

test

May 7, 2025

## 1 001 Binary Search Tree (BST)

```
[24]: dict1 = {  
        'value' : 4,  
        'left' : None,  
        'right' : None  
    }  
  
dict1['value']
```

[24]: 4

```
[36]: root = {  
        'value' : 14,  
        'left' : {  
            'value' : 10,  
            'left' : {  
                'value' : 8,  
                'left' : None,  
                'right' : None  
            },  
            'right' : {  
                'value' : 12,  
                'left' : None,  
                'right' : None  
            }  
        },  
        'right' : {  
            'value' : 20,  
            'left' : {  
                'value' : 19,  
                'left' : None,  
                'right' : None  
            },  
            'right' : {  
                'value' : 21,  
                'left' : None,  
                'right' : None  
            }  
        }  
    }
```

```

        'right' : None
    }
}

print(root['value'])
print(root['left']['value']) # ----- since 10<root ; left
print(root['right']['value']) # ----- since 20>root ; right

```

14  
10  
20

```

[ ]: # to access to the other next one
print(root['left']['left']['value'])

print(root['right']['right']['value'])

```

8  
21

- similarly

```

temp = self.root

if (some thing case):

    temp = temp.left (in this way, we must switch to left)

else:

    temp = temp.right

```

## 2 003 Big O

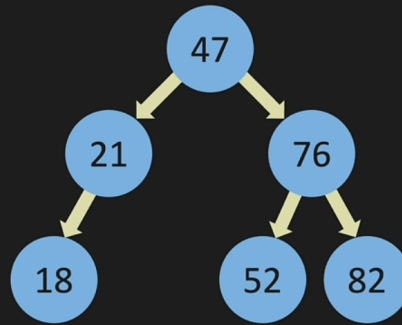
[ ]:

## 3 004 BST Constructor

```

create new_node
if root == None then root = new_node
temp = self.root
while loop
    if new_node == temp return False
    if < left else > right
    if None insert new_node else move to next

```



If the new node is ever equal to temp.

### ✓ The while loop i

python

```

while condition:
    # run this block

```

- It **checks the condition**
- If the condition is **True**
- It loops back and **keep**
- If the condition becom

```

[1]: class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

```

### ✓ The while loop in Python:

python

Copy

Edit

```

while condition:
    # run this block

```

- It **checks the condition**
- If the condition is **True**, it runs the block inside
- It loops back and **keeps checking** the condition each time
- If the condition becomes **False**, it stops the loop


```

create new_node
temp = self.root
while loop
    if < left else > ri
    if None insert new_

```

The first o  
this happ

### 3.1

 **The `while` loop in Python:**

python


Copy

Edit

```
while condition:
    # run this block
```

- It **checks the condition**
- If the condition is **True**, it runs the block inside
- It loops back and **keeps checking** the condition each time
- If the condition becomes **False**, it stops the loop

### 3.2

 **The `while` loop in Python:**

python

Copy

Edit

```
while condition:
    # run this block
```

- It **checks the condition**
- If the condition is **True**, it runs the block inside
- It loops back and **keeps checking** the condition each time
- If the condition becomes **False**, it stops the loop

```
create new_node
temp = self.root
while loop
    if < left else > ri
    if None insert new_
```

The first o  
this happ

```
[2]: class Node:
      def __init__(self, value):
          self.value = value
          self.left = None
          self.right = None

      class BinaryTree:
          def __init__(self):
              self.root = None
```

```
my_binarytree = BinaryTree()
```

## 4 005 BST Insert - Intro.mp4

### ✓ The while loop in Python:

python

Copy

Edit

```
while condition:
    # run this block
```

- It **checks the condition**
- If the condition is **True**, it runs the block inside
- It loops back and **keeps checking** the condition each time
- If the condition becomes **False**, it stops the loop

```
create new_node
if root == None then
    temp = self.root
while loop
    if new_node ==
    if < left else > ri
    if None insert new_
```

If the

### 4.0.1 Edge cases

```
create new_node
temp = self.root
while loop
    if < left else > right
    if None insert new_node else move to next
```

root → None

The first one being if we have an empty tree, if this happens we'll just set root equal to the new

### ✓ The while loop i

python

```
while condition:
    # run this block
```

- It **checks the condition**
- If the condition is **True**
- It loops back and **keep**
- If the condition becom

## 5 006 BST Insert - Code.mp4

```
[48]: class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self):
        self.root = None    # Jaba binaryTree class ko instance banainxa; Yo
        ↪ class ko / Or / Yo tree ko eauta matra self.root hunxa !! Be caareful

    def insert(self, value):
        new_node = Node(value)
        if self.root is None:
            self.root = new_node
            return True

        temp = self.root
        while (True):
            if temp.value == new_node.value:
                return False

            if new_node.value < temp.value :
                if temp.left is None:
                    temp.left = new_node
                    return True
                temp = temp.left
            else:
                if temp.right is None:
                    temp.right = new_node
                    return True
                temp = temp.right

        # def print_tree

my_binarytree = BinaryTree()
my_binarytree.insert(10)
my_binarytree.insert(20)
my_binarytree.insert(5)

print(my_binarytree.root)
print(my_binarytree.root.left)
print(my_binarytree.root.right)
```

```
print(my_binarytree.root.value)
print(my_binarytree.root.left.value)
print(my_binarytree.root.right.value)
```

```
<__main__.Node object at 0x79b64ebb84c0>
<__main__.Node object at 0x79b64ebba110>
<__main__.Node object at 0x79b64f56a950>
10
5
20
```

Redo the insert method:

```
[49]: class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self):
        self.root = None

    def insert(self, value):
        # edge cases
        new_node = Node(value)
        if self.root == None:
            self.root = new_node
            return True

        temp = self.root
        while (True): # while True sets up an infinite loop.
            if temp.value == new_node.value:
                return False

            if new_node.value < temp.value:
                if temp.left is None:
                    temp.left = new_node
                    return True

                temp = temp.left
            else:

                if temp.right is None:
                    temp.right = new_node
                    return True
                temp = temp.right
```

```

my_binarytree = BinaryTree()
my_binarytree.insert(5)

my_binarytree.insert(6)

my_binarytree.insert(4)

my_binarytree.insert(3)

print(my_binarytree.root.value)
print(my_binarytree.root.left.value)
print(my_binarytree.root.right.value)
print(f'-----')
print(my_binarytree.root.left.left.value)

```

```

5
4
6
-----
3

```

**\*\*return True\*\*** or **\*\*return False\*\*** is for the function, not the while loop.

Here's how it works:

When Python sees a return inside a function:

It exits the entire function immediately,

And sends the return value (True or False) back to wherever the function was called,

The while loop also ends because the function is no longer running.

```
while True:
```

```
    if condition_met:
```

```
        return True # ends the whole function here

```



## ✓ The `while` loop in Python:

python

Copy

Edit

```
while condition:
    # run this block
```

- It checks the condition
- If the condition is **True**, it runs the block inside
- It loops back and **keeps checking** the condition each time
- If the condition becomes **False**, it stops the loop

- that means while (True):

Function 1st work: TO check the Condition if it is True or not

but condition = True

And the true is always true

Therefore, it setup an infinite function

## 6 007 BST Contains.mp4

```
[ ]: class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self):
        self.root = None

    def insert(self, value):
        new_node = Node(value)

        temp = self.root

        if self.root is None:
            self.root = new_node
            return True

        while(True):
```

```

        if temp.value == new_node.value:
            return False

        if new_node.value < temp.value :
            if temp.left is None:
                temp.left = new_node
                return True
            temp = temp.left
        else:
            if temp.right is None:
                temp.right = new_node
                return True
            temp = temp.right

    def contain(self, value):
        if self.root == None :
            return None

        temp = self.root

        while temp is not None:
            if temp.value == value :
                return temp

            if value < temp.value:
                temp = temp.left
            else:
                temp = temp.right

#
↪ -----

my_binarytree = BinaryTree()
my_binarytree.insert(5)

my_binarytree.insert(6)

my_binarytree.insert(4)

my_binarytree.insert(3)

print(my_binarytree.root.value)
print(my_binarytree.root.left.value)
print(my_binarytree.root.right.value)
print(f'-----')
```

```
print(my_binarytree.root.left.left.value)

print(f'-----Now finding -----')
address = my_binarytree.contain(3)

print(address.value)
```

```
5
4
6
-----
3
-----Now finding -----
3
```

### 6.0.1 Write a clean code

[ ]: