# Data Structures

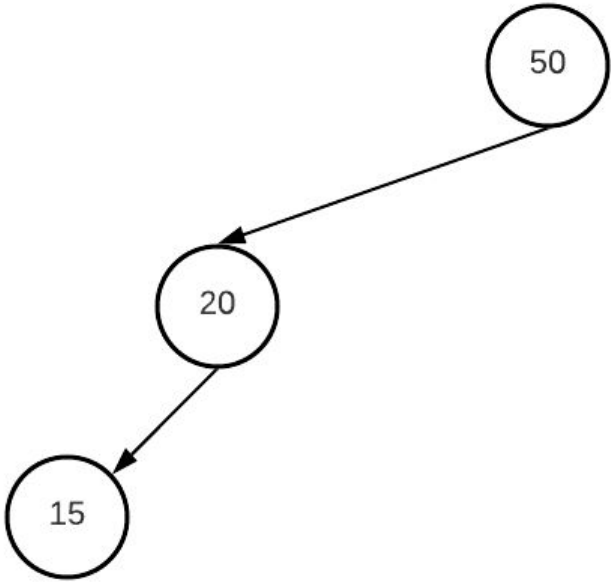
# BST Insertion

**Mostafa S. Ibrahim**
*Teaching, Training and Coaching since more than a decade!*

*Artificial Intelligence & Computer Vision Researcher*
*PhD* from Simon Fraser University - Canada
*Bachelor / Msc* from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

# Insertion

- Assume we have the following BST
- How can we insert values: 45, 35?
- We need to find the right parent and add the value
- Try to code: void **insert**(int target)



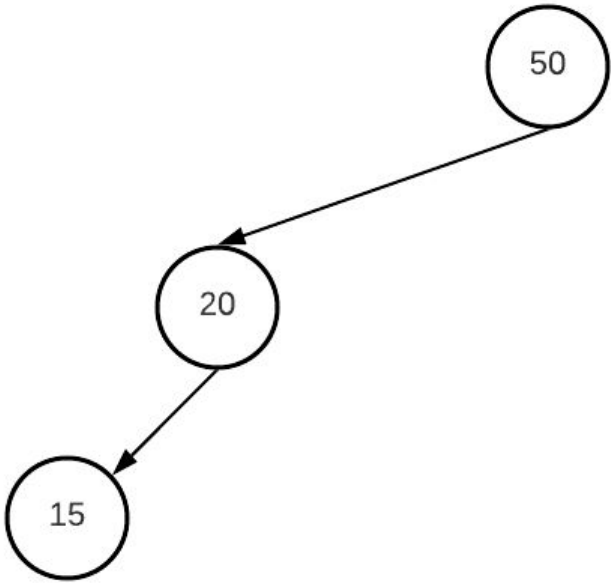
# Insertion

- Given value, we identify where to insert

```
void insert(int target) {
    if (target < data) {
        if (!left)
            left = new BinarySearchTree(target);
        else
            left->insert(target);
    } else if(target > data){
        if (!right)
            right = new BinarySearchTree(target);
        else
            right->insert(target);
    } // else: exists already
}
```

# Insert 45

- At 50: go left
- At 20: go right
  - No right
  - Create right(45)

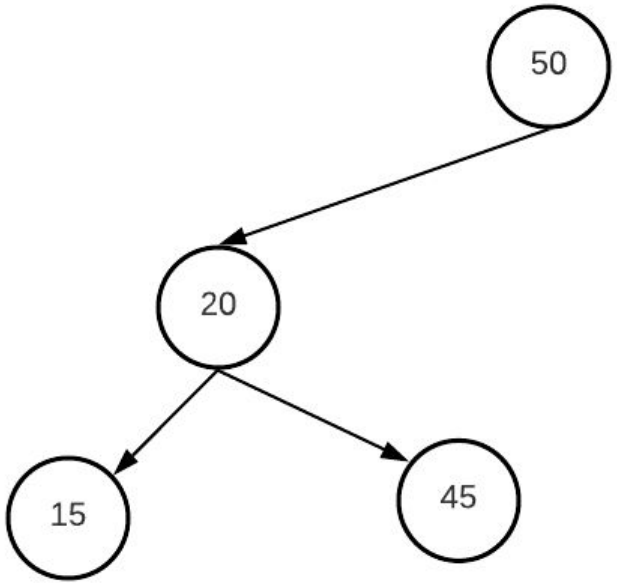



```
void insert(int target) {
    if (target < data) {
        if (!left)
            left = new BinarySearchTree(target);
        else
            left->insert(target);
    } else if(target > data){
        if (!right)
            right = new BinarySearchTree(target);
        else
            right->insert(target);
    } // else: exists already
}
```

# Insert 35

- At 50: go left
- At 20: go right
- At 45: go left
  - No left
  - Create left(35)

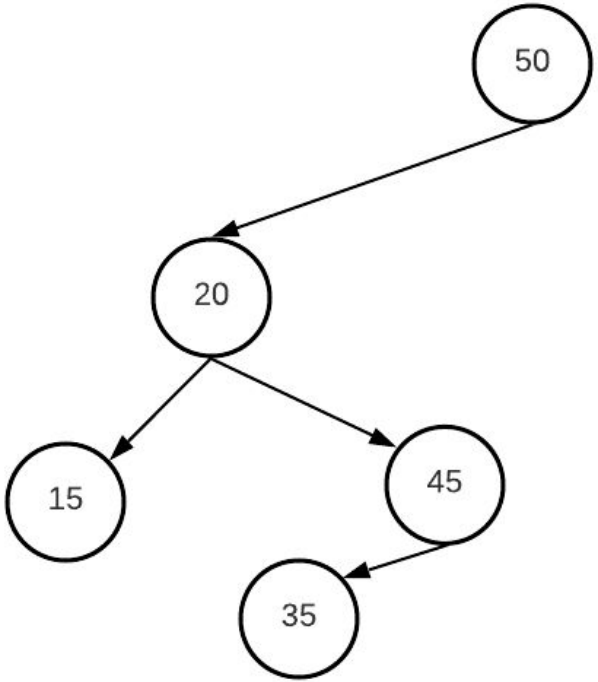
```
void insert(int target) {
    if (target < data) {
        if (!left)
            left = new BinarySearchTree(target);
        else
            left->insert(target);
    } else if(target > data){
        if (!right)
            right = new BinarySearchTree(target);
        else
            right->insert(target);
    } // else: exists already
}
```



# Insert 70

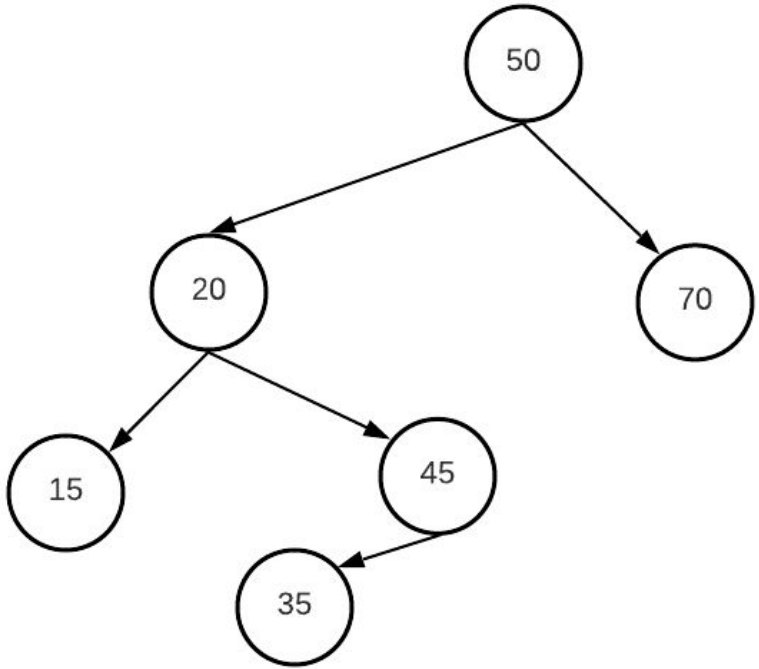
- At 50: go right
- No right
  - Create right(70)

```
void insert(int target) {
    if (target < data) {
        if (!left)
            left = new BinarySearchTree(target);
        else
            left->insert(target);
    } else if(target > data){
        if (!right)
            right = new BinarySearchTree(target);
        else
            right->insert(target);
    } // else: exists already
}
```



# Insert 60

- At 50: go right
- At 70: go left
  - No left
  - Create left(60)

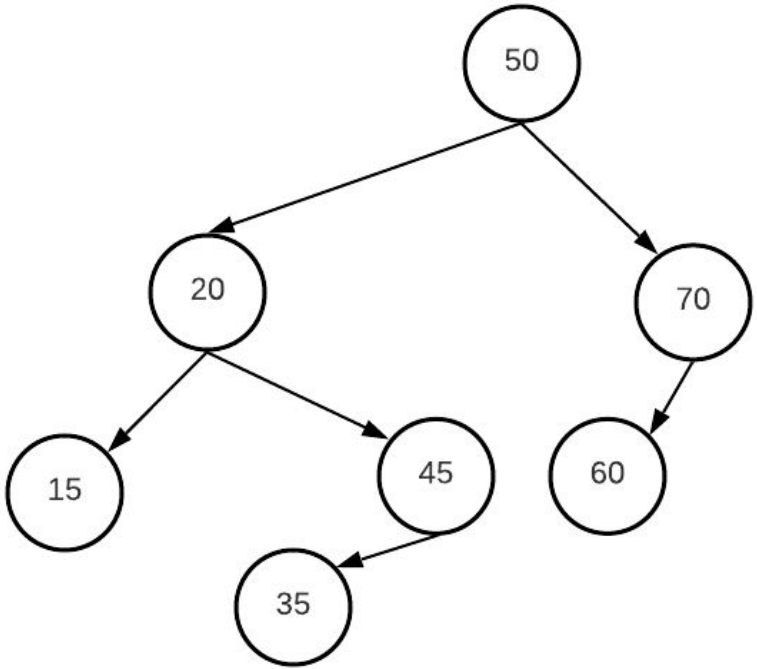



```
void insert(int target) {
    if (target < data) {
        if (!left)
            left = new BinarySearchTree(target);
        else
            left->insert(target);
    } else if(target > data){
        if (!right)
            right = new BinarySearchTree(target);
        else
            right->insert(target);
    } // else: exists already
}
```

# Insert 73

- At 50: go right
- At 70: go right
  - No right
  - Create right(73)

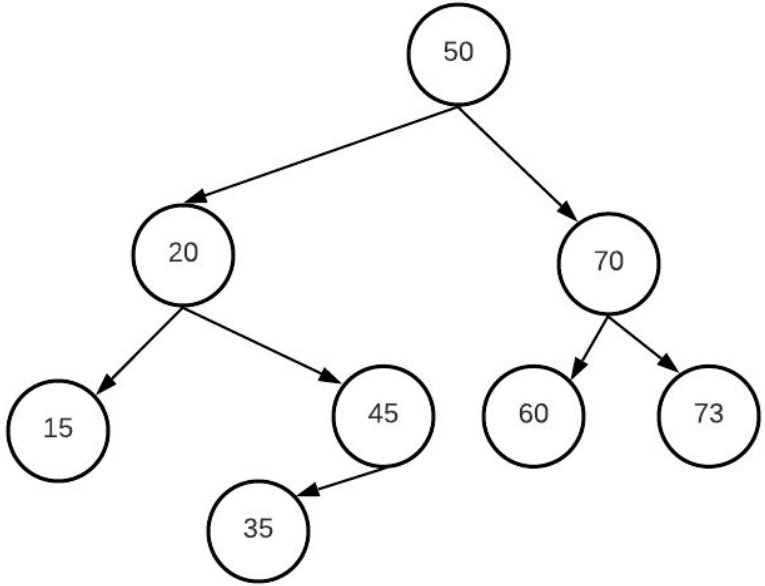
```
void insert(int target) {
    if (target < data) {
        if (!left)
            left = new BinarySearchTree(target);
        else
            left->insert(target);
    } else if(target > data){
        if (!right)
            right = new BinarySearchTree(target);
        else
            right->insert(target);
    } // else: exists already
}
```



# Insertion complexity

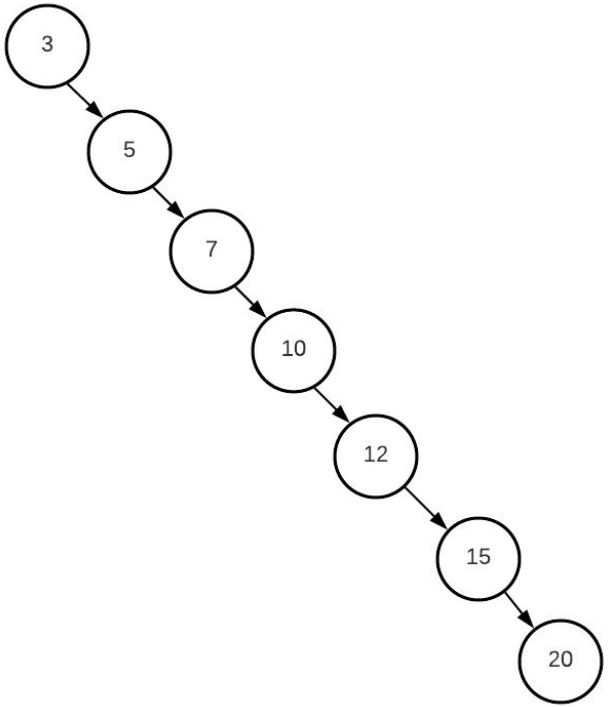
- O(h) time and memory
  - Time as we go chain for an element
  - Memory: Auxiliary for stack

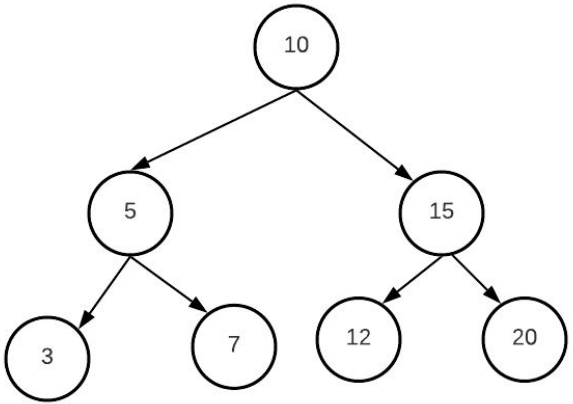
```
void insert(int target) {
    if (target < data) {
        if (!left)
            left = new BinarySearchTree(target);
        else
            left->insert(target);
    } else if(target > data){
        if (!right)
            right = new BinarySearchTree(target);
        else
            right->insert(target);
    } // else: exists already
}
```



# Order of insertion

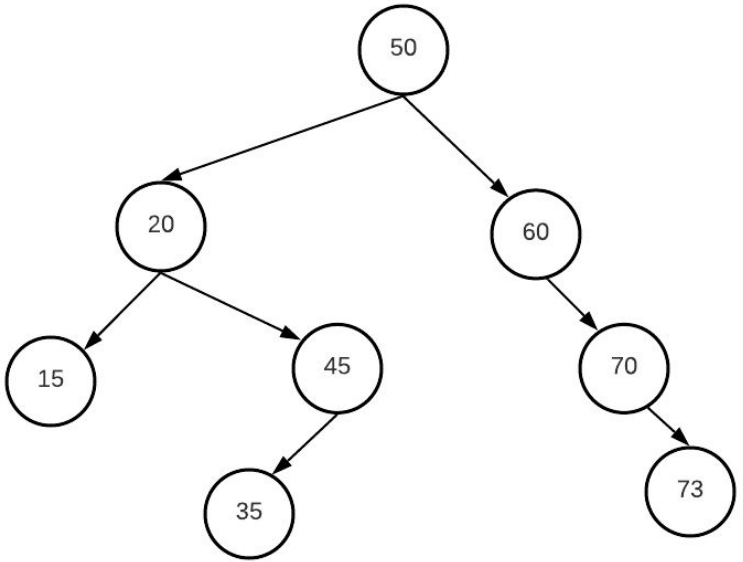
- Tree shape depends on **insertion order**
- In **best case**, we can have balanced tree
- But in **worst cast** it could be degenerate!
- Shape affects insertion/search time
  - From O(logn) to o(n)



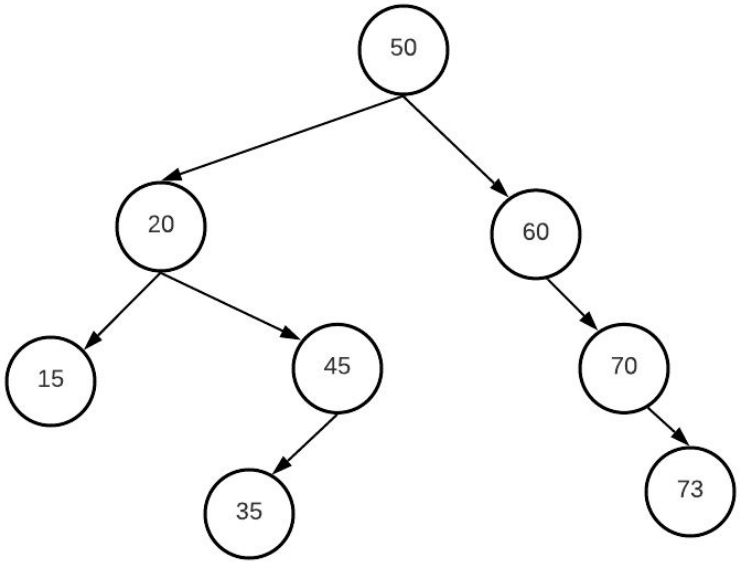
# Minimum?

- Given a subtree root, what is the minimum value in it? Max value?



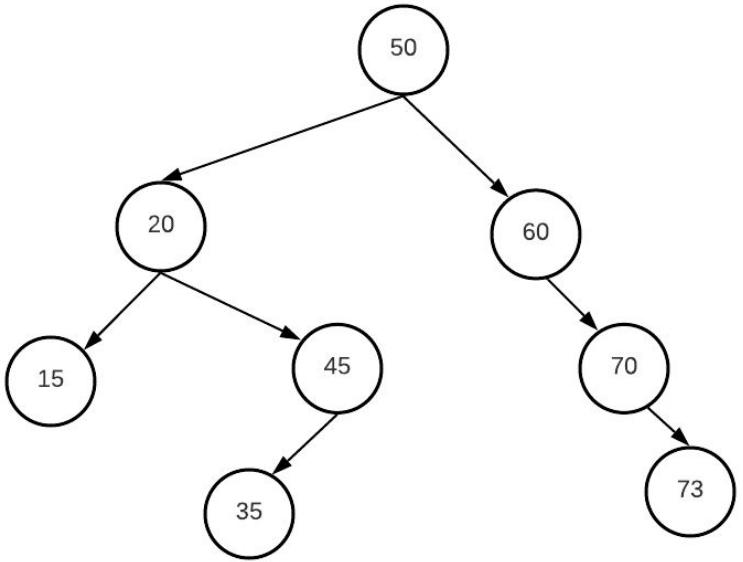
# Inorder **Successor** in Binary Search Tree?

- Given node x, find node y that is the smallest y > x  [in O(h)]
- In other words: get inorder traversal
  - 15 20 35 45 50 60 70 73
  - It is the **next value** in the array
  - 70 ⇒ 73
  - 20 ⇒ 35
  - 45 ⇒ 50
  - 35 ⇒ 45
- Think for 15 min in 2 cases:
  - 1) x has right  2) x doesn't have right



# Node deletion?

- Think for 20 minutes how can we delete a node given value?
  - The remaining tree must be BST
  - Utilize successor idea
- Consider 3 cases:
  - 73 [no children]
  - 60 [1 child]
  - 20 [2 children]



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."