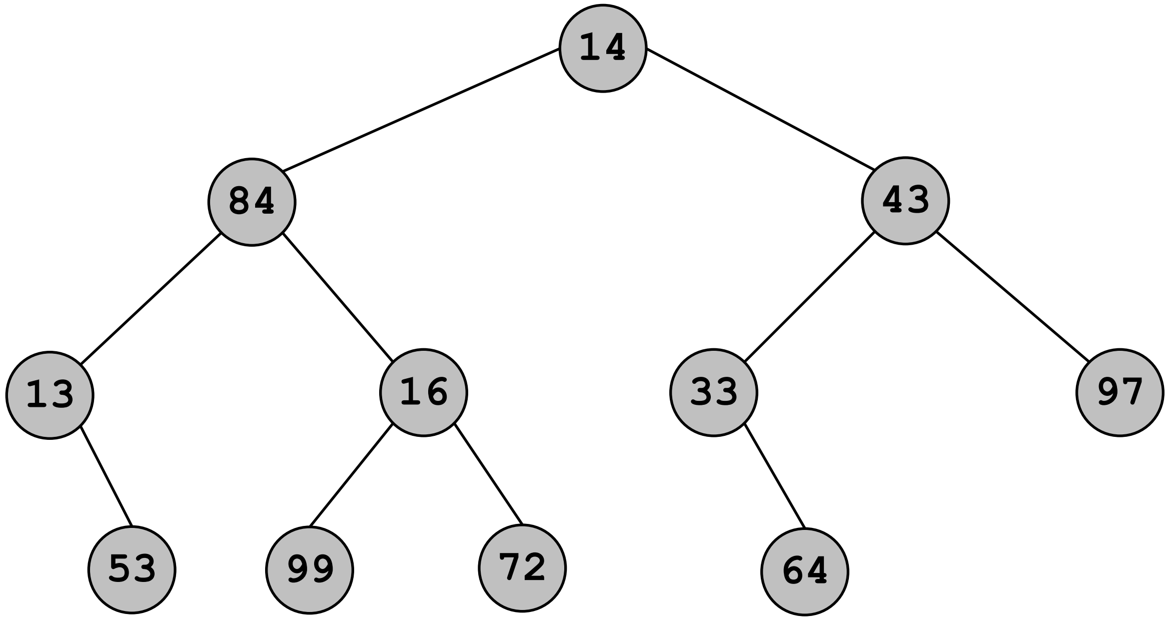


Inorder Traversal with a Stack

```
Push the root in stack
set current = current->left until current is NULL and push current
If current is NULL and stack is not empty then
    a) Pop the top item from stack and print
    b) set current = current->right c) Go to step 2
If current is NULL and stack is empty then we are done.
```

```
13 53 84 99 16 72 14 33 64 43 97
```



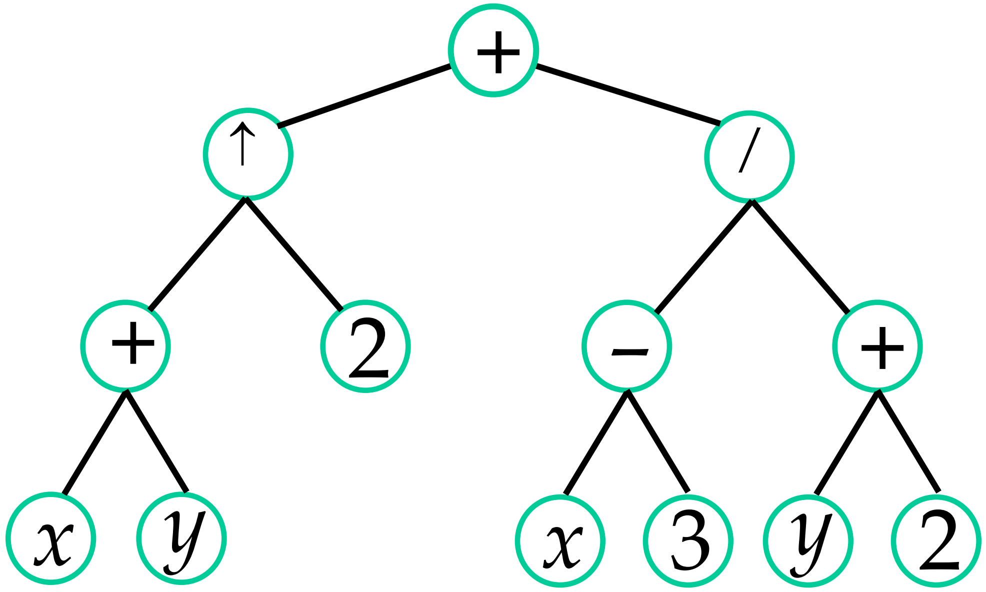


Representing Arithmetic Expressions

- Complicated arithmetic expressions can be represented by an ordered rooted tree
  - Internal vertices represent operators
  - Leaves represent operands
- Build the tree bottom-up
  - Construct smaller subtrees
  - Incorporate the smaller subtrees as part of larger subtrees

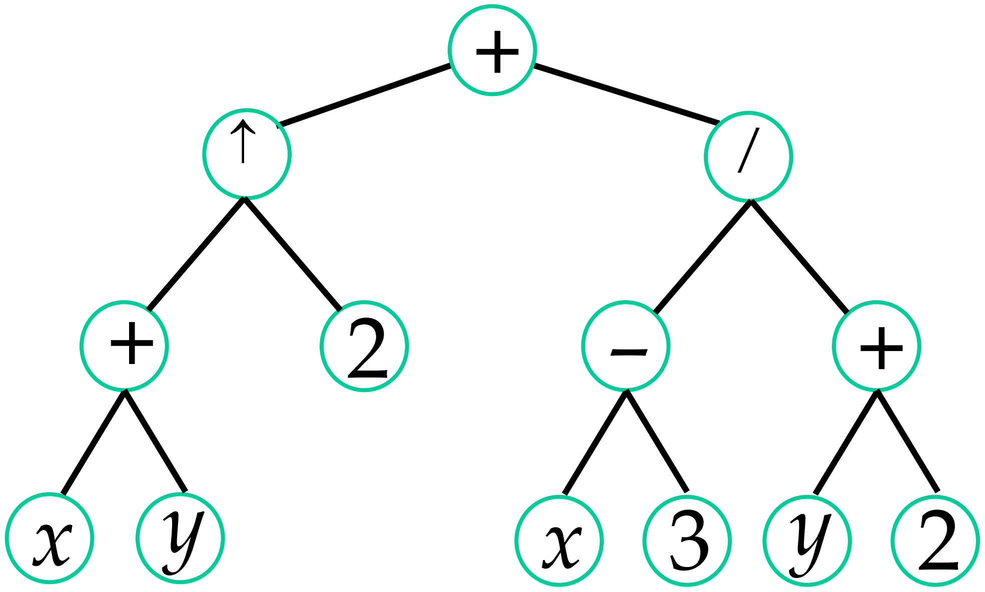
Example

$$(x+y)^2 + (x-3)/(y+2)$$



Infix Notation

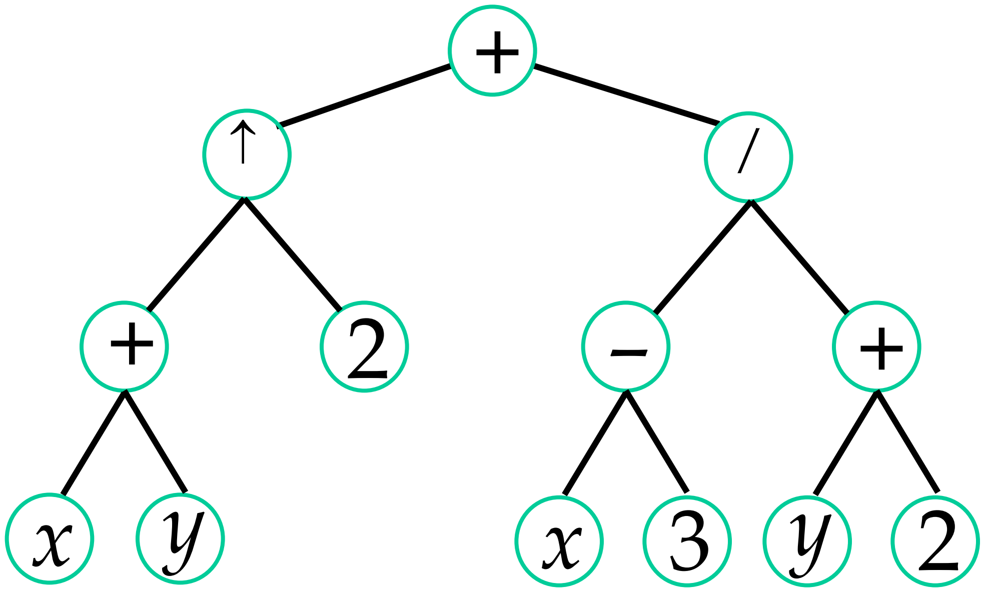
- Traverse in inorder (LVR) adding parentheses for each operation



$$(((x + y) \uparrow 2) + ((x - 3) / (y + 2)))$$

Prefix Notation

- Traverse in preorder (VLR)



$+ \ \uparrow \ + \ x \ y \ 2 \ / \ - \ x \ 3 \ + \ y \ 2$

Evaluating Prefix Notation

- In an prefix expression, a binary operator precedes its two operands
- The expression is evaluated right-left
- Look for the first operator from the right
- Evaluate the operator with the two operands immediately to its right

Example

$+ \ / \ + \ 2 \ 2 \ 2 \ / \ - \ 3 \ 2 \ \boxed{+ \ 1 \ 0}$

$+ \ / \ + \ 2 \ 2 \ 2 \ / \ \boxed{- \ 3 \ 2} \ 1$

$+ \ / \ + \ 2 \ 2 \ 2 \ \boxed{/ \ 1 \ 1}$

$+ \ / \ \boxed{+ \ 2 \ 2} \ 2 \ 1$

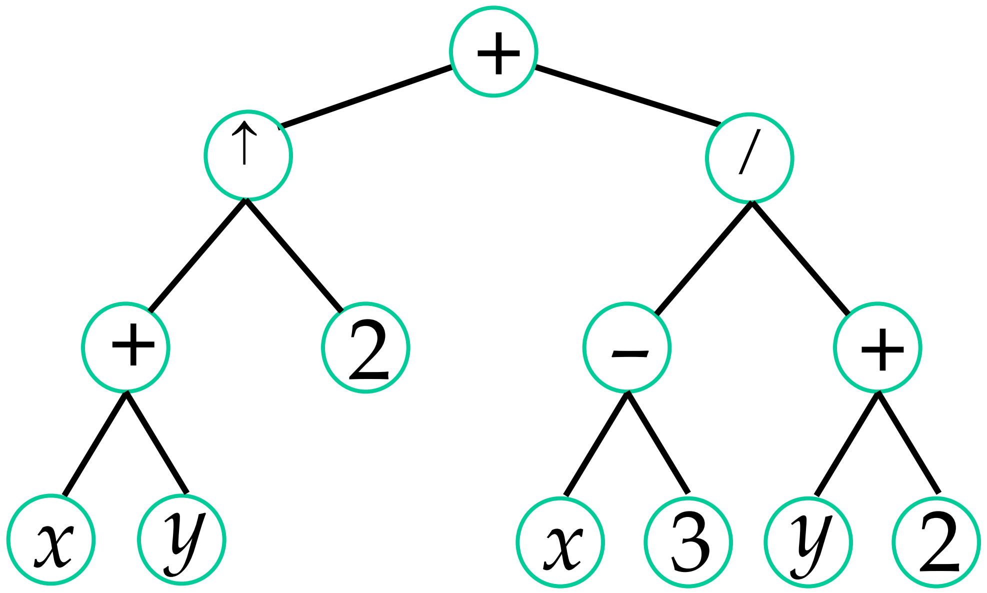
$+ \ \boxed{/ \ 4 \ 2} \ 1$

$\boxed{+ \ 2 \ 1}$

3

Postfix Notation (Reverse Polish)

- Traverse in postorder (LRV)



$x \ \ y \ + \ 2 \ \uparrow \ x \ \ 3 \ - \ y \ \ 2 \ + \ / \ +$

Evaluating Postfix Notation

- In an postfix expression, a binary operator follows its two operands
- The expression is evaluated left-right
- Look for the first operator from the left
- Evaluate the operator with the two operands immediately to its left

Example

$$\boxed{2\ 2\ +}\ 2\ /\ 3\ 2\ -\ 1\ 0\ +\ /\ +$$

$$\boxed{4\ 2\ /}\ 3\ 2\ -\ 1\ 0\ +\ /\ +$$

$$2\ \boxed{3\ 2\ -}\ 1\ 0\ +\ /\ +$$

$$2\ 1\ \boxed{1\ 0\ +}\ /\ +$$

$$2\ \boxed{1\ 1\ /}\ +$$

$$\boxed{2\ 1\ +}$$

$$3$$