University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Spring 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming deadlines:
  - Lab 3 is due on 2/11
  - Homework 4 is due on 2/14

Previous lecture …

- How to add and delete from a Red-Black Binary Search Tree (BST)

CourseMIRROR Reflections

Today …

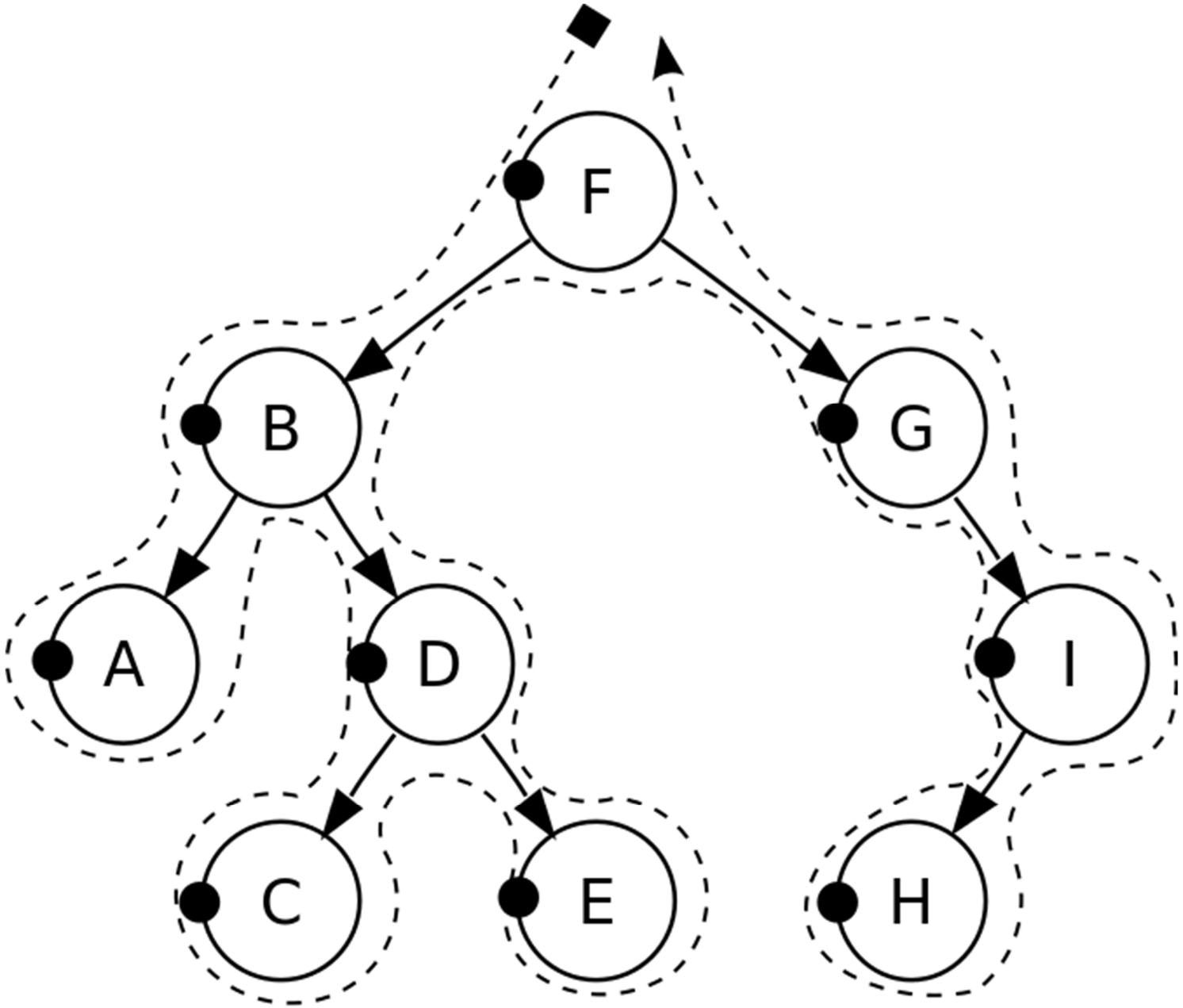
- How to traverse a Binary Tree
  - General Binary Tree
    - Pre-order, in-order, post-order
  - Binary Search Tree
    - including Red-Black BST

Traversals of a General Binary Tree

- Preorder traversal
  - Visit root before we visit root's subtree

Pre-order traversal

- F
- B
- A
- D
- C
- E
- G
- I
- H



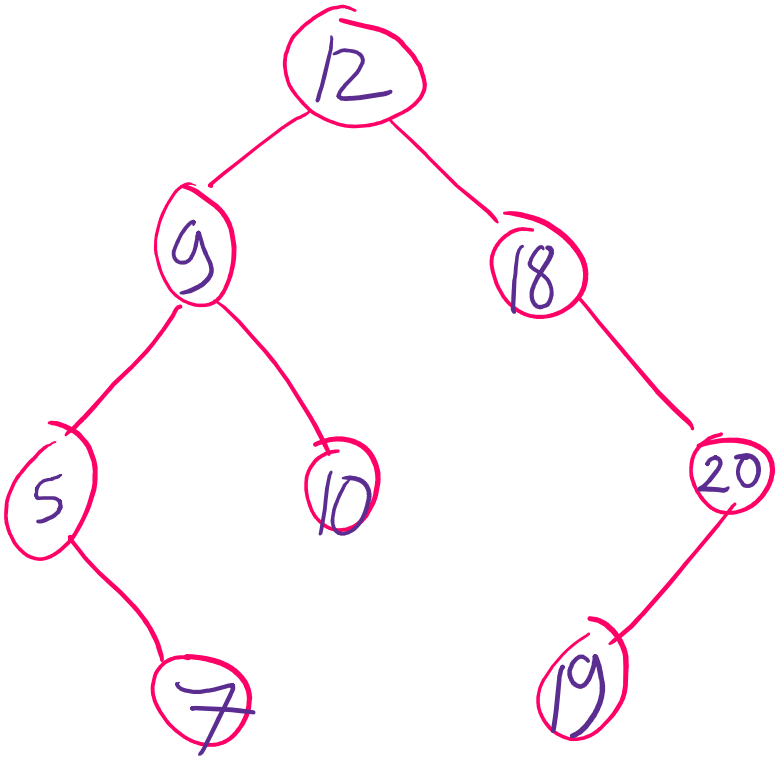
Pre-order traversal implementation

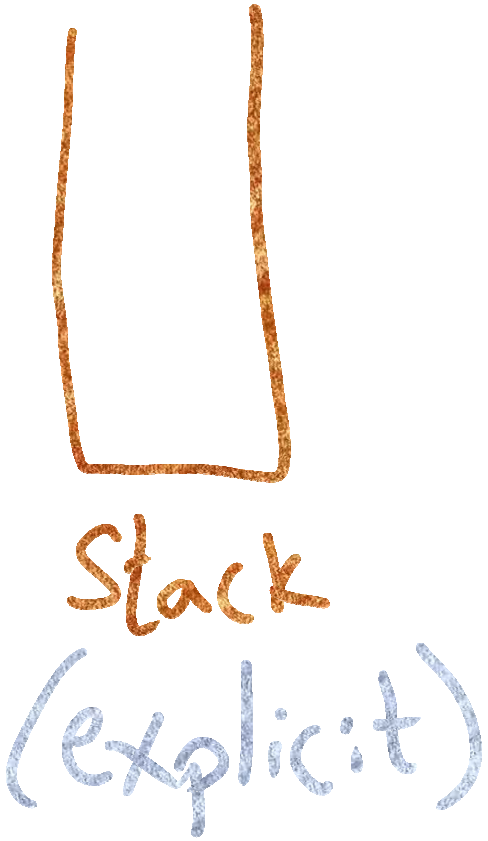
```
void traverse(BinaryNode<T> root){

   if(root != null){

      System.out.println(root.data);

      traverse(root.left);

      traverse(root.right);

    }

}
```

Why use iteration instead of recursion?

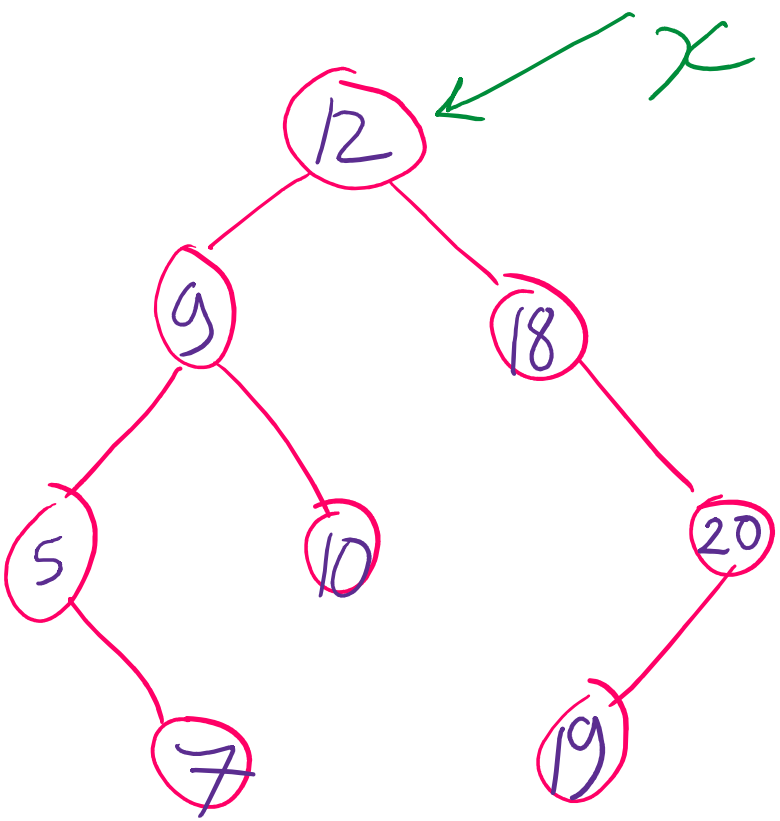
- Iteration is faster than recursion
  - No function call overhead (stack allocation)
- Especially useful for frequently used operations
  - Search is a good example of these operations

Iterative Preorder traversal





Iterative Preorder traversal



Stack (explicit)

while (stack not empty)

pop

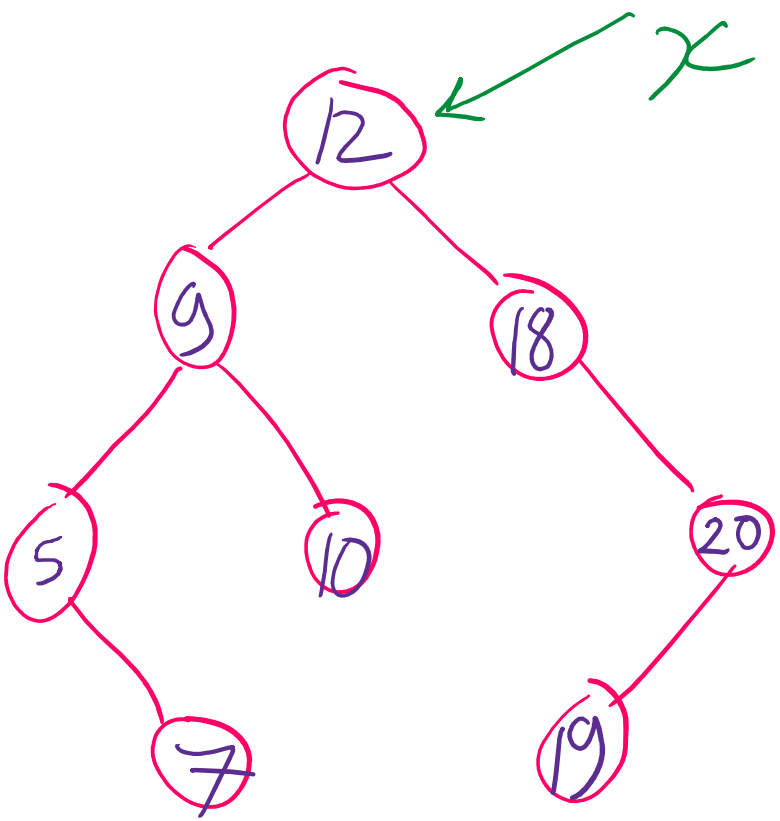
visit

Push right

Push left

Visit order : 12

Iterative Preorder traversal



Stack (explicit)

while (stack not empty)

pop

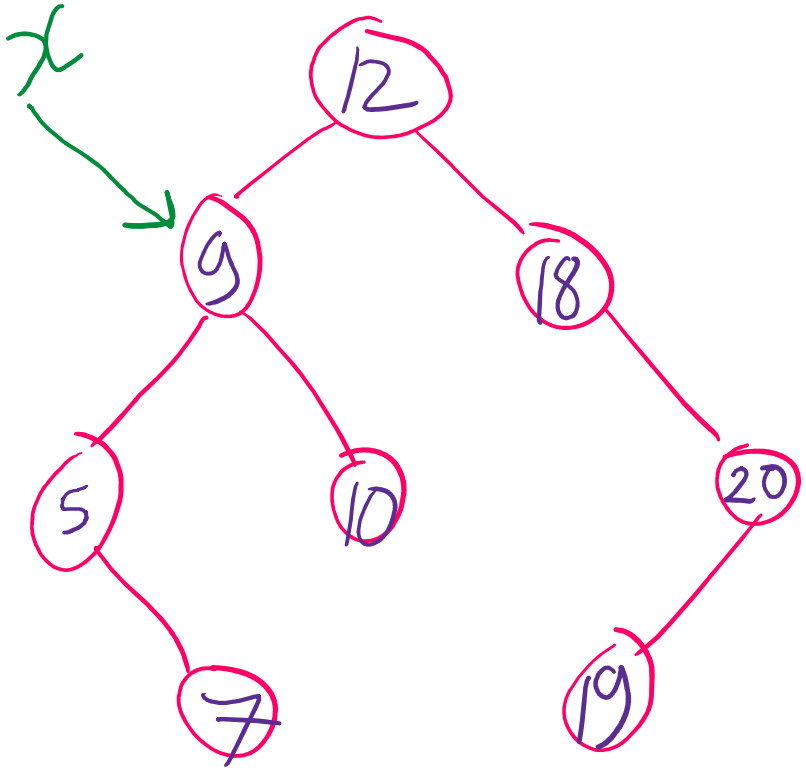
visit

Push right

Push left

Visit order : 12

Iterative Preorder traversal



Stack (explicit)

while (stack not empty)

pop

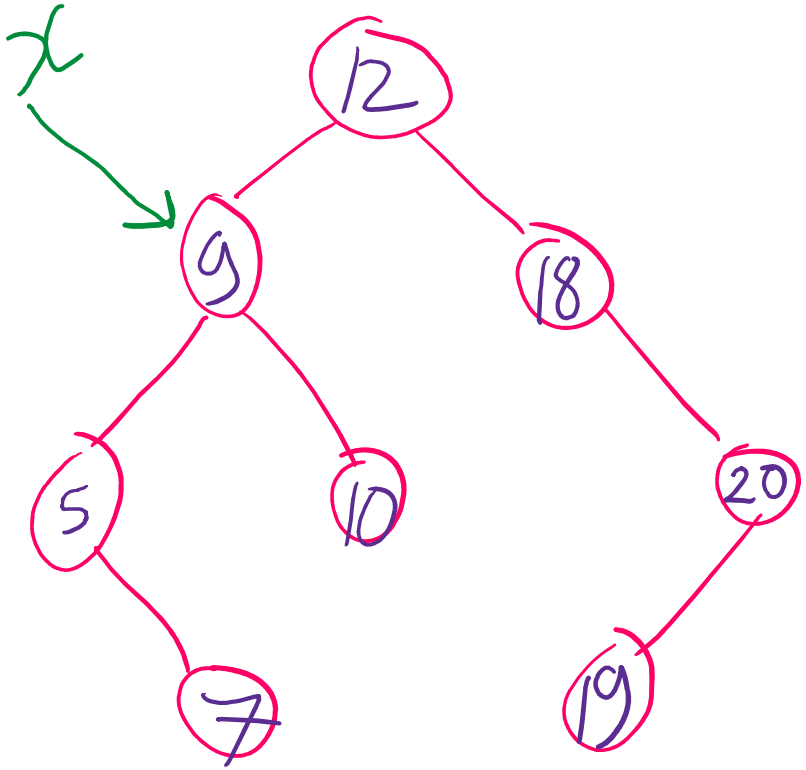
visit

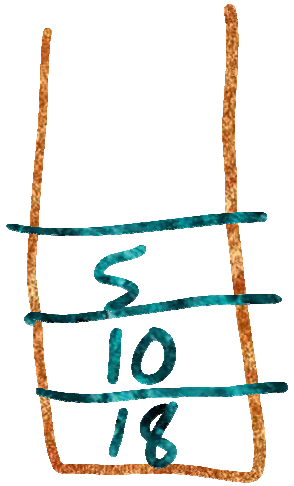
push right

push left

Visit order: 12, 9

Iterative Preorder traversal





Stack (explicit)

while (stack not empty)

pop

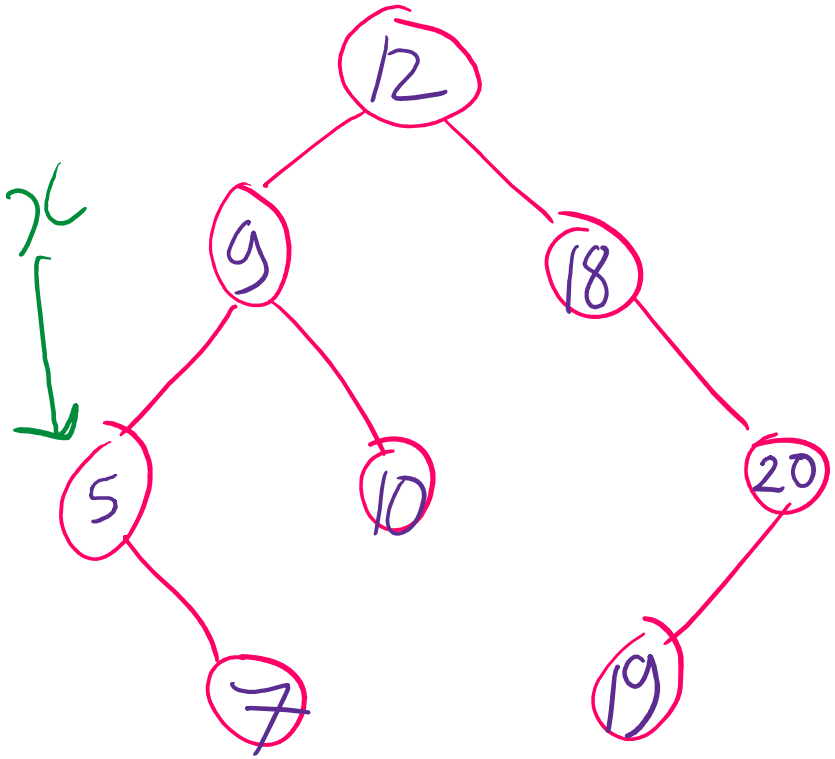
visit

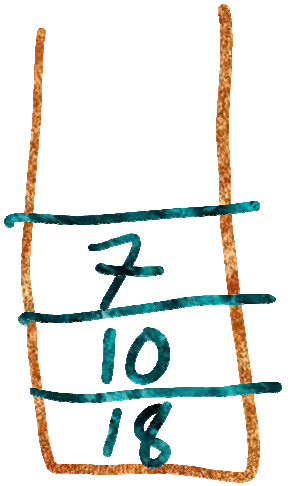
Push right

Push left

Visit order: 12, 9

Iterative Preorder traversal





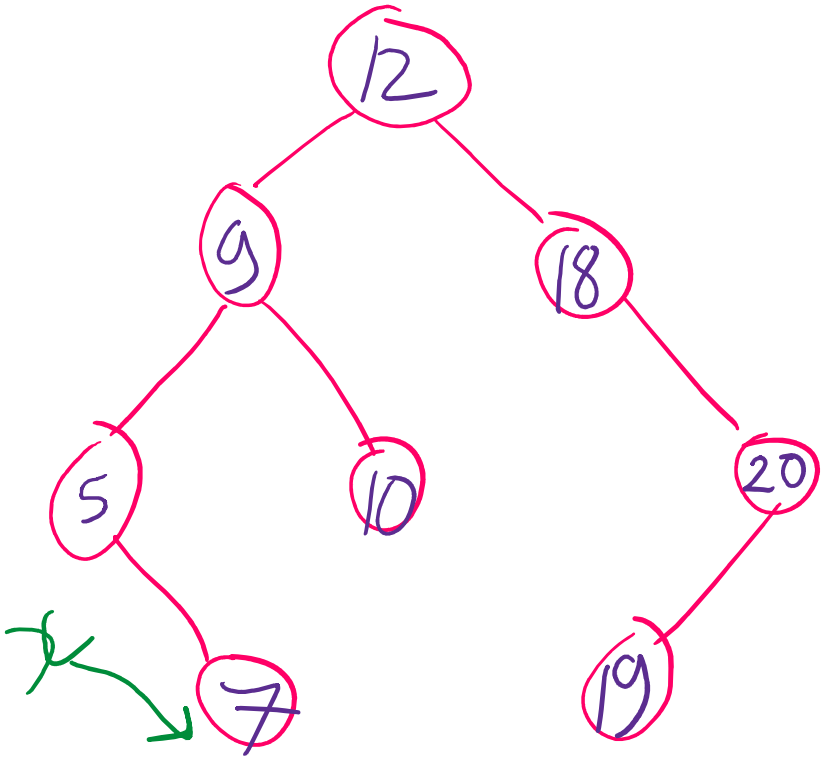
Stack (explicit)

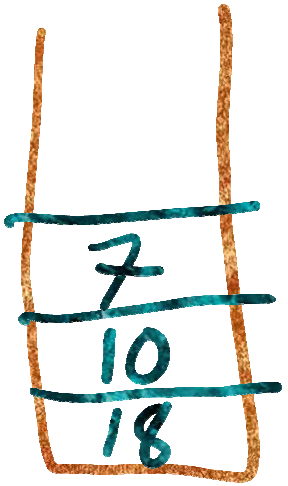
while (stack not empty)

pop
visit
push right
push left

Visit order: 12, 9, 5

Iterative Preorder traversal





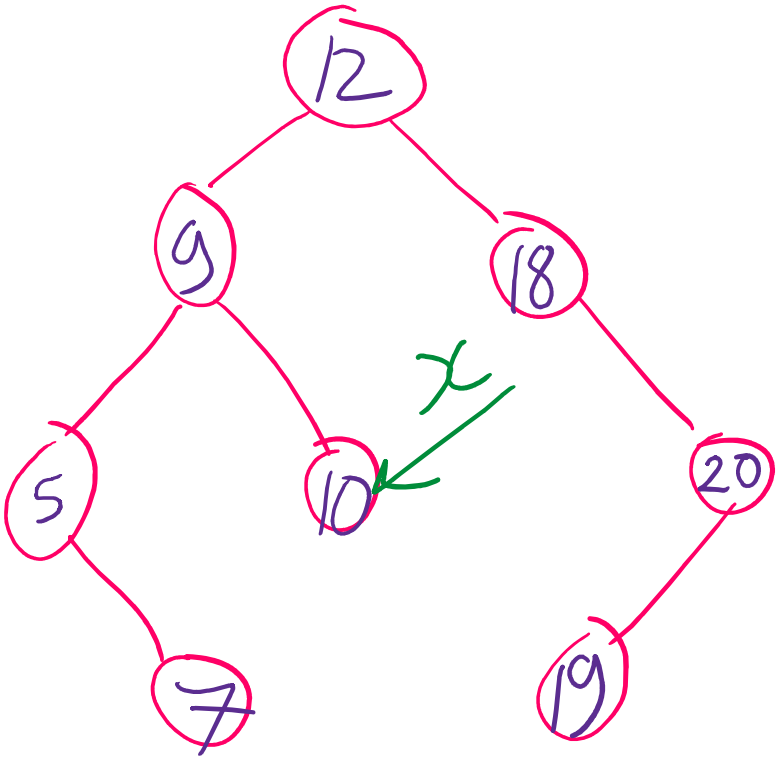
Stack (explicit)

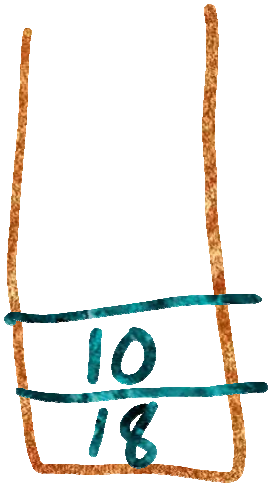
while (stack not empty)

pop
visit
push right
push left

Visit order: 12, 9, 5, 7

Iterative Preorder traversal





Stack (explicit)

while (stack not empty)

pop

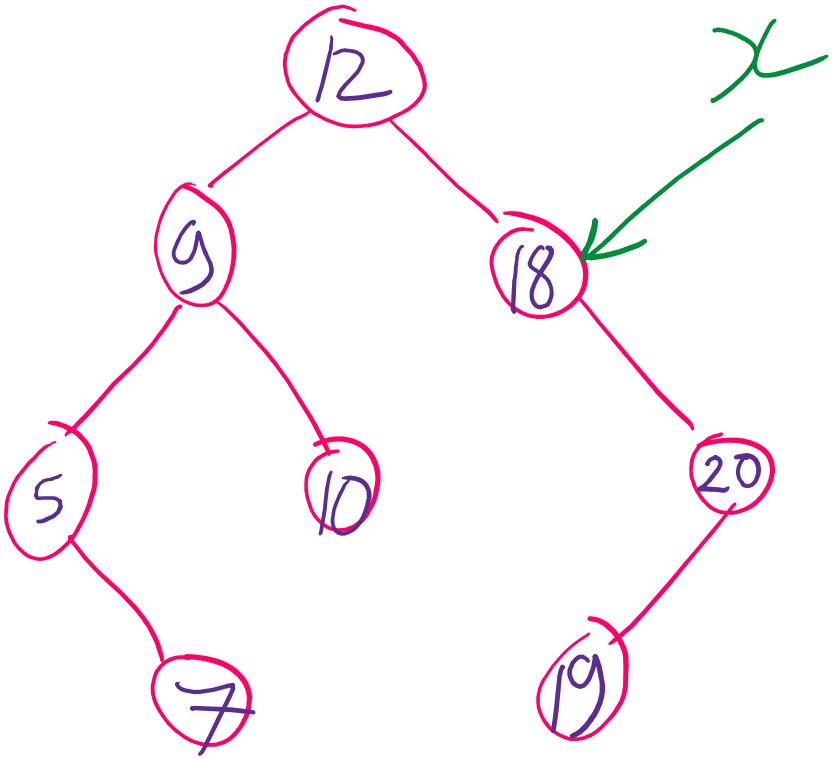
visit

Push right

Push left

Visit order: 12, 9, 5, 7, 10

Iterative Preorder traversal



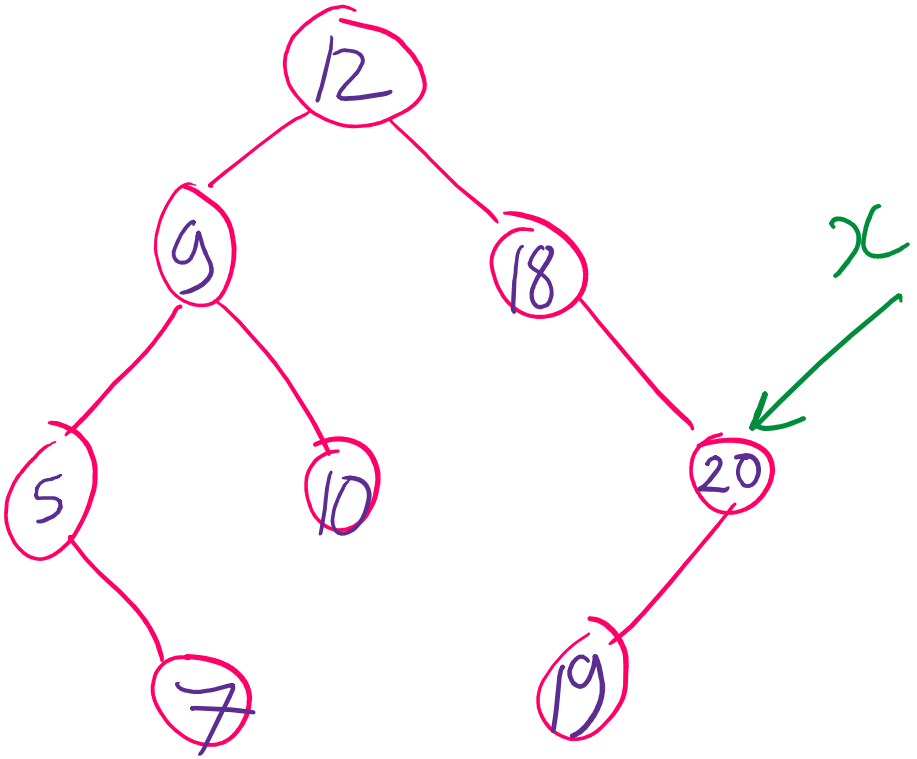
Stack (explicit)

while (stack not empty)

pop
visit
push right
push left

Visit order: 12, 9, 5, 7, 10, 18

Iterative Preorder traversal

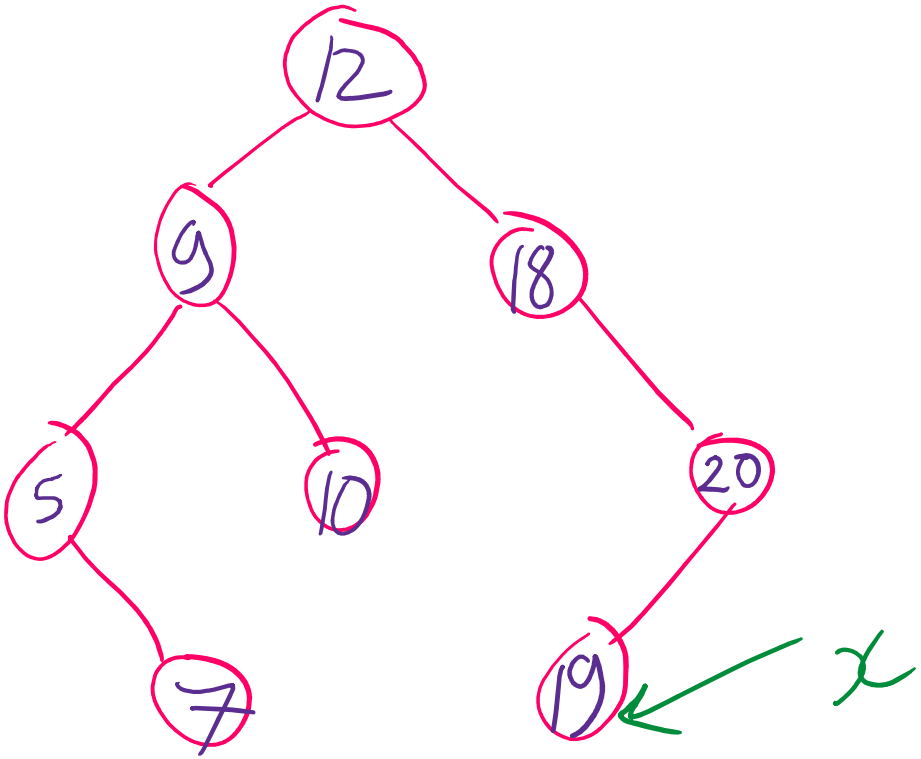


Stack (explicit)

while (stack not empty)
- pop
- visit
- push right
- push left

Visit order: 12, 9, 5, 7, 10, 18, 20

Iterative Preorder traversal

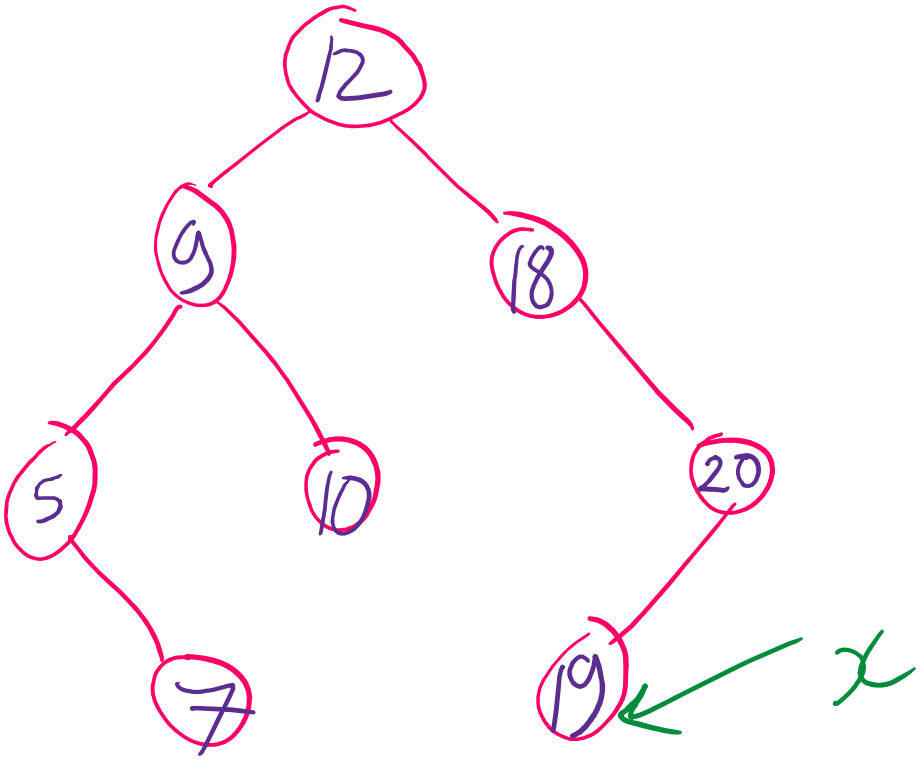


Stack (explicit)

while (stack not empty)
- pop
- visit
- push right
- push left

Visit order: 12, 9, 5, 7, 10, 18, 20, 19

Iterative Preorder traversal



while (stack not empty)

pop

visit

push right

push left

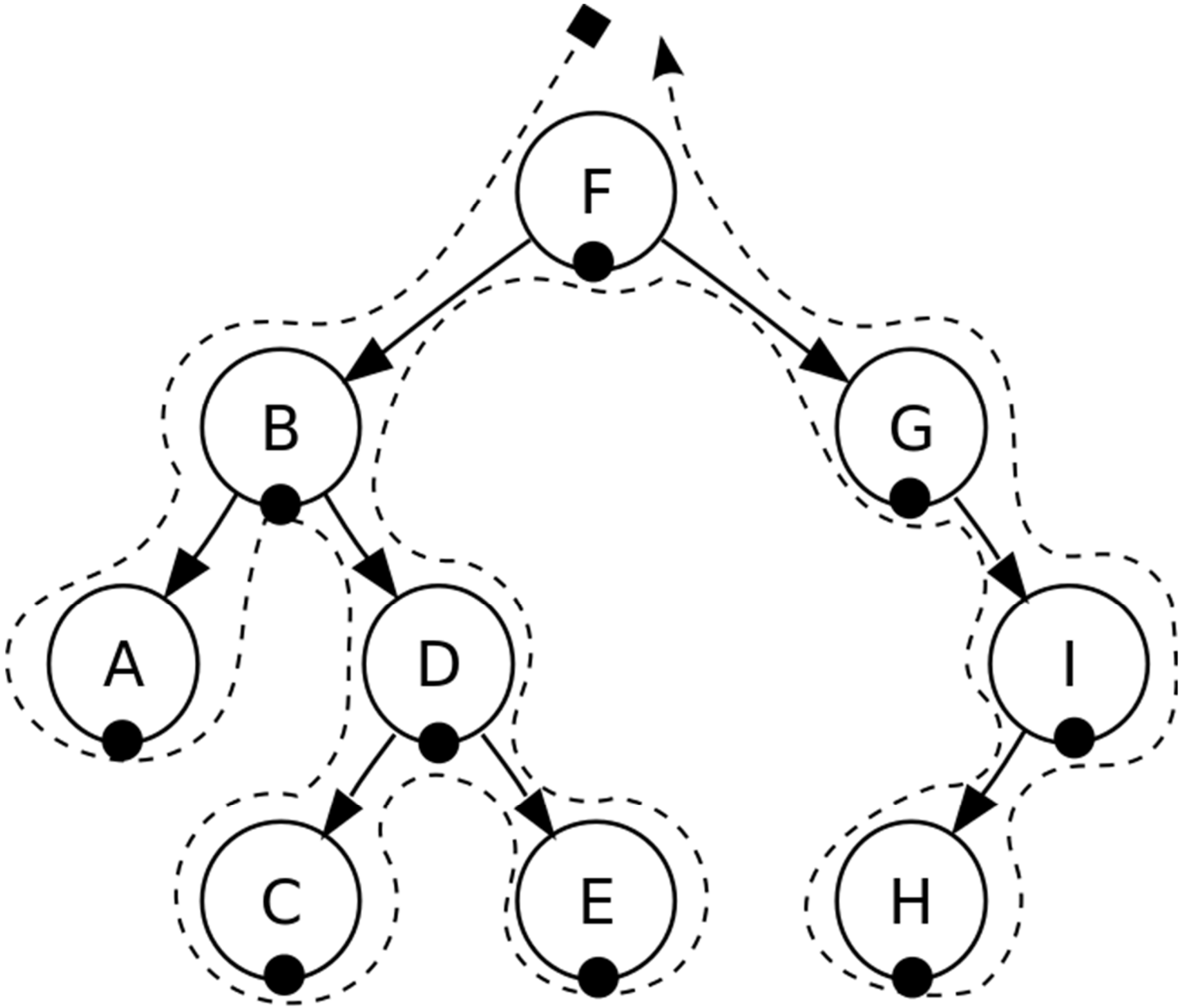
Visit order: 12, 9, 5, 7, 10, 18, 20, 19

Traversals of a Binary Tree

- Preorder traversal
  - Visit root before we visit root's subtrees
- In-order traversal
  - Visit root of a binary tree between visiting nodes in root's subtrees.
  - left then root then right

In-order traversal

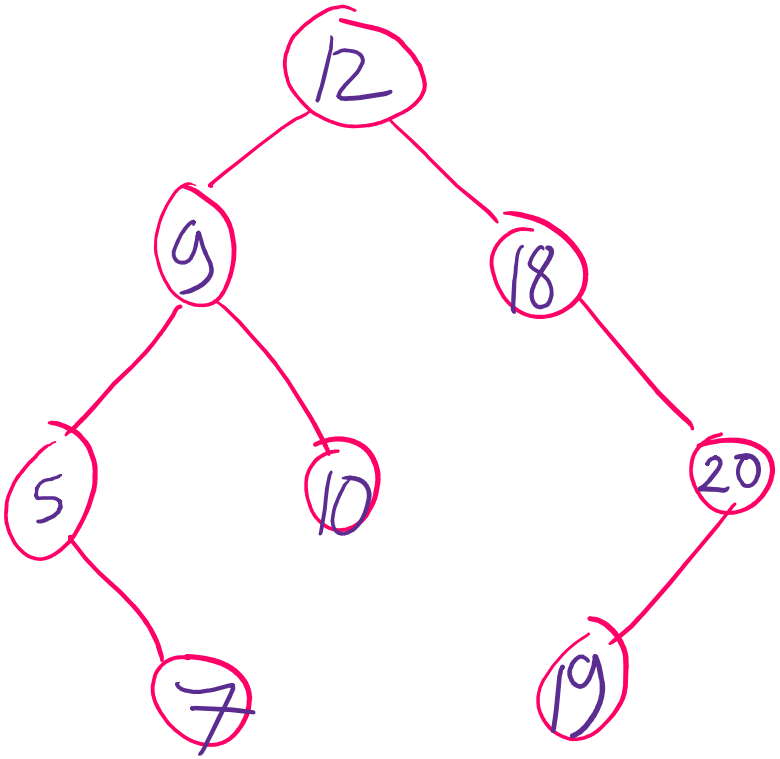
A
B
C
D
E
F
G
H
I



In-order traversal implementation

```
void traverse(BinaryNode<T> root){

   if(root != null){

      traverse(root.left);

      System.out.println(root.data);

      traverse(root.right);

    }

}
```

Iterative Inorder traversal



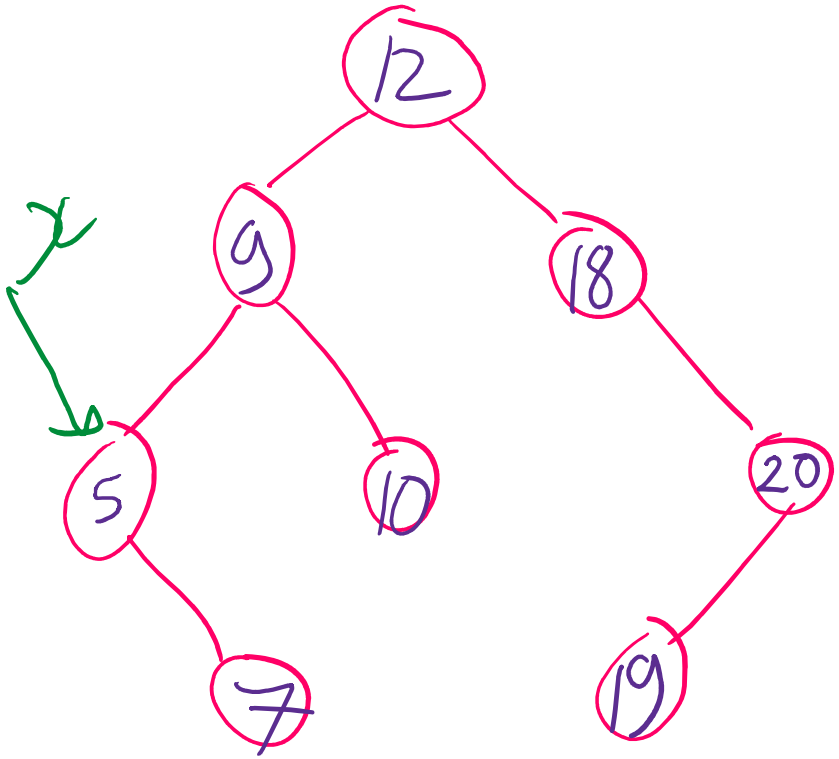
repeat until stack empty & x == null

Stack (explicit)

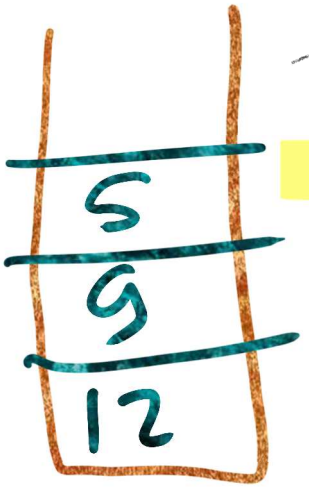
x = leftmost node
push to stack while moving down
pop & visit
x = x.right

Visit order:

Iterative Inorder traversal



repeat until stack empty & x == null



x = leftmost node

push to stack while moving down

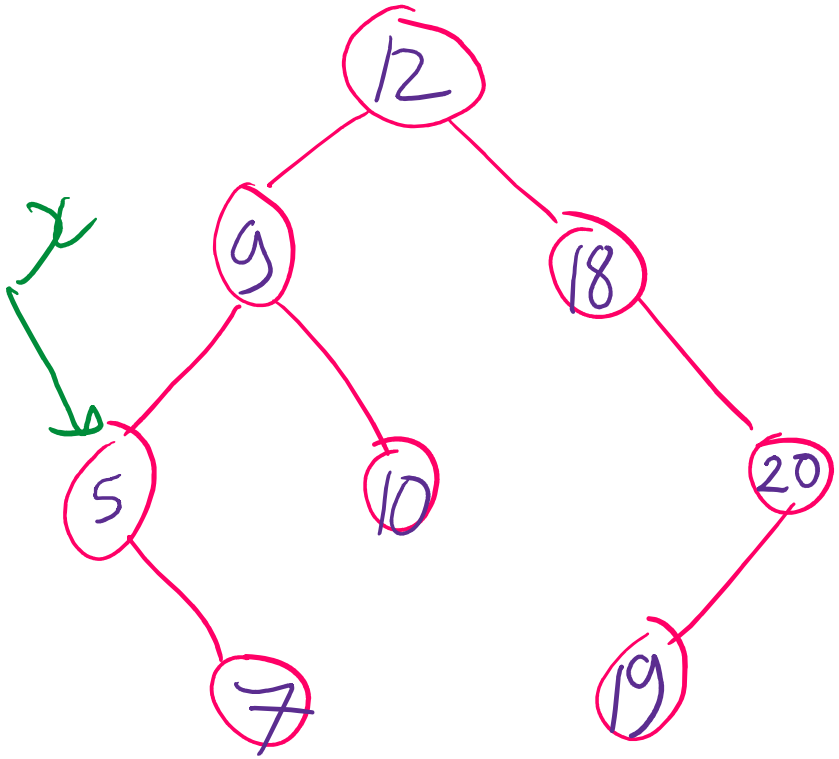
pop & visit

x = x.right

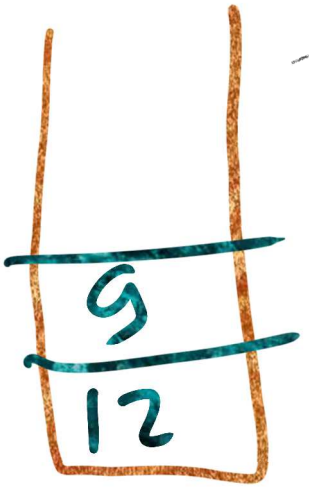
Stack (explicit)

Visit order:

Iterative Inorder traversal



repeat until stack empty & x == null



x = leftmost node

push to stack while moving down

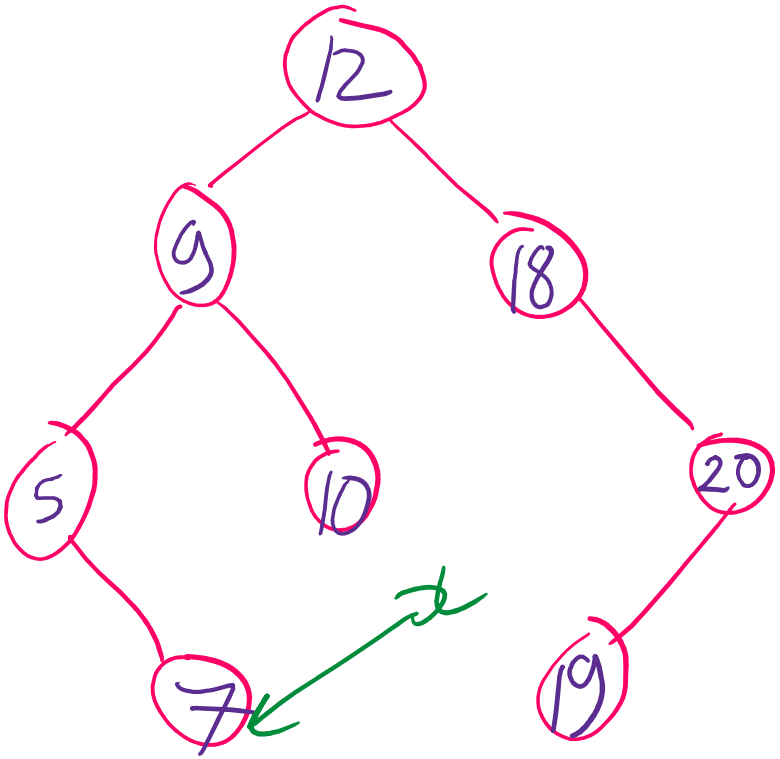
pop & visit

x = x.right

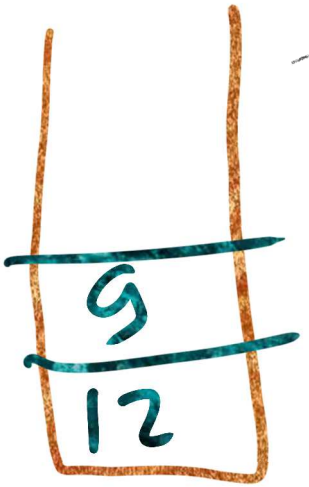
Stack (explicit)

Visit order: 5

Iterative Inorder traversal



repeat until stack empty & x == null



x = leftmost node

push to stack while moving down

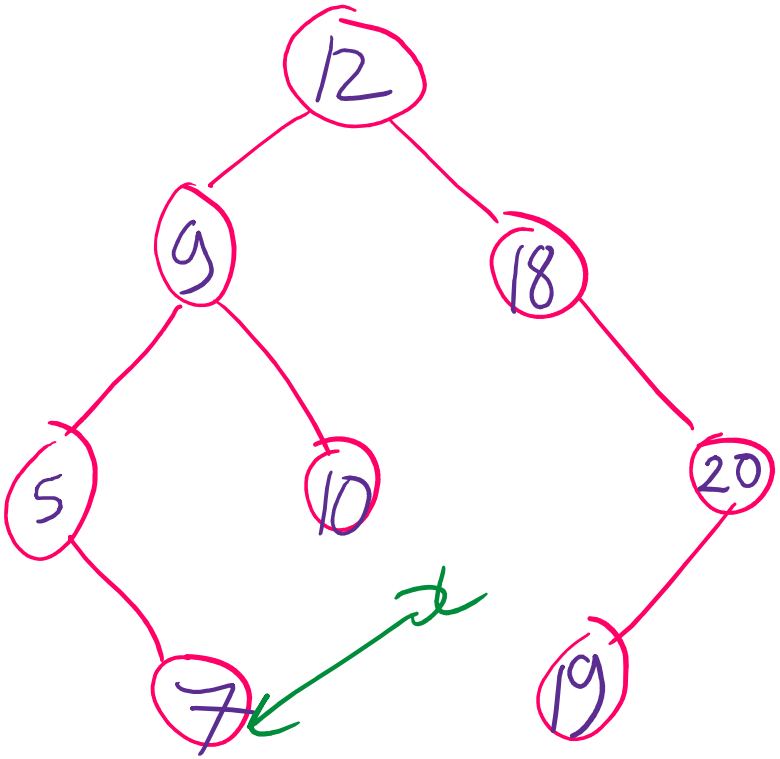
pop & visit

x = x.right

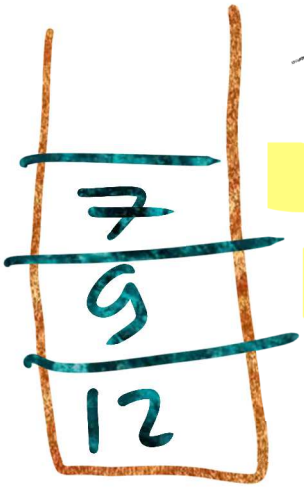
Stack (explicit)

Visit order: 5

Iterative Inorder traversal



repeat until stack empty & x == null



x = leftmost node

push to stack while moving down

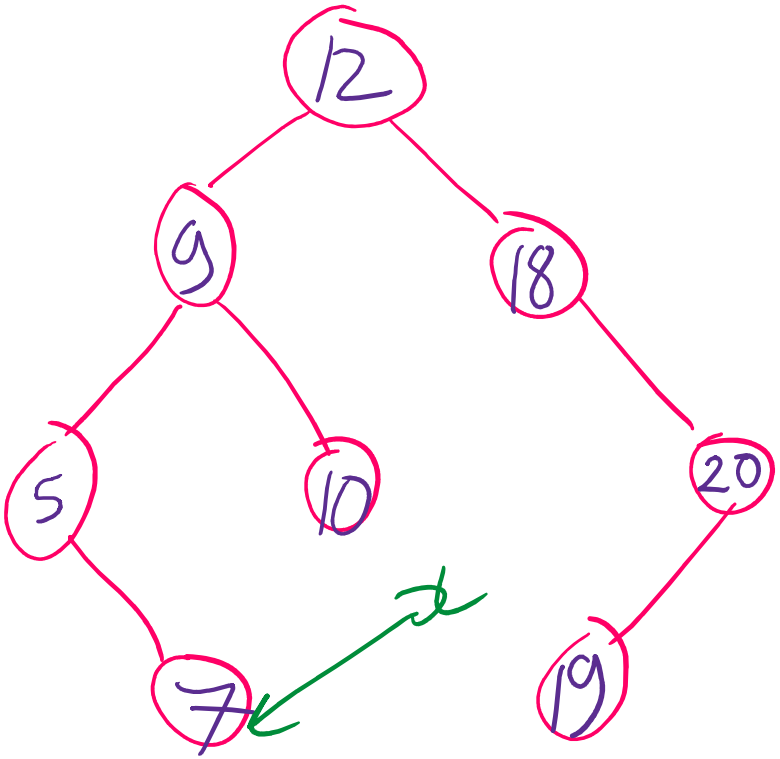
pop & visit

x = x.right

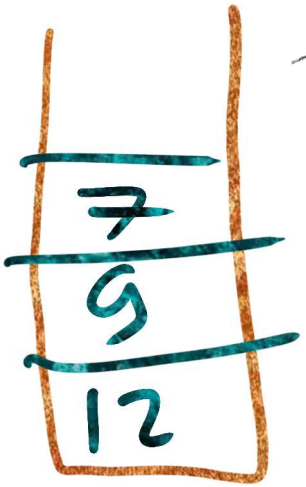
Stack (explicit)

Visit order: 5

Iterative Inorder traversal



repeat until stack empty & x == null



x = leftmost node

push to stack while moving down

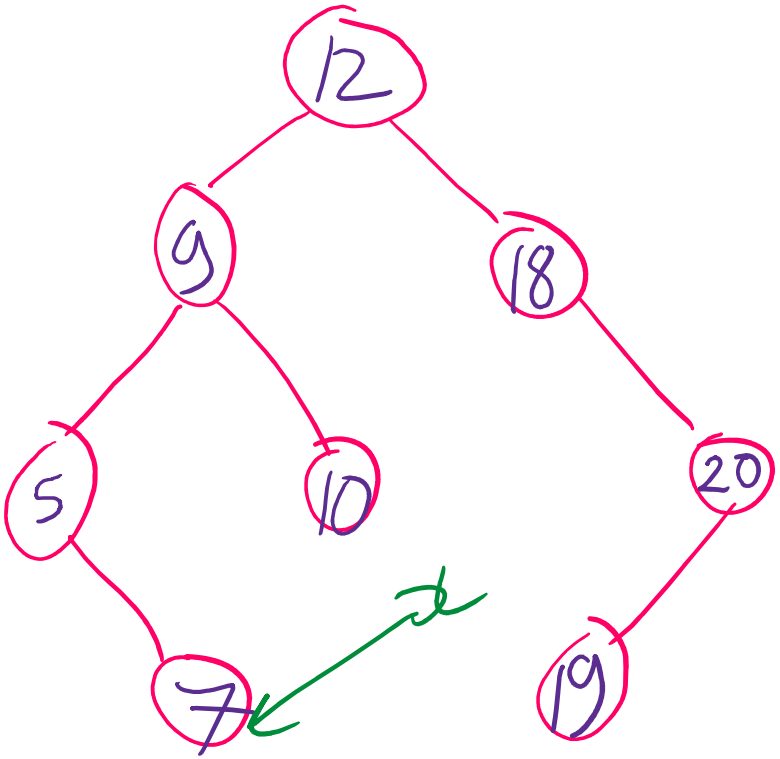
Pop & visit

x = x.right

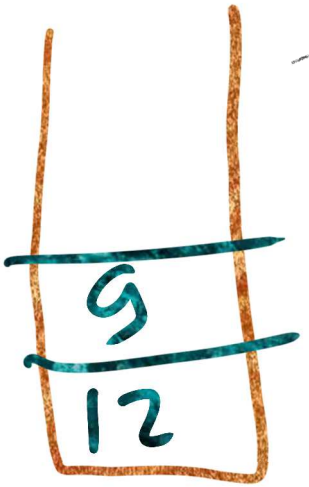
Stack (explicit)

Visit order: 5, 7

Iterative Inorder traversal



repeat until stack empty & x == null



Stack (explicit)

x = leftmost node

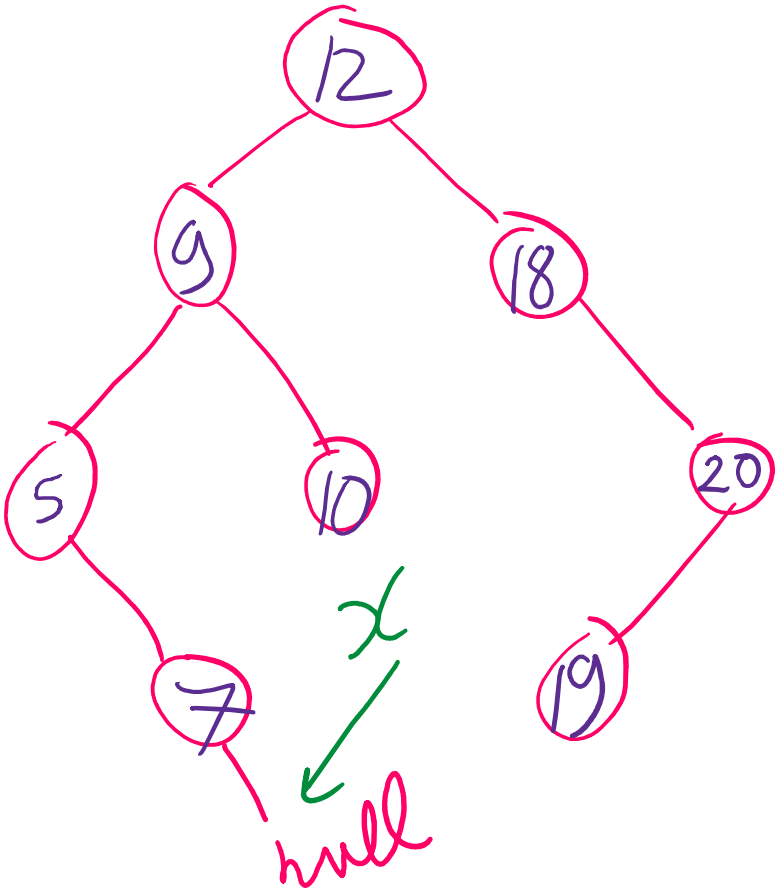
push to stack while moving down

Pop & visit

x = x.right

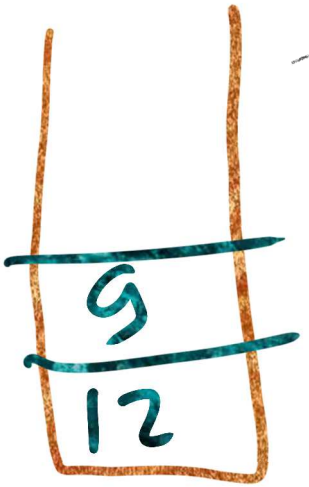
Visit order: 5, 7

Iterative Inorder traversal



Visit order: 5, 7

repeat until stack empty & x == null



Stack (explicit)

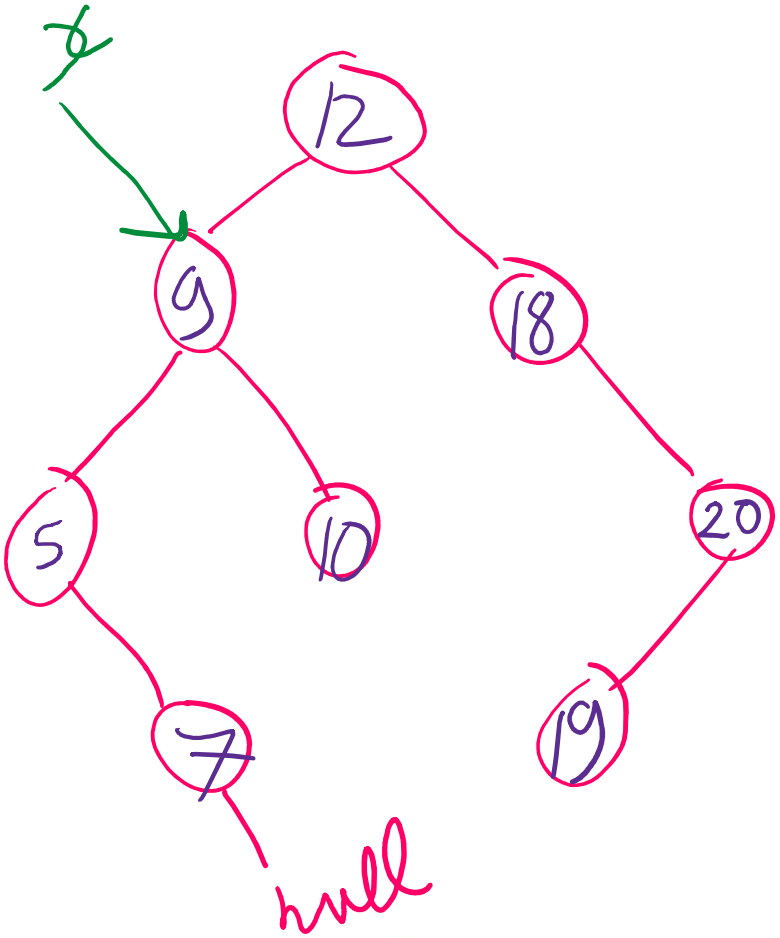
x = leftmost node

push to stack while moving down

pop & visit

x = x.right

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

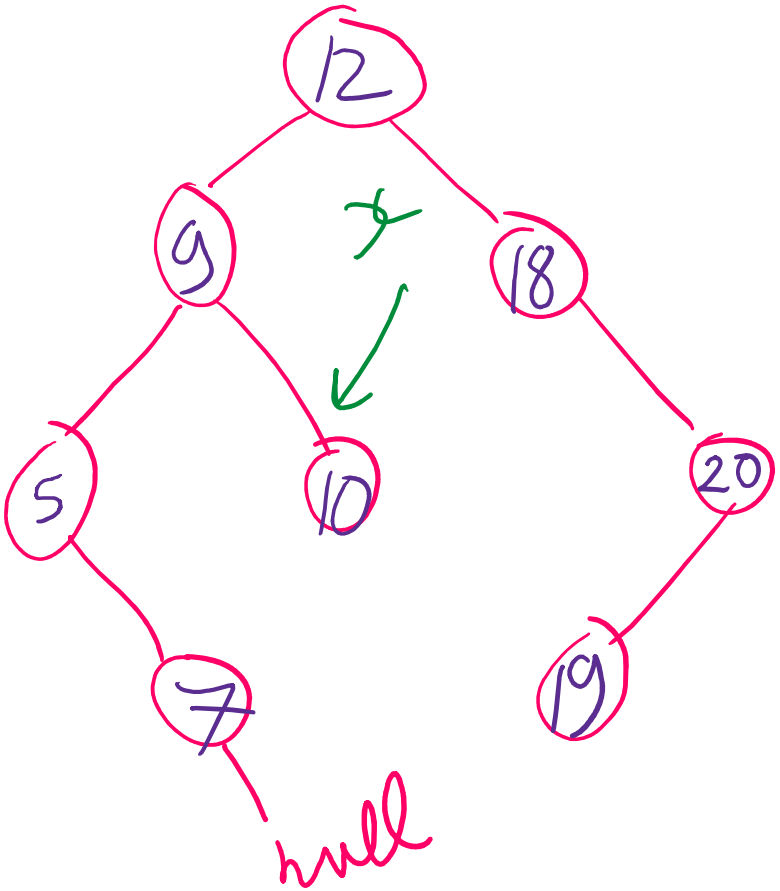
pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9

Iterative Inorder traversal



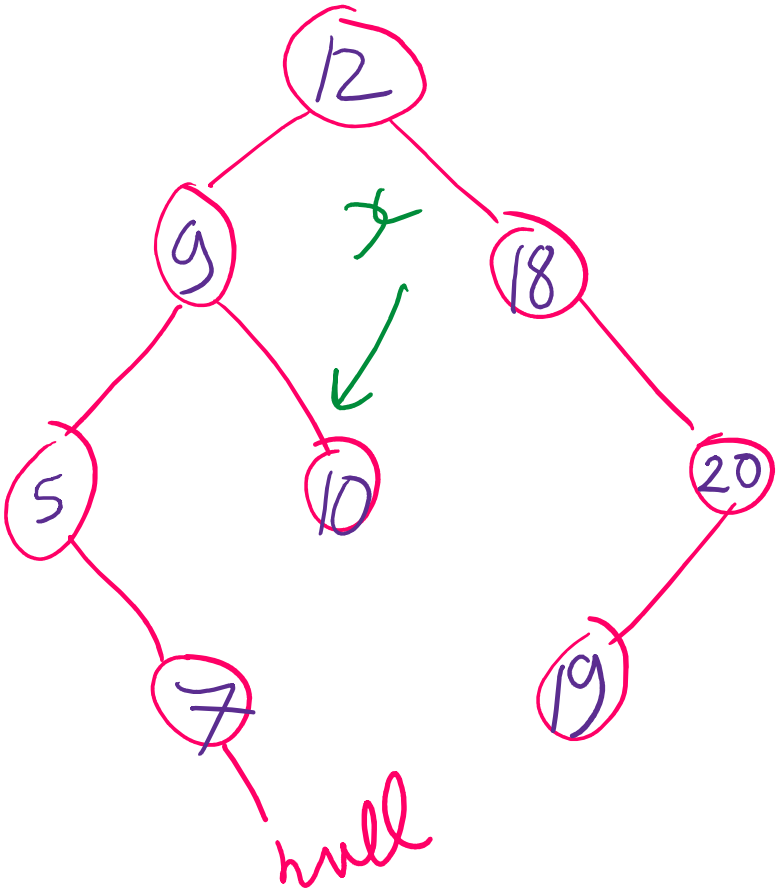
repeat until stack empty & x == null

x = leftmost node
push to stack while moving down
pop & visit
x = x.right

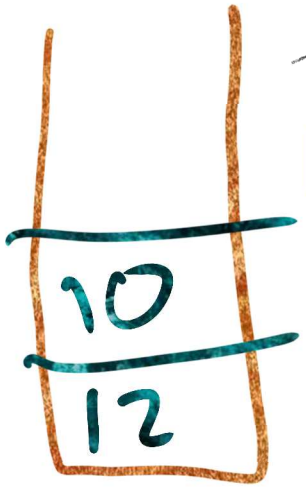
Stack (explicit)

Visit order: 5, 7, 9

Iterative Inorder traversal



repeat until stack empty & x == null



x = leftmost node

Push to stack while moving down

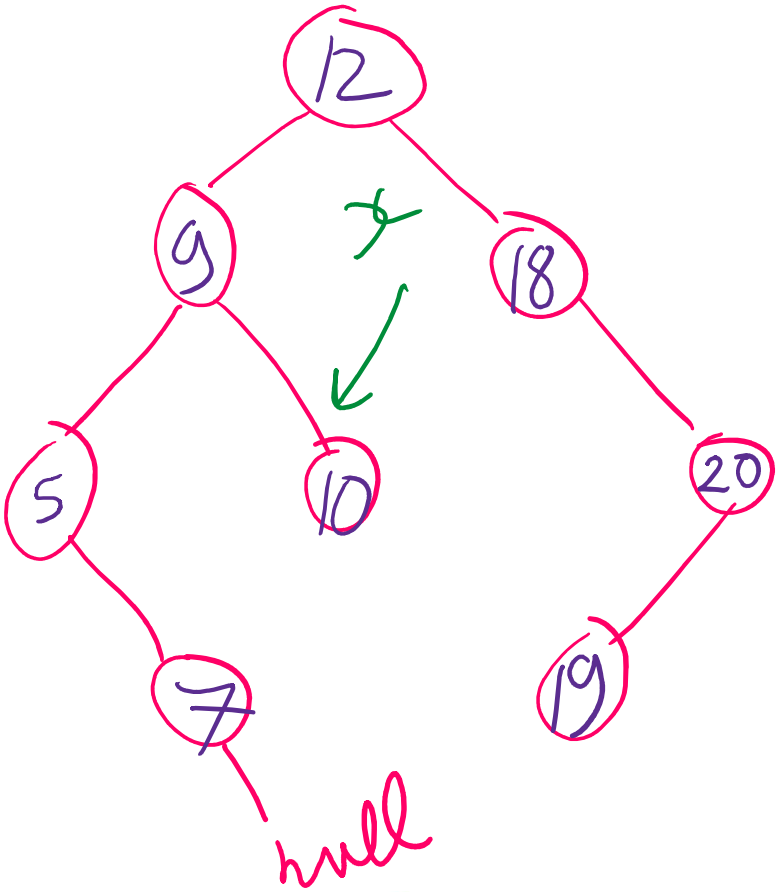
Pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

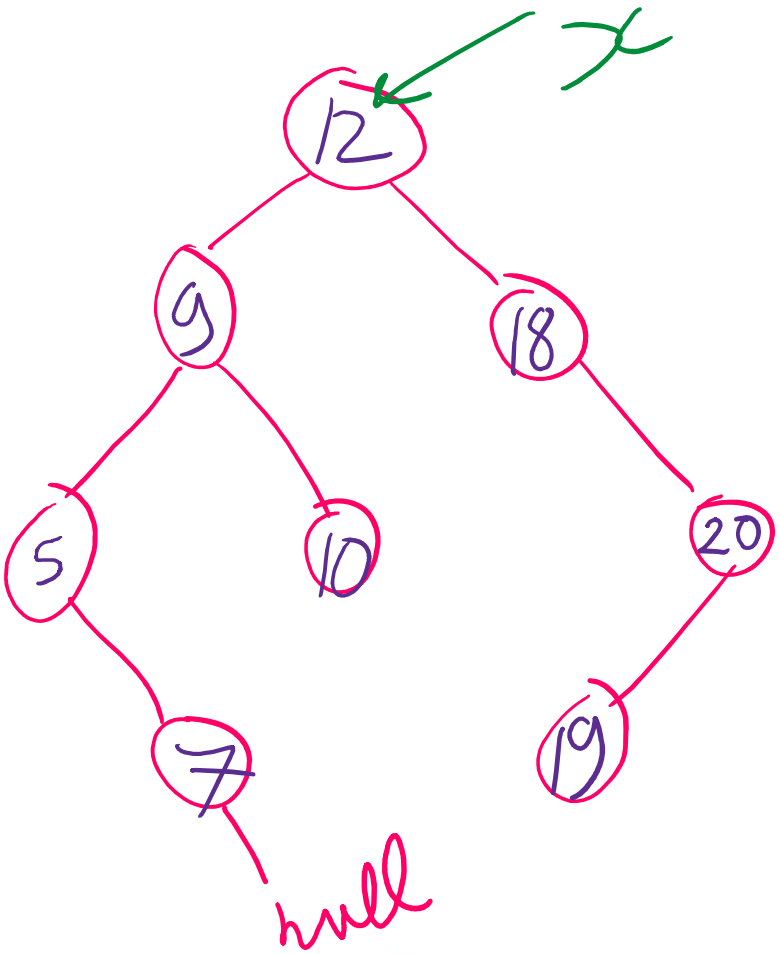
pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

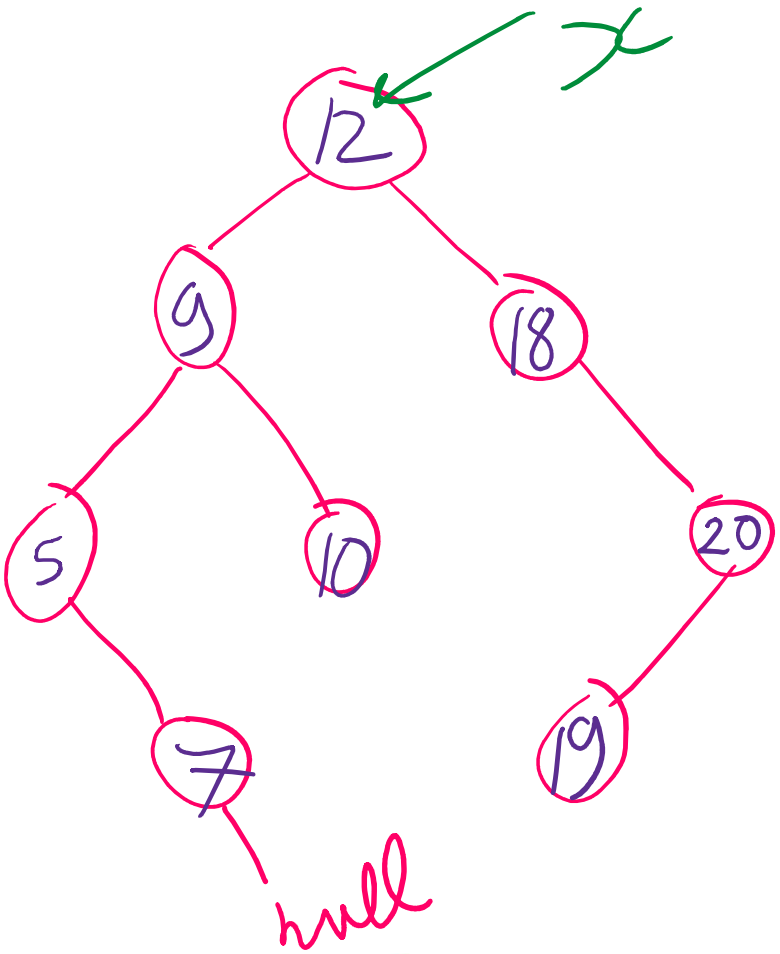
pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

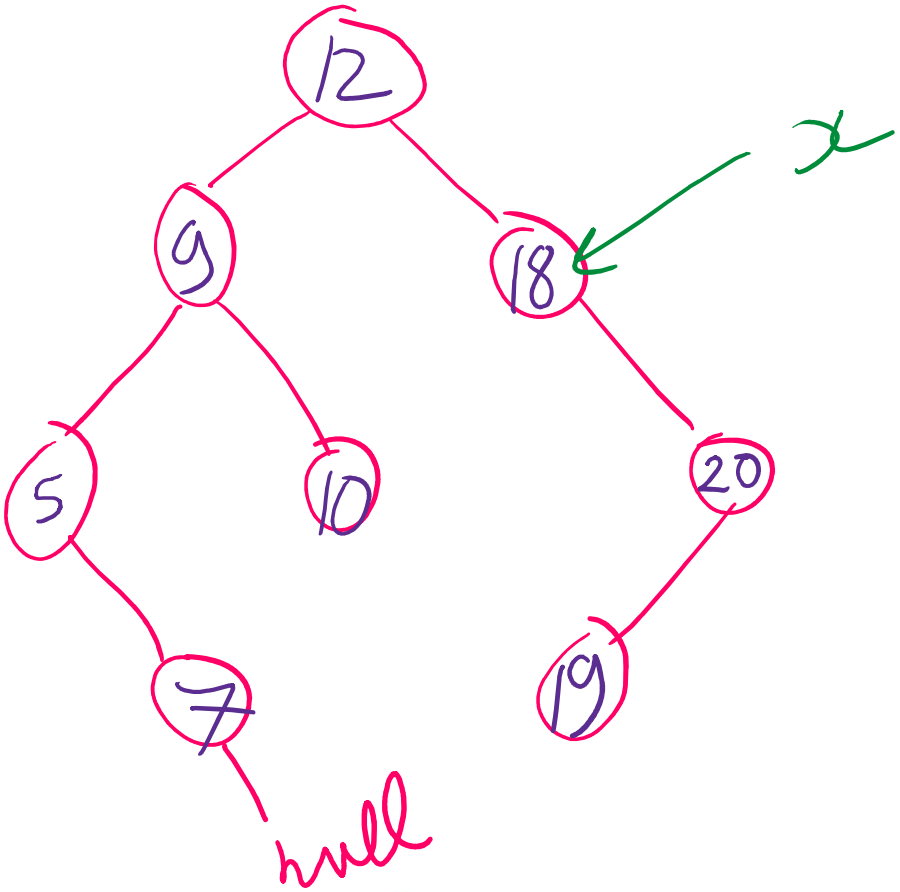
pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10, 12

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

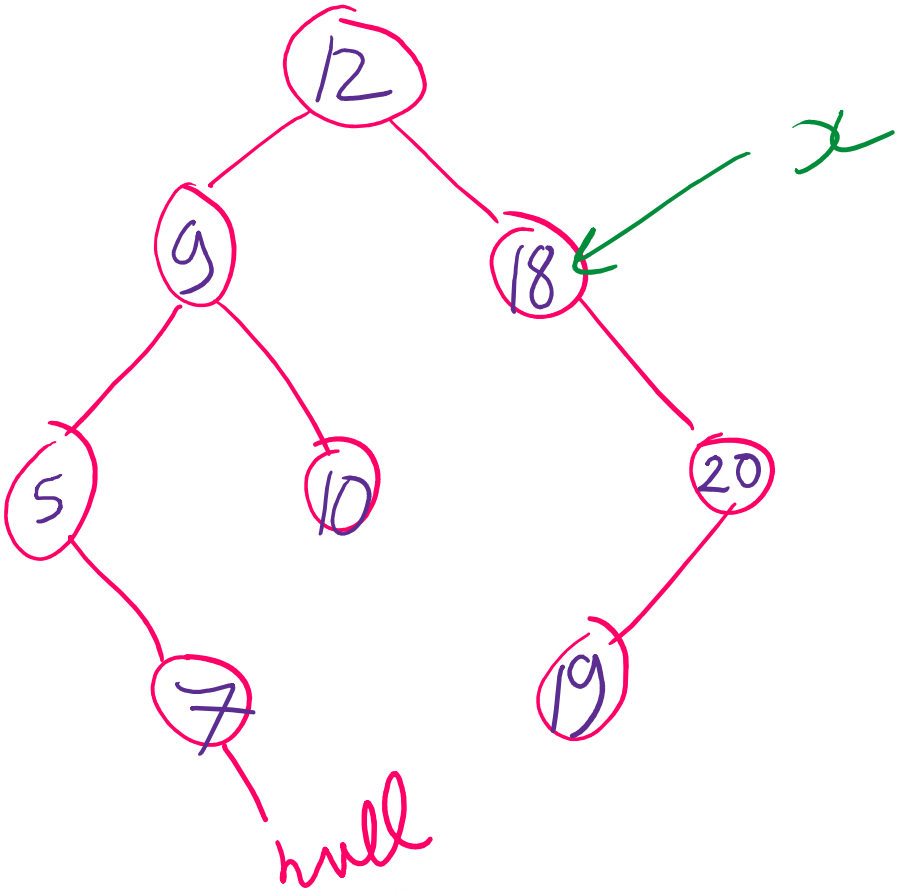
pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10, 12

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

Push to stack while moving down

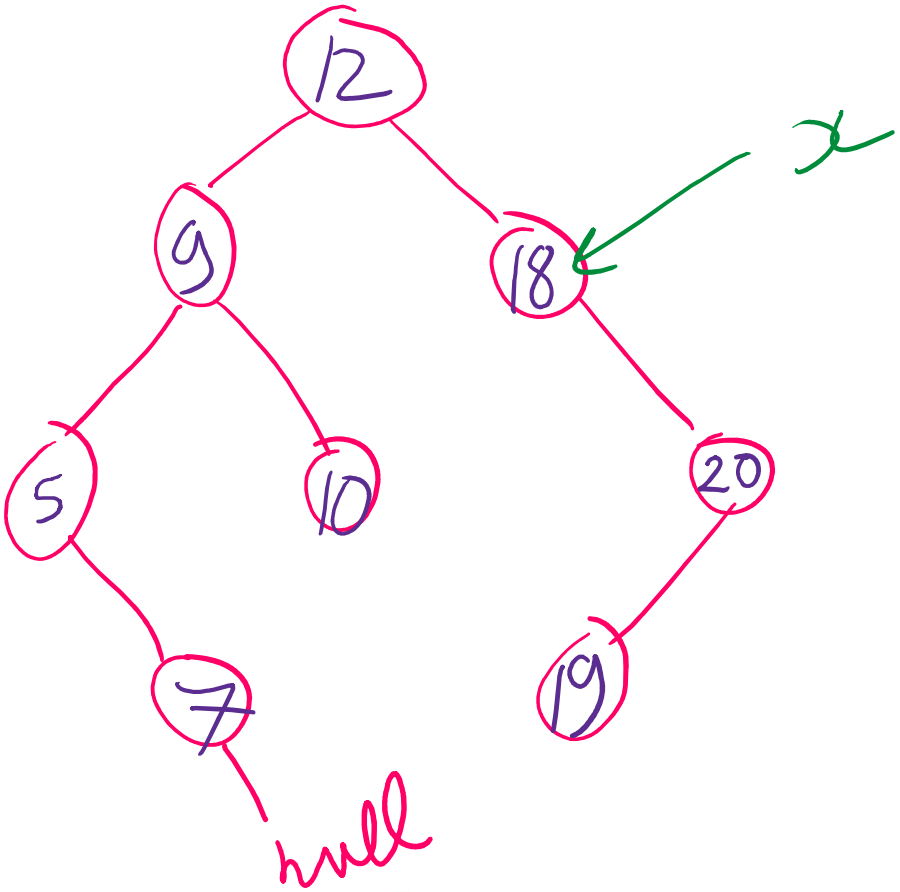
Pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10, 12

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

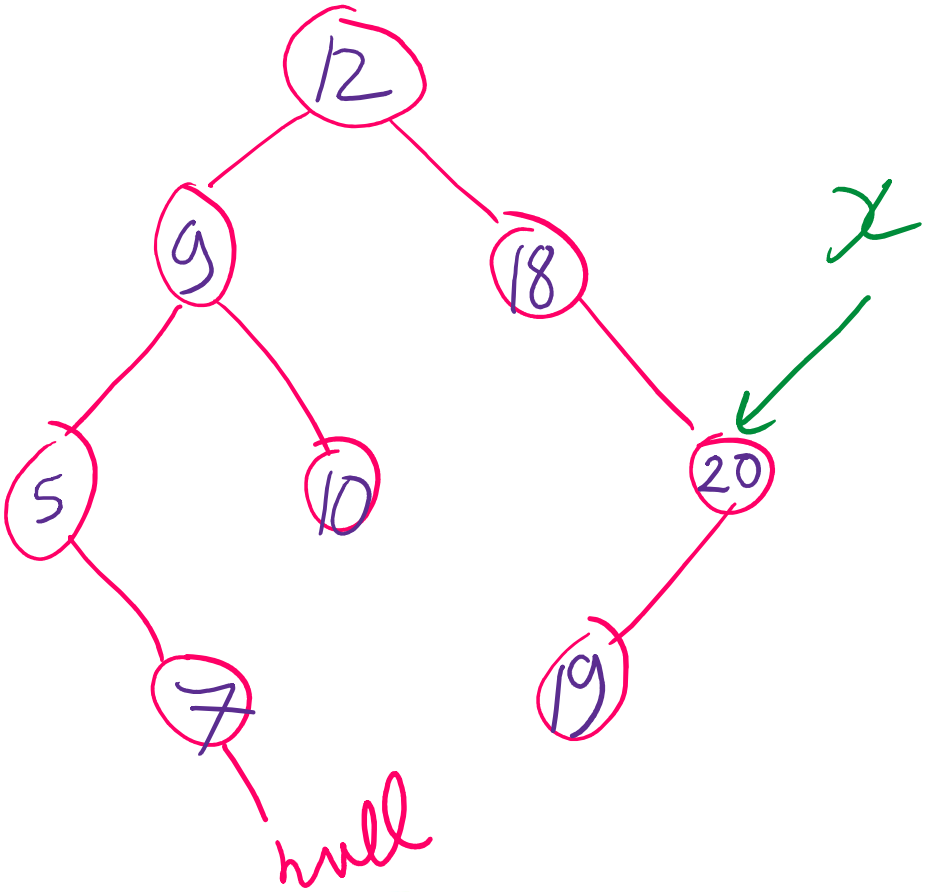
pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10, 12, 18

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

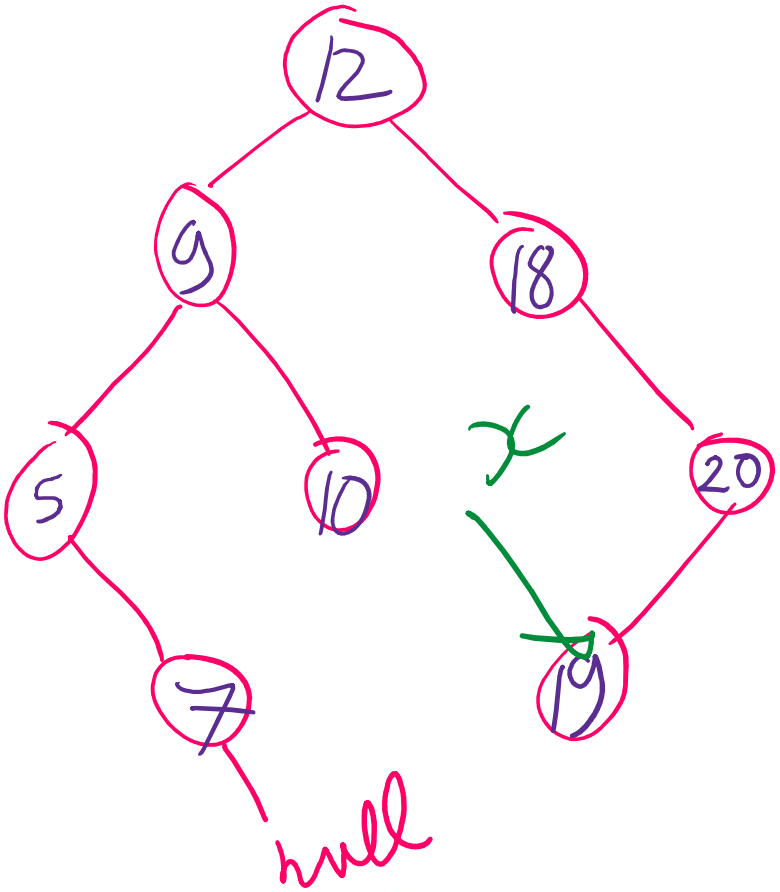
pop & visit

x = x.right

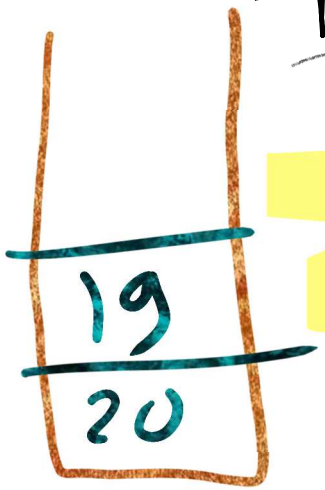
Stack (explicit)

Visit order: 5, 7, 9, 10, 12, 18

Iterative Inorder traversal



repeat until stack empty & x == null



x = leftmost node

Push to stack while moving down

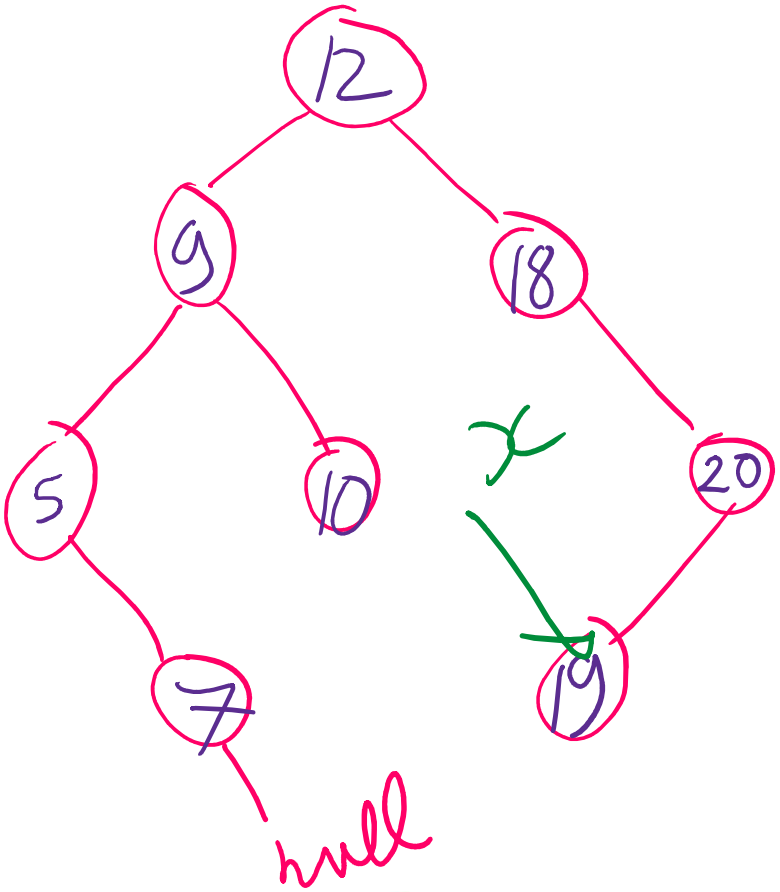
Pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10, 12, 18

Iterative Inorder traversal



repeat until stack empty & x == null

x = leftmost node

push to stack while moving down

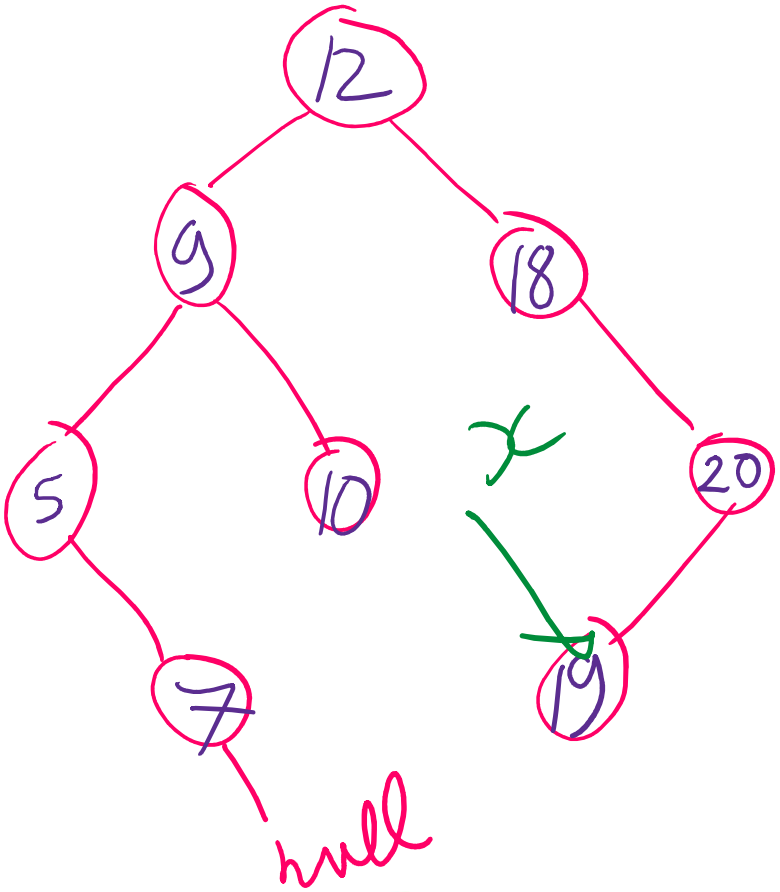
Pop & visit

x = x.right

Stack (explicit)

Visit order: 5, 7, 9, 10, 12, 18, 19

Iterative Inorder traversal



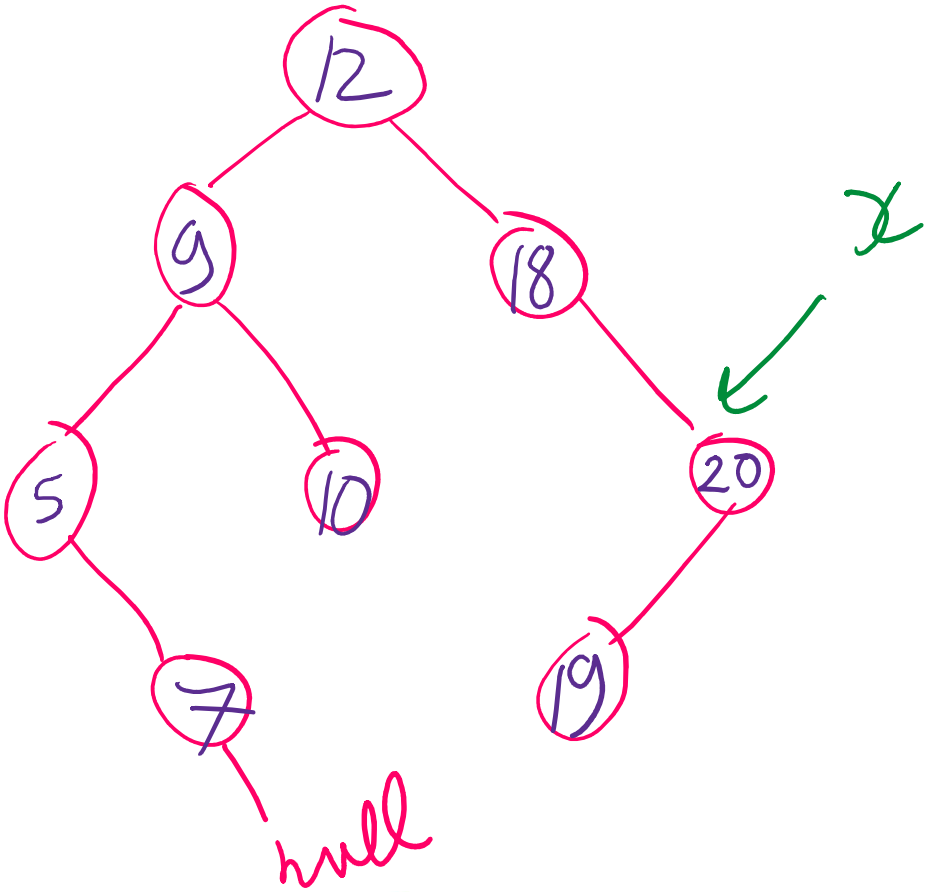
repeat until stack empty & x == null

Stack (explicit)

x = leftmost node
push to stack while moving down
Pop & visit
x = x.right

Visit order: 5, 7, 9, 10, 12, 18, 19

Iterative Inorder traversal



repeat until stack empty & x == null

Stack (explicit)

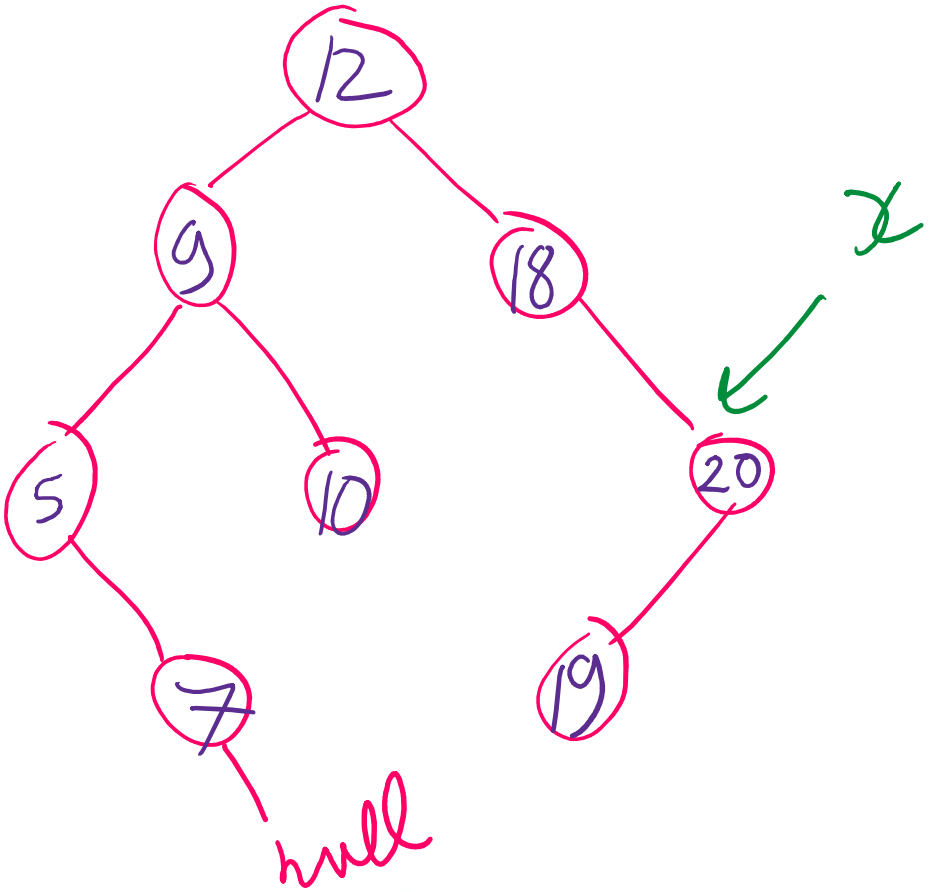
x = leftmost node
push to stack while moving down

Pop & visit

x = x.right

Visit order: 5, 7, 9, 10, 12, 18, 19, 20

Iterative Inorder traversal



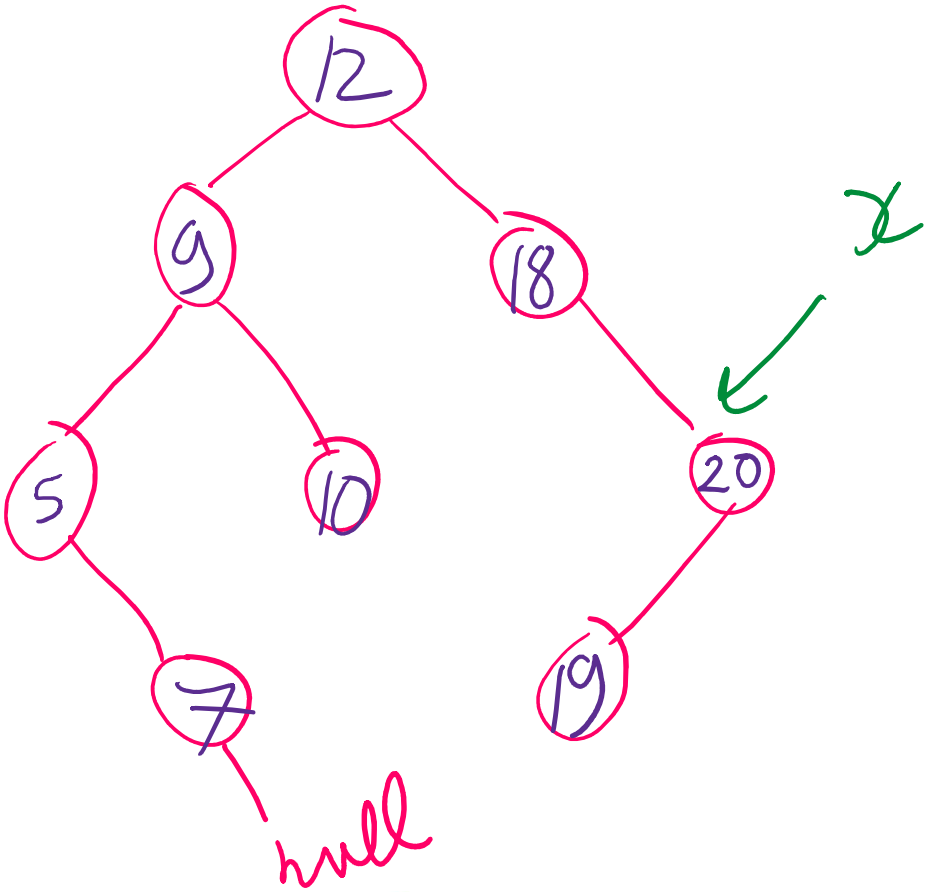
repeat until stack empty & x == null

Stack (explicit)

x = leftmost node
push to stack while moving down
pop & visit
x = x.right

Visit order: 5, 7, 9, 10, 12, 18, 19, 20

Iterative Inorder traversal



repeat until stack empty & x == null

Stack (explicit)

x = leftmost node
push to stack while moving down
pop & visit
x = x.right

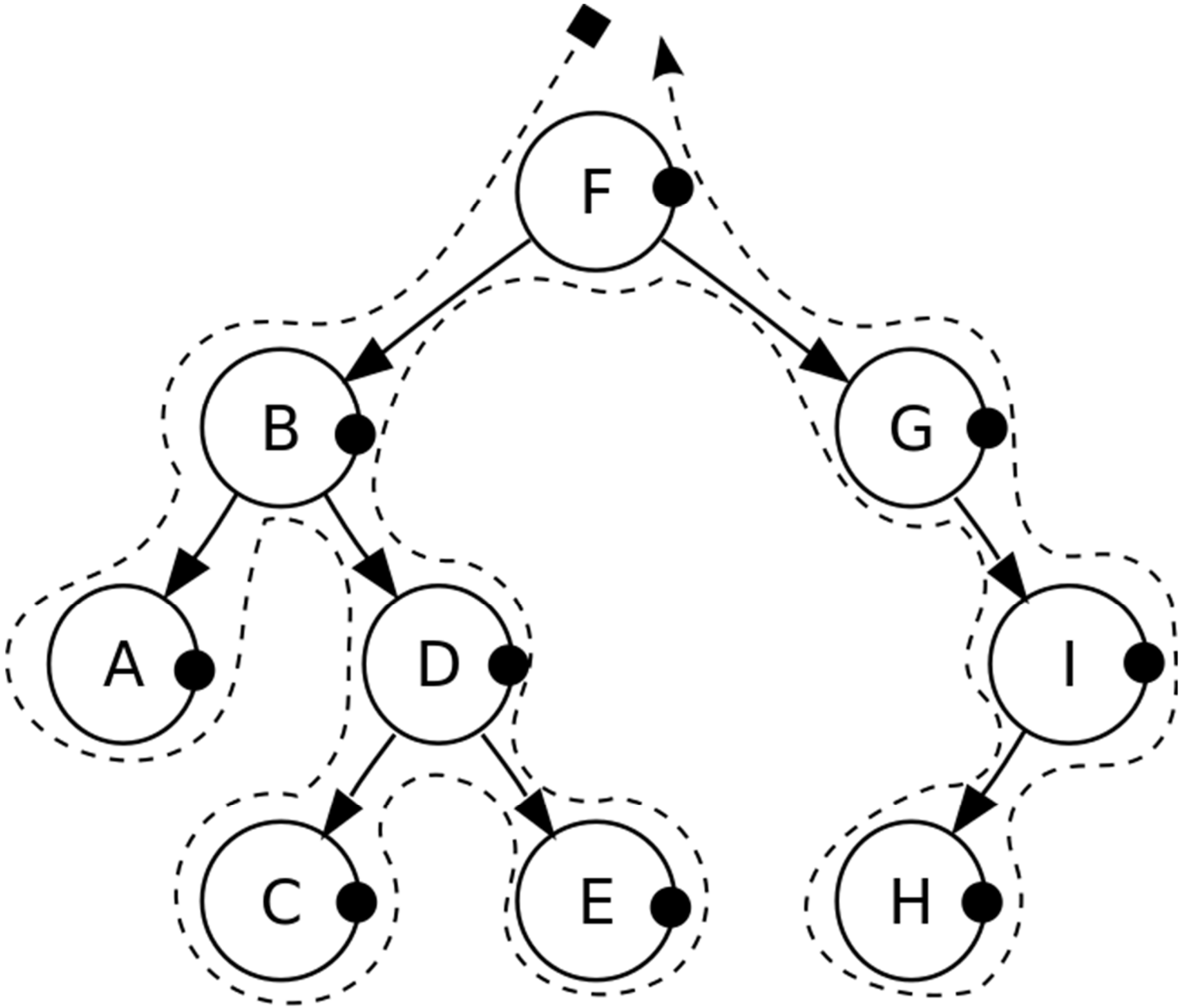
Visit order: 5, 7, 9, 10, 12, 18, 19, 20

Traversals of a Binary Tree

- Preorder traversal
  - Visit root before we visit root's subtrees
- Inorder traversal
  - Visit root of a binary tree between visiting nodes in root's subtrees.
- Postorder traversal
  - Visit root of a binary tree after visiting nodes in root's subtrees
  - left then right then root

Post-order traversal

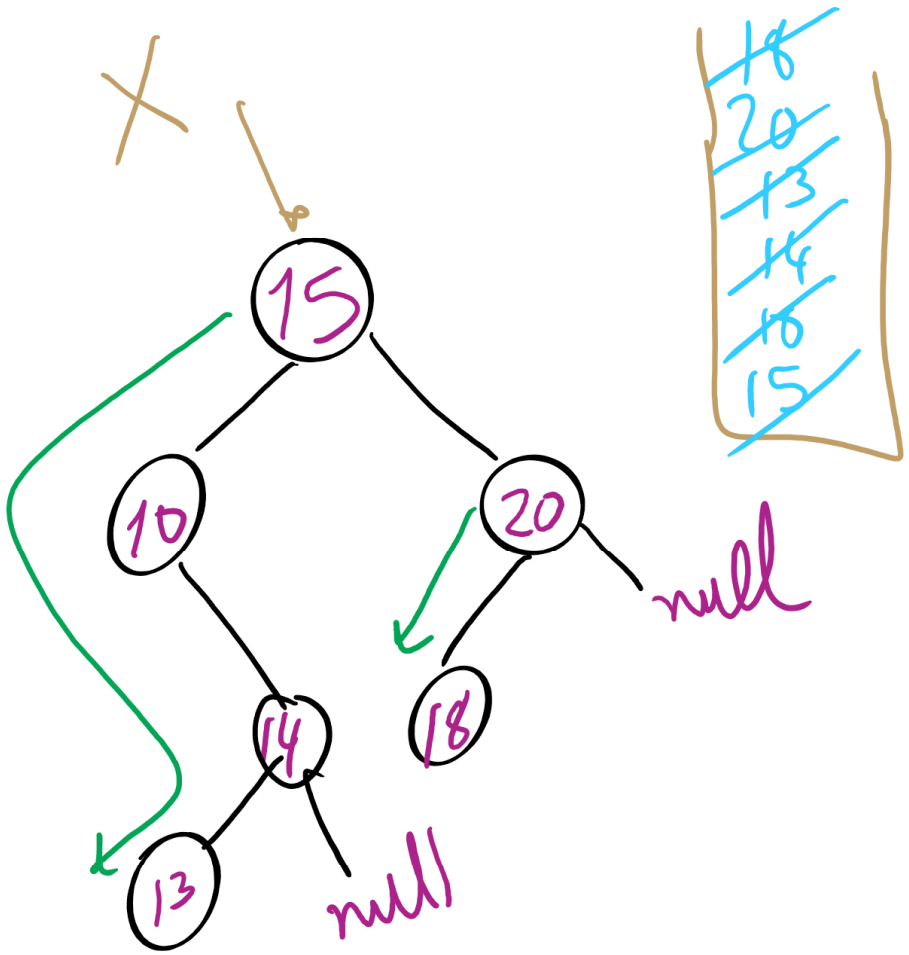
A
C
E
D
B
H
I
G
F



Post-order traversal implementation

```
void traverse(BinaryNode<T> root){

   if(root != null){

      traverse(root.left);

      traverse(root.right);

      System.out.println(root.data);

    }

}
```

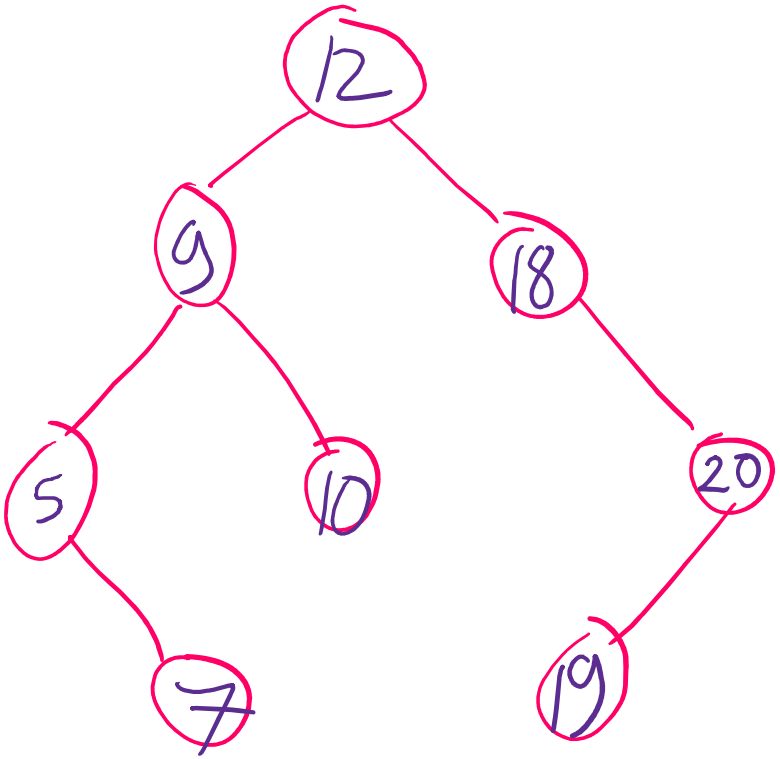
Iterative Post-order Traversal

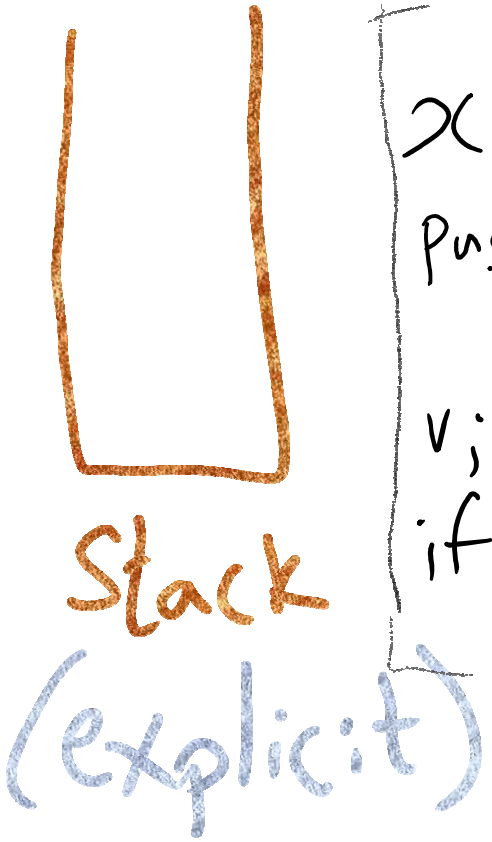


13, 14, 10, 18, 2015

```
stack = empty stack
x = root
While( x != null ||
       stack is not empty){
  -find leftmost leaf
  -push nodes as we
     go down the tree
  - x = pop() & visit x
  - if x is a left child
       x = right sibling
  else
       x = null
}
```

Iterative Postorder traversal





repeat until stack empty & x == null

x = leftmost leaf

push to stack while moving down

visit x

if x is rightchild

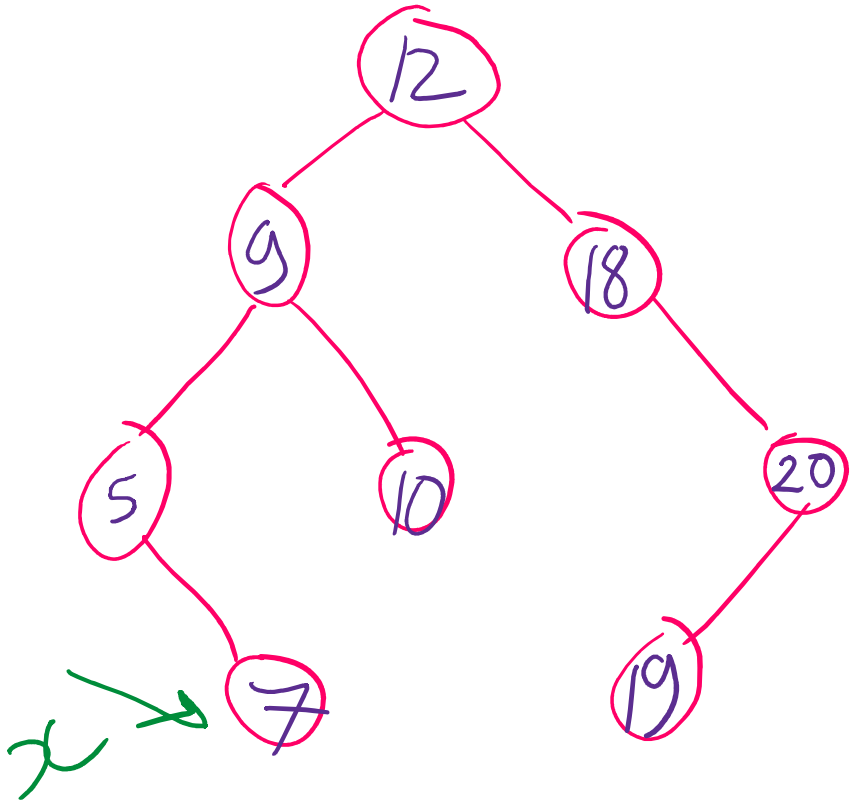
pop & visit

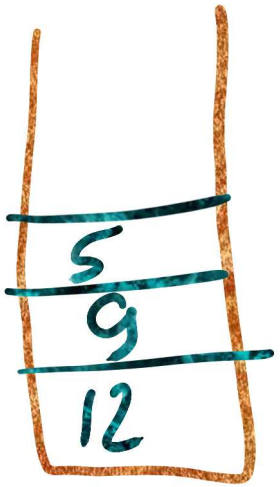
skip leftmost leaf

else x = parent.right

Visit order:

Iterative Postorder traversal





Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down

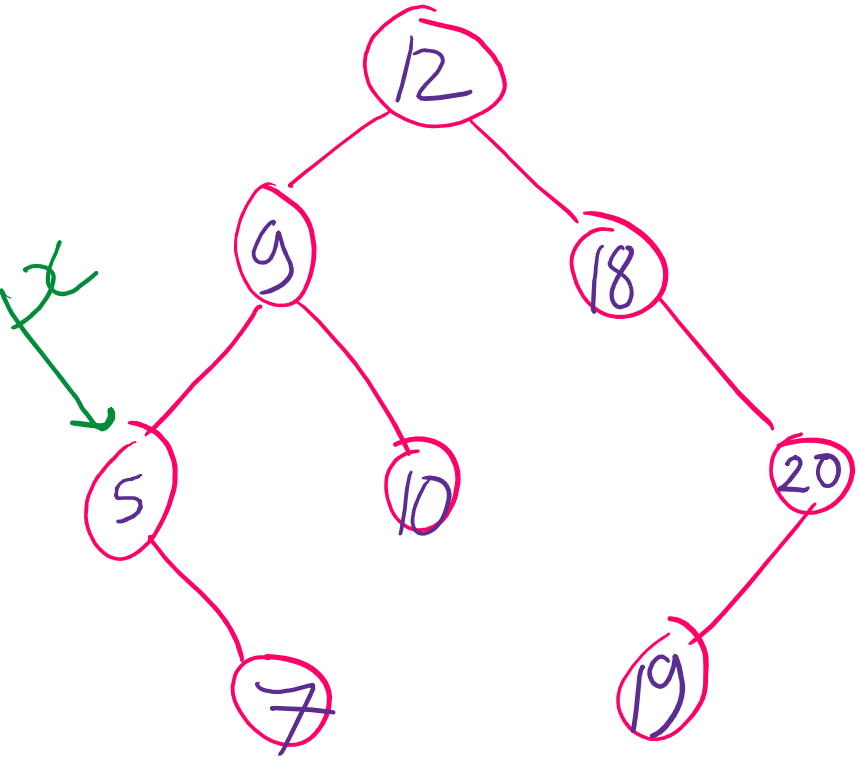
visit x

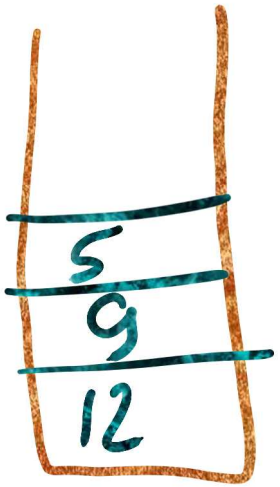
if x is rightchild
pop & visit
skip leftmost leaf

else x = parent.right

Visit order: 7

Iterative Postorder traversal





Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf

push to stack while moving down

visit x

if x is rightchild

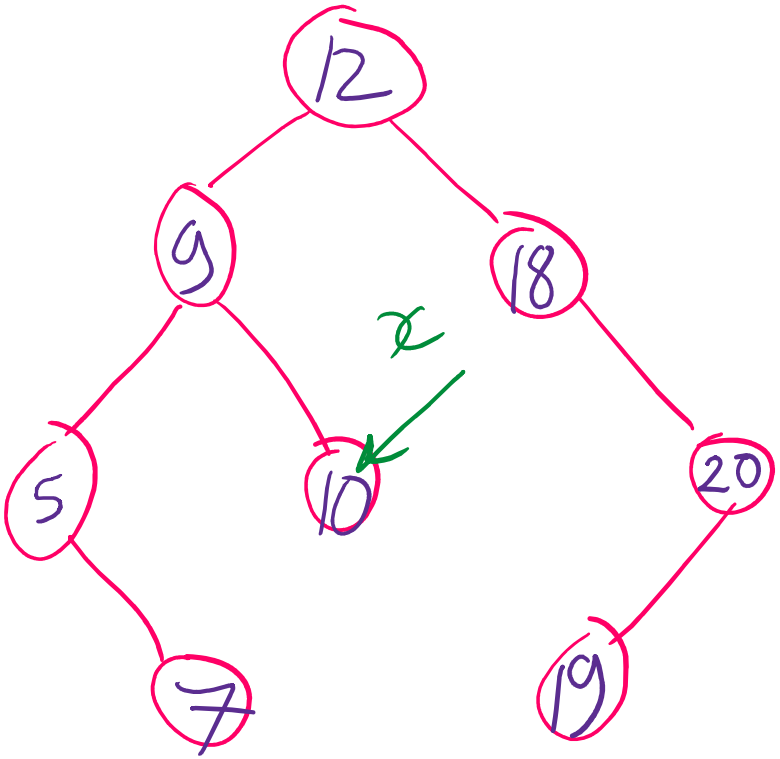
pop & visit

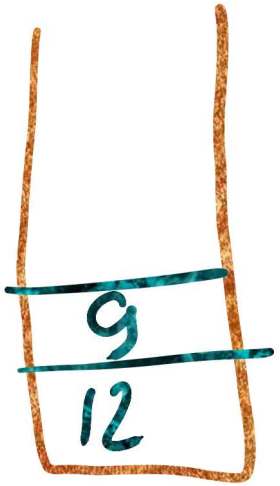
skip leftmost leaf

else x = parent.right

Visit order: 7, 5

Iterative Postorder traversal





Stack (explicit)

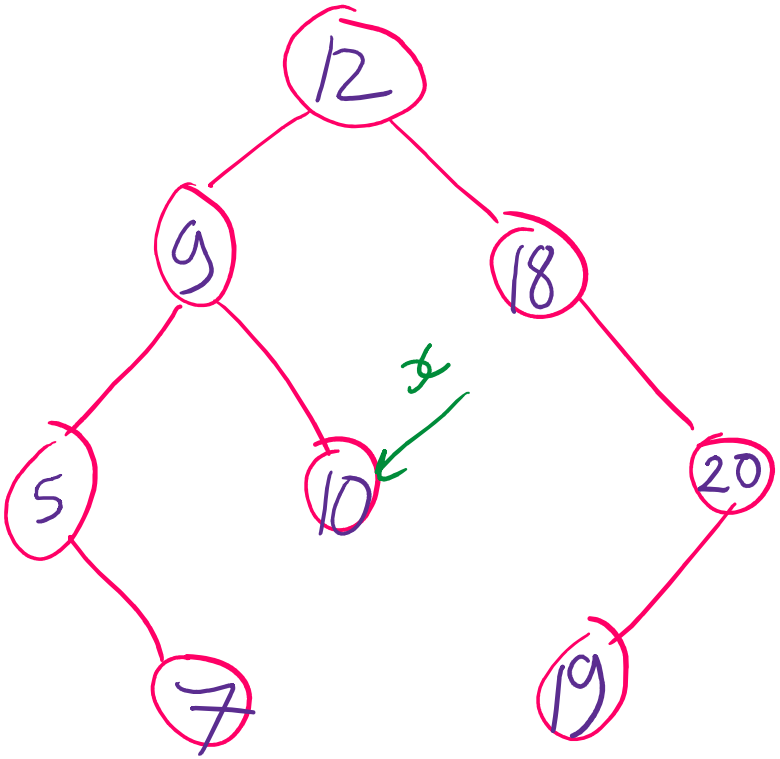
repeat until stack empty & x == null

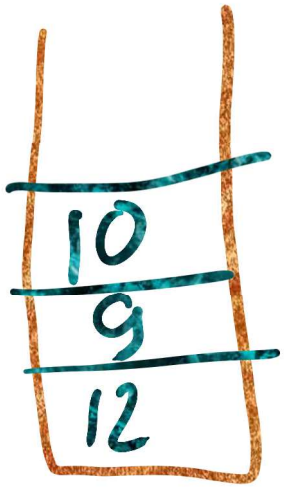
x = leftmost leaf
push to stack while moving down
visit x
if x is rightchild
pop & visit
skip leftmost leaf

else x = parent.right

Visit order: 7, 5

Iterative Postorder traversal





Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf

Push to stack while moving down

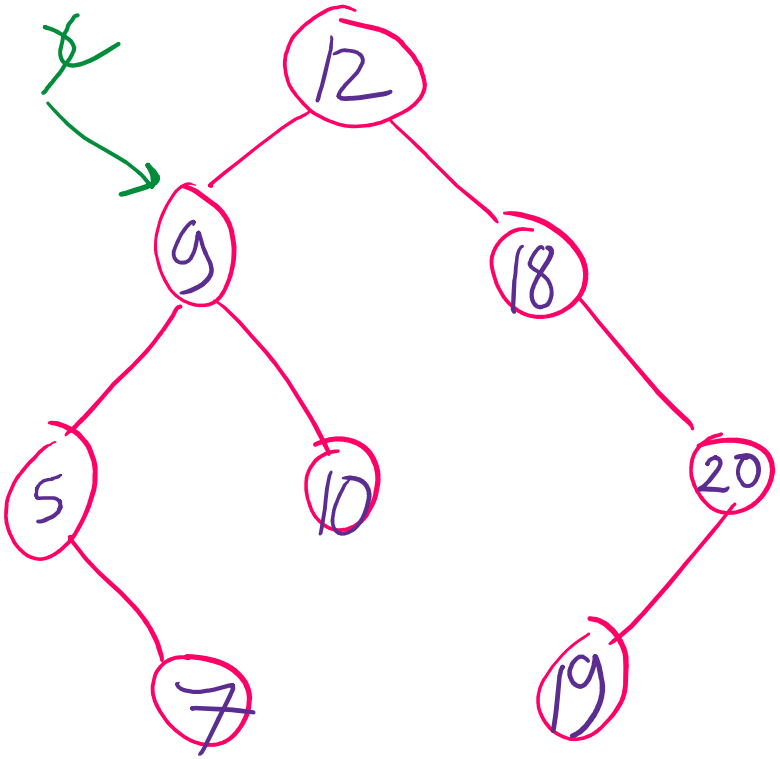
visit x

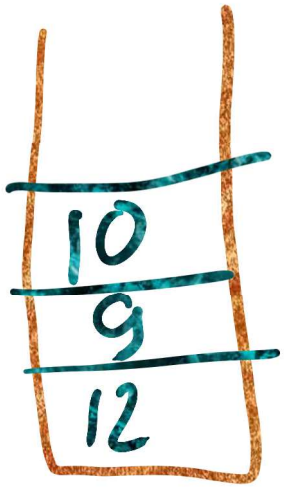
if x is rightchild
pop & visit
skip leftmost leaf

else x = parent.right

Visit order: 7, 5, 10

Iterative Postorder traversal





Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down

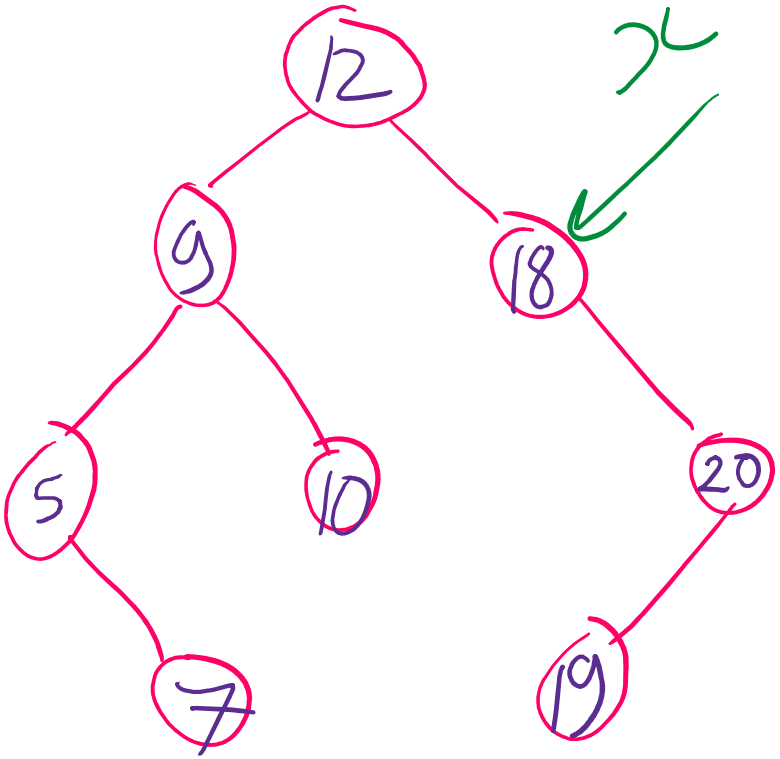
visit x

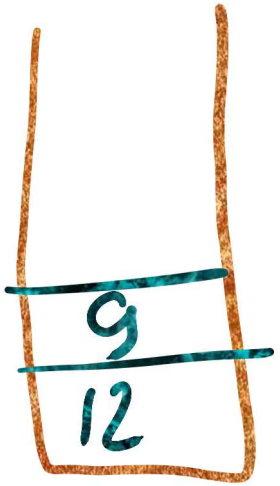
if x is right child
pop & visit
skip leftmost leaf

else x = parent.right

Visit order: 7, 5, 10, 9

Iterative Postorder traversal





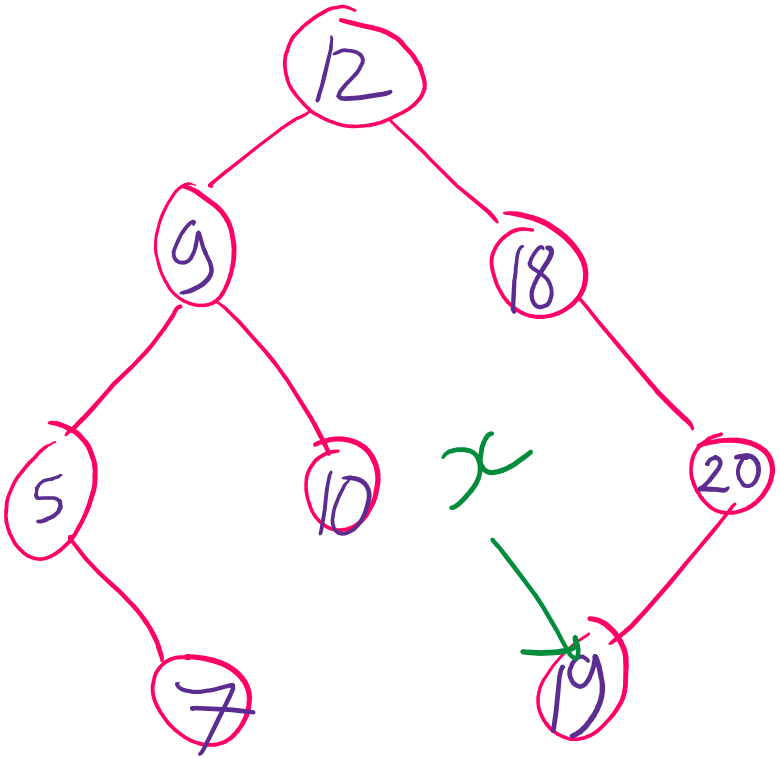
Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down
visit x
if x is rightchild
pop & visit
skip leftmost leaf
else x = parent.right

Visit order: 7, 5, 10, 9

Iterative Postorder traversal



Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf

Push to stack while moving down

visit x

if x is rightchild

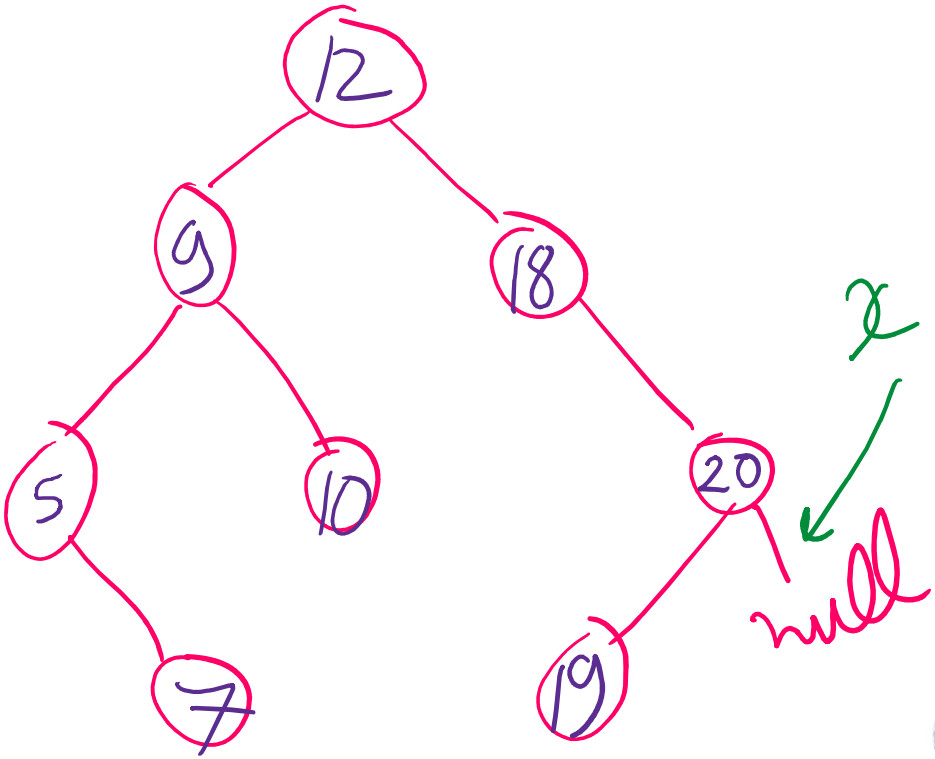
pop & visit

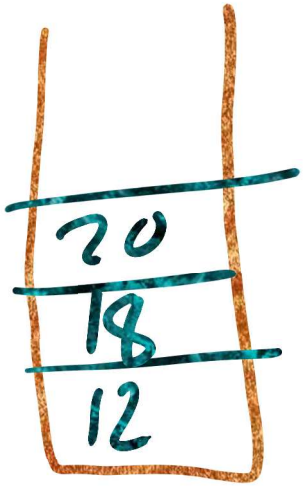
skip leftmost leaf

else x = parent.right

Visit order: 7, 5, 10, 9, 19

Iterative Postorder traversal





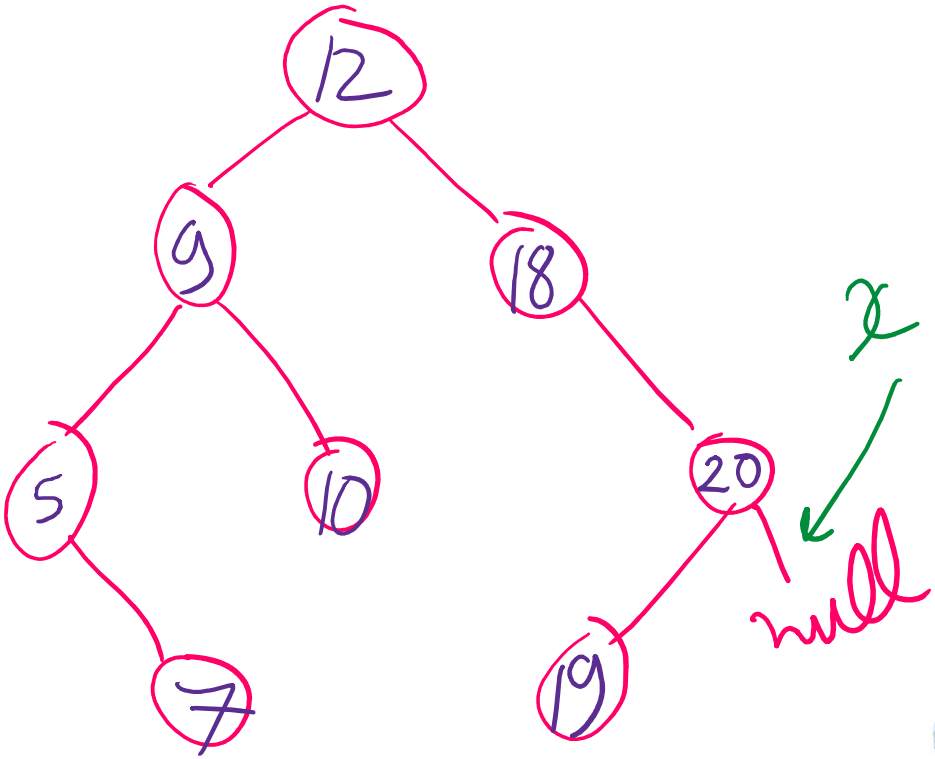
Stack (explicit)

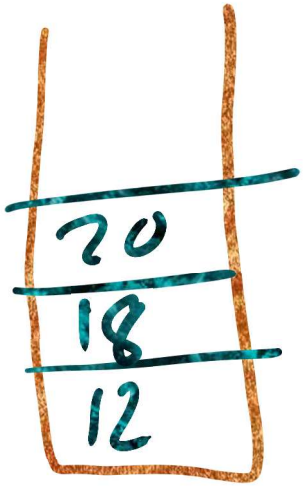
repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down
visit x
if x is rightchild
pop & visit
skip leftmost leaf
else x = parent.right

Visit order: 7, 5, 10, 9, 19

Iterative Postorder traversal





Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down

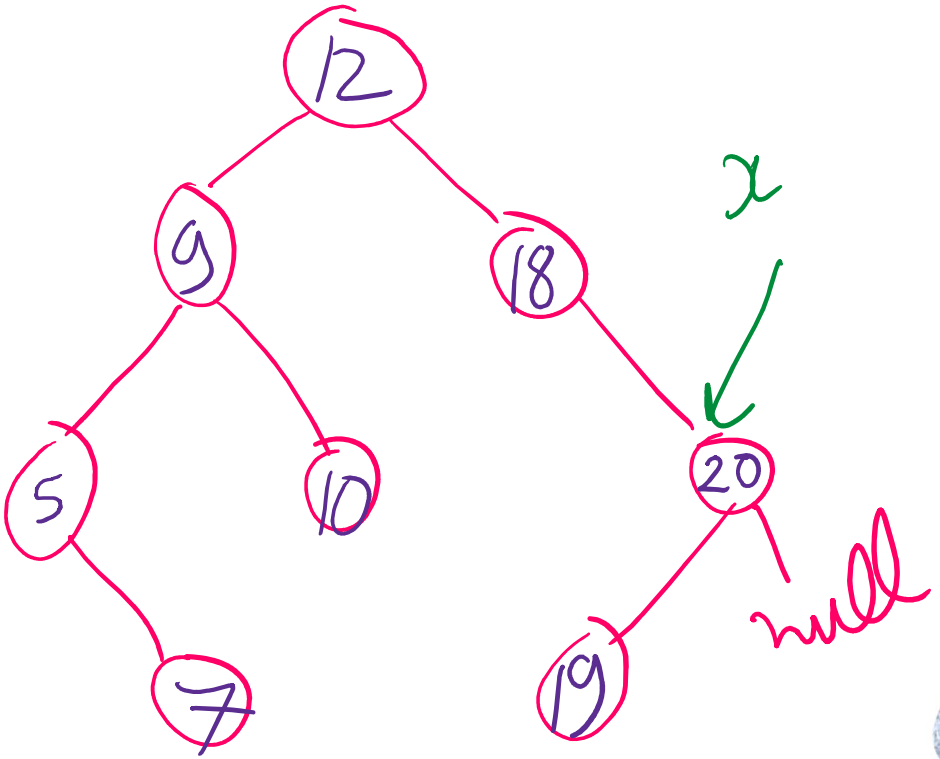
visit x

if x is right child
pop & visit
skip leftmost leaf

else x = parent.right

Visit order: 7, 5, 10, 9, 19

Iterative Postorder traversal



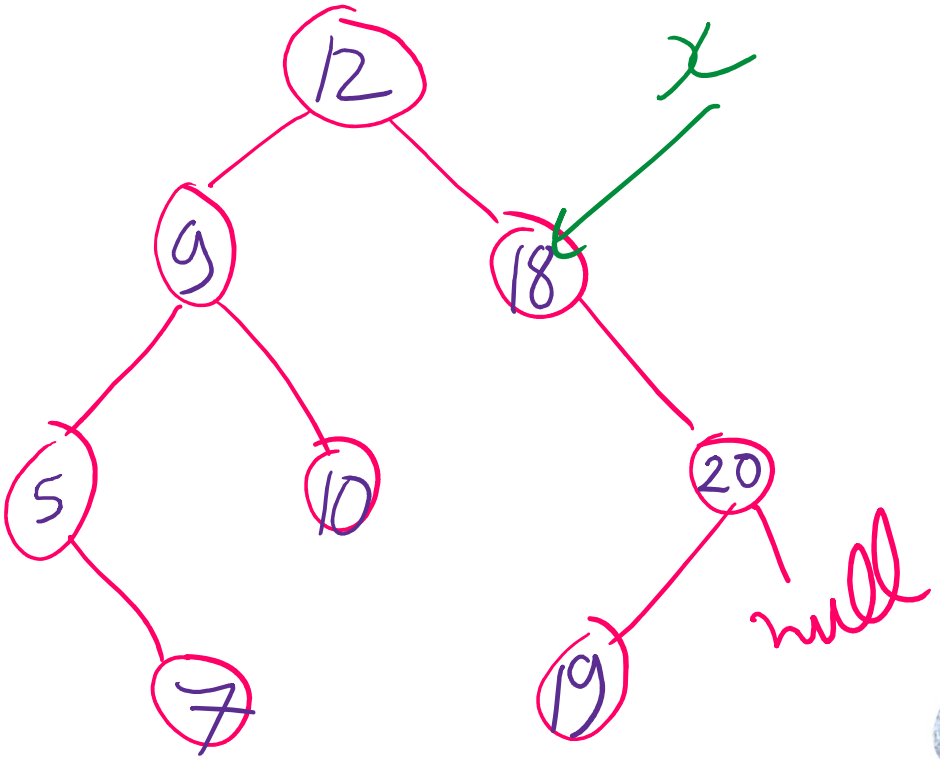
Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf
else x = parent.right

Visit order: 7, 5, 10, 9, 19, 20

Iterative Postorder traversal



Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down

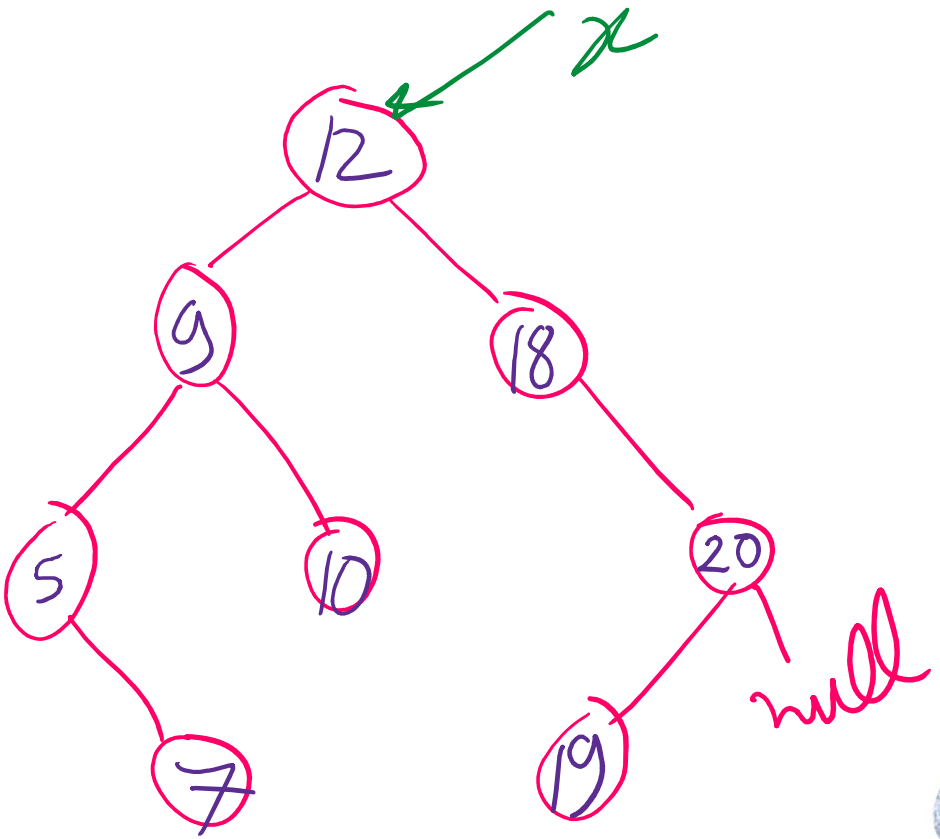
visit x

if x is right child
pop & visit
skip leftmost leaf

else x = parent.right

Visit order: 7, 5, 10, 9, 19, 20, 18

Iterative Postorder traversal



Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down

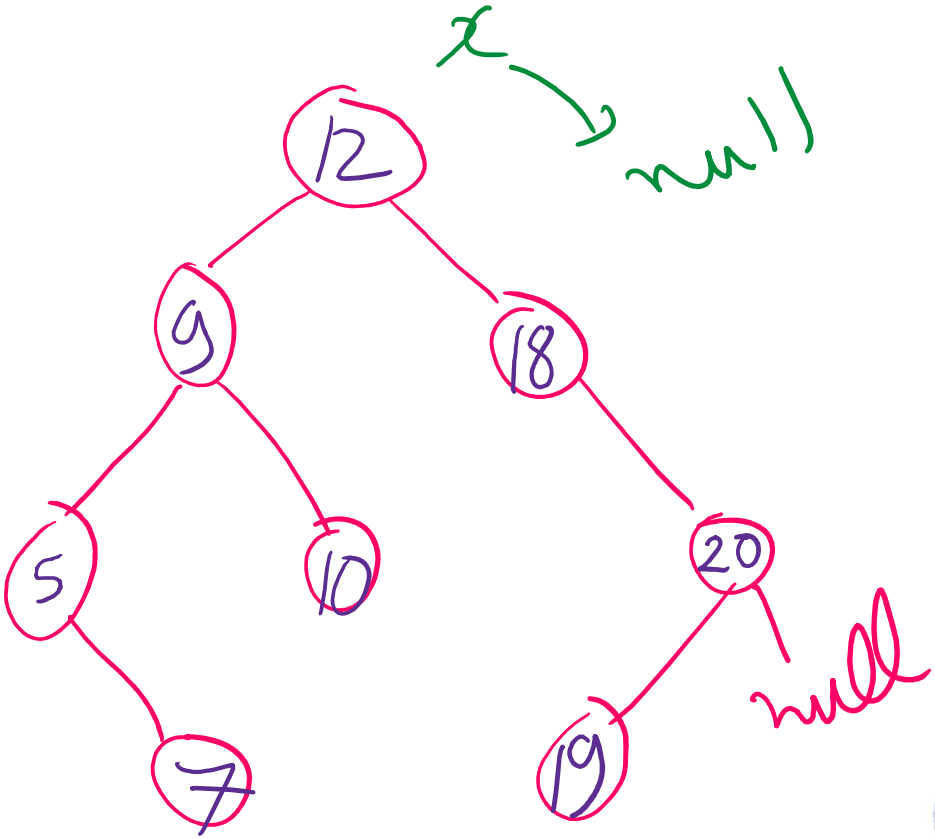
visit x

if x is right child
pop & visit
skip leftmost leaf

else x = parent.right

Visit order: 7, 5, 10, 9, 19, 20, 18, 12

Iterative Postorder traversal



Stack (explicit)

repeat until stack empty & x == null

x = leftmost leaf
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf
else x = parent.right

Visit order: 7, 5, 10, 9, 19, 20, 18, 12

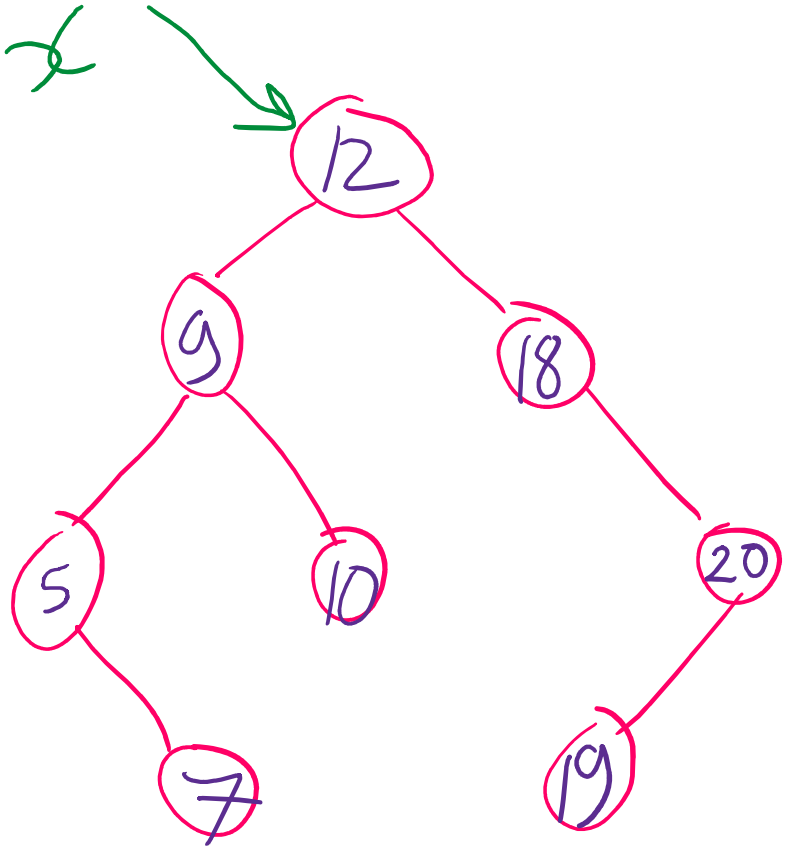
Traversals of a Binary Tree

- Preorder traversal
  - Visit root before we visit root's subtrees
- Inorder traversal
  - Visit root of a binary tree between visiting nodes in root's subtrees.
- Postorder traversal
  - Visit root of a binary tree after visiting nodes in root's subtrees
- Level-order traversal
  - Begin at root and visit nodes one level at a time
  - We will see the implementation when we learn Breadth-First Search of Graphs

Traversals of Binary Search Trees

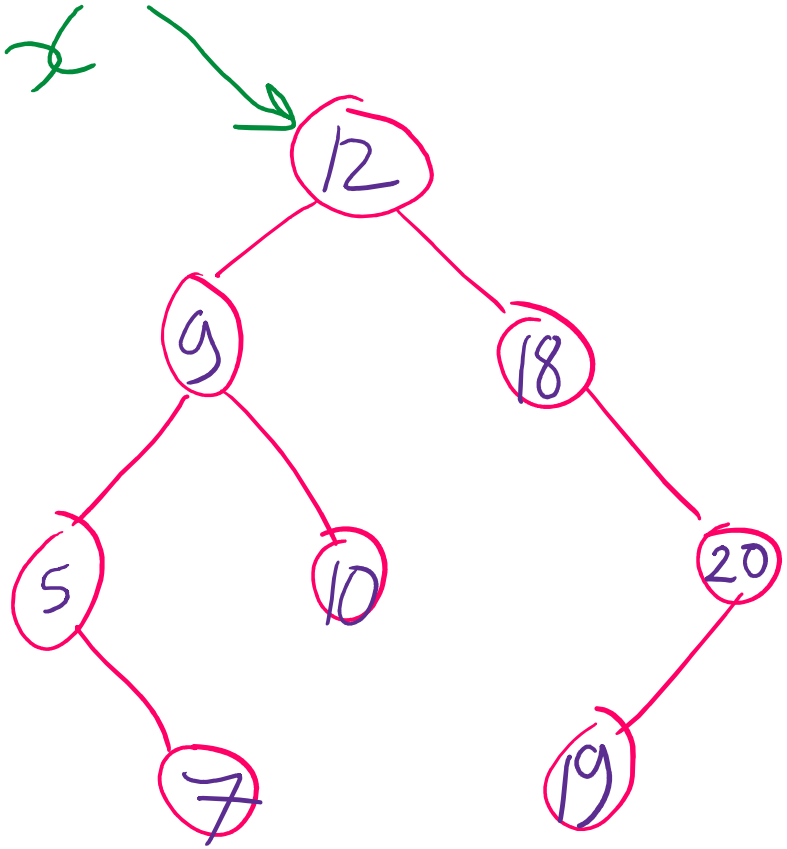
- Traversal for enumerating all items
- Traversal for finding an item

BST search using iteration




```
x = root
while(x != null){
    if equal break;
    if < x = x.left
    if > x = x.right
}
```

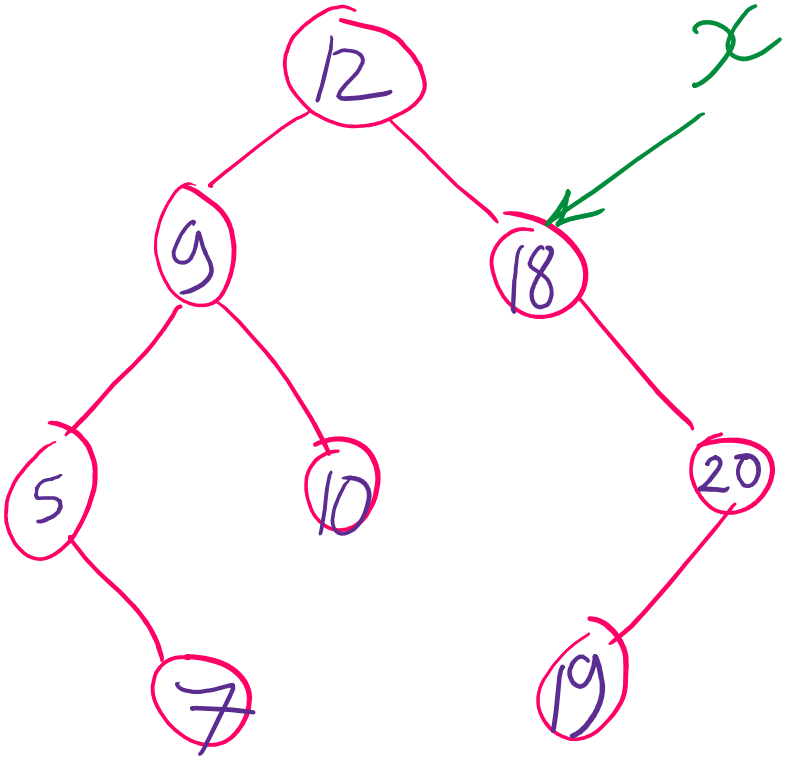
BST search using iteration




```
x = root
while(x != null){
    if equal break;
    if < x = x.left
    if > x = x.right
}
```

Search for 19

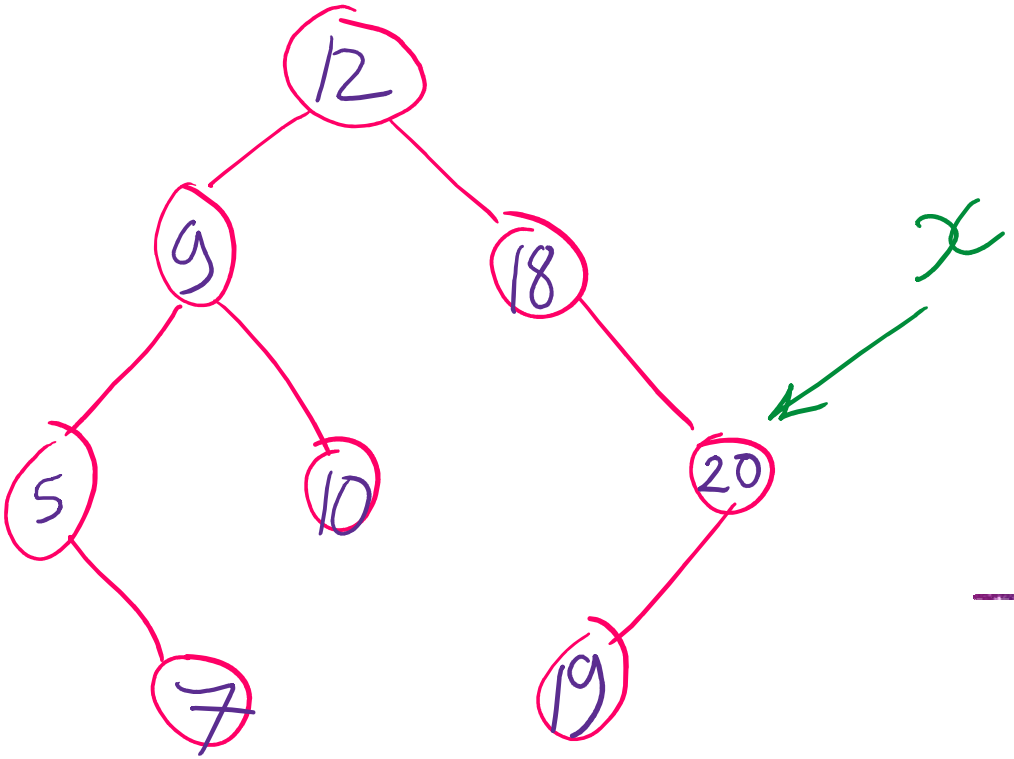
BST search using iteration




```
x = root
while(x != null){
    if equal break;
    if < x = x.left
    if > x = x.right
}
```

Search for 19

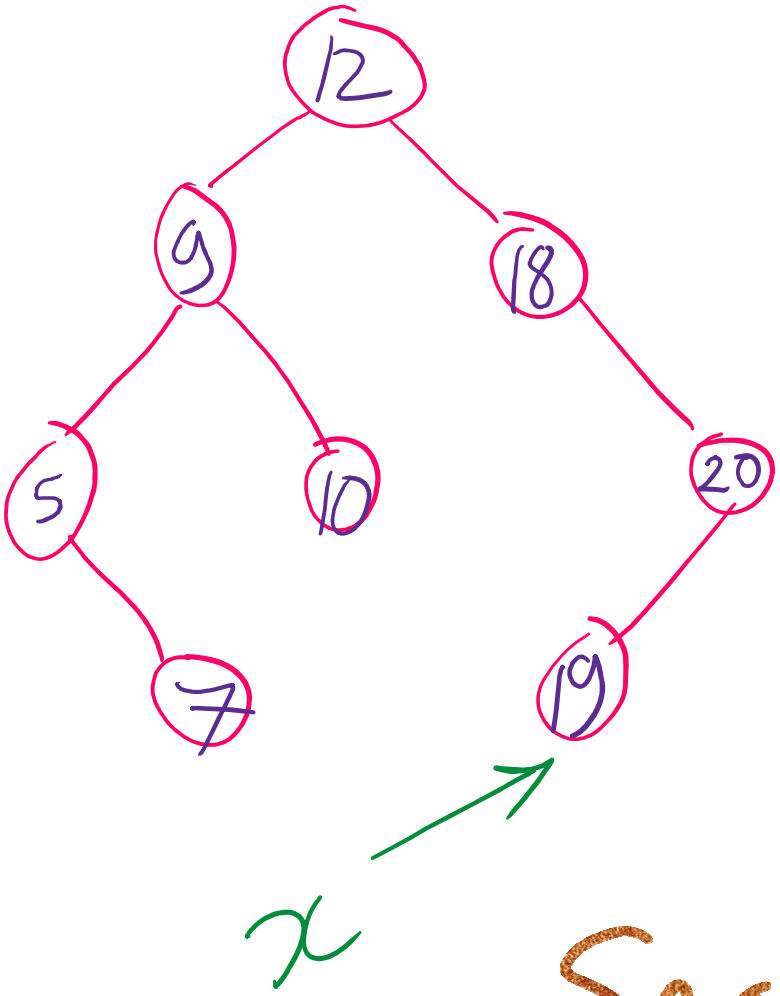
BST search using iteration




```
x = root
while(x != null){
    if equal break;
    if < x = x.left
    if > x = x.right
}
```

Search for 19

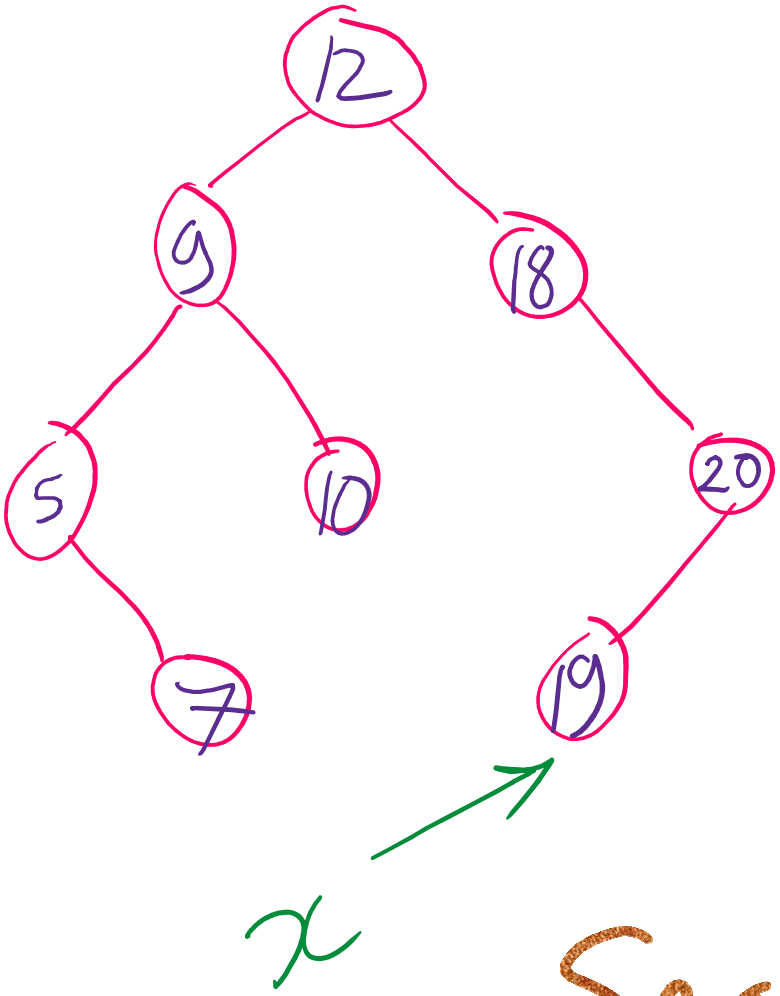
BST search using iteration




```
x = root
while(x != null){
    if equal break;
    if < x = x.left
    if > x = x.right
}
```

Search for 19

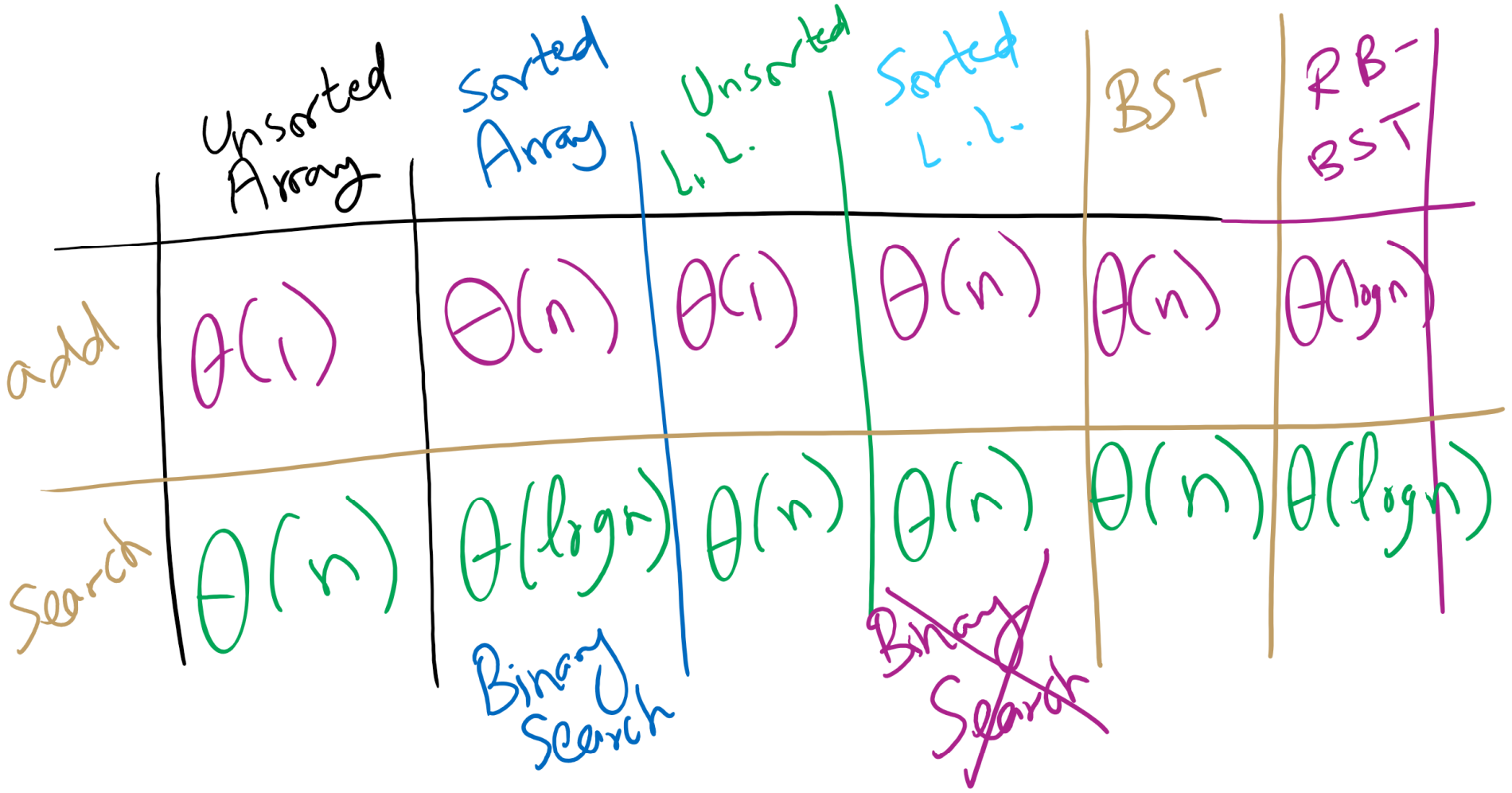
BST search using iteration




```
x = root
while(x != null){
    if equal break;
    if < x = x.left
    if > x = x.right
}
```

Search for 19

Symbol Table Implementations



Digital Search Trees (DSTs)

- Instead of looking at less than/greater than, lets go left right based on the bits of the key, so we again have 4 options:
  - Node ref is null, k not found
  - k is equal to the current node's key, k is found
  - current bit of k is 0, continue to left child
  - current bit of k is 1, continue to right child

Please submit your reflections by using the CourseMIRROR App

If you are having a problem with CourseMIRROR, please send an email to coursemirror.development@gmail.com