Binary search tree (BST):

Binary tree → Max 2 child

BST → left child < Parent data < right child
↳ each node empty node

is this a BST?

Q→ is this BST? ×

BST:
{ Size = 5
height = 3
Root data = 50 }

Node {
int data
Node left
Node right
}

display → 25<50<75
12<25<37
x<12 ×
x>12 ✓
12<x<25 ✓
x<25 ✓
x>25 ✓
→ display.

add(50);
add(12);
add(75);

add(50);

add(50);
Root
Node Root;
int size;
int height;
int(root.data);
75
12
37

public: BST {
clear Node {}

⇒ height = O(log n)

37<50
50 → left.
25 → add(37)

add(O(log n))

$2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^h = N$.

$\frac{2^h - 1}{2 - 1} = N$.

$2^h = N + 1$

$\log_2 2^h = \log_2 (N+1)$.

$h = \log_2 N$

↳ Balanced add binary search tree.

```
public void add(int data){
    if (this.Root == null){
        this.Root = new Node(data);
        this.Size++;
    }
    this.add(this.Root, data);
}
private void add(Node node, int data){
    if (data < node.data){
        if (node.left == null){
            node.left = new Node(data, null, null);
            this.Size++;
        } else {
            this.add(node.left, data);
        }
    } else if (data > node.data){
        if (node.right == null){
            node.right = new Node(data, null, null);
            this.Size++;
        } else {
            this.add(node.right, data);
        }
    } else {
        // do nothing
    }
}
```

add(10);
add(20);
add(30);
add(40);
add(50);

add → O(n)

BST tree {
Given:
Skew
}

add();

Max → Binary tree
↳ O(n) worst
↳ O(log n)

```
public: int max(){
    this.max(this.Root);
}
private int max(Node node){
    int max = node.data;
    if (node.right != null){
        max = this.max(node.right);
    }
    return max;
}
```


O(log n)

```
public void remove(int data){
    this.Root = this.remove(this.Root, data);
}
private Node remove(Node node, int data){
    if (data < node.data){
        node.left = this.remove(node.left, data);
        return node;
    } else if (data > node.data){
        node.right = this.remove(node.right, data);
        return node;
    } else {
        if (node.left == null && node.right == null){
            return null;
        } else if (node.right == null){
            this.size--;
            return node.left;
        } else if (node.left == null){
            this.size--;
            return node.right;
        } else {
            int lmax = this.max(node.left); // 37
            node.data = lmax;
            node.left = this.remove(node.left, lmax);
            return node;
        }
    }
}
```

Resume at 8:15pm

{ (node.data) > max : node.data = max;
return false; }

(1) → 12, 25, 37, 50, 62, 75, 87

↳ find
traversal & check
whether it is BST or not?
↳ BST → Sorted
↳ NO BST → not Sorted

```
public boolean isBst(){
    return this.isBst(this.Root, -∞, ∞);
}
private boolean isBst(Node node, int min, int max){
    if (node == null){
        return true;
    }
    if (node.data < min || node.data > max){
        return false;
    } else if (!this.isBst(node.left, min, node.data)){
        return false;
    } else if (!this.isBst(node.right, node.data, max)){
        return false;
    } else {
        return true;
    }
}
```

arr = { 10, 15, 20, 25, 30, 35, 40 }
0 1 2 3 4 5 6

BST → add

BST
↳ Balanced

arr = { 10, 15, 20, 25, 30, 35, 40 }
0 1 2 3 4 5 6

BST(int[] arr){
this.Root = Construct(arr, 0, arr.length - 1);
}

```
private Node Construct(int[] arr, int pi, int[] idx){
    if (i > pi){
        return null;
    }
    int mid = (i + pi)/2;
    Node node = new Node(arr[mid], null, null);
    node.left = this.Construct(arr, i, mid - 1);
    node.right = this.Construct(arr, mid + 1, pi);
    return node;
}
```

BST → BT

Friday → Recursion → Subsequence
↳ Backtrack