Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Insertion in BST

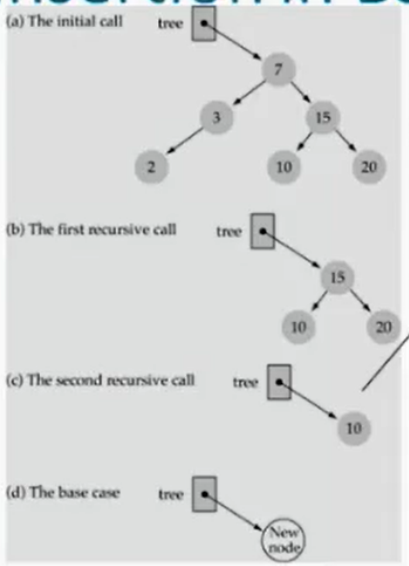
```
insertKey(tree, key) {
    if tree = NULL
        create new node (key, NULL, NULL);
    else tree->data < key
        tree->right = insertKey(tree->right, key);
    else
        tree->left = insertKey(tree->left, key);

}
```

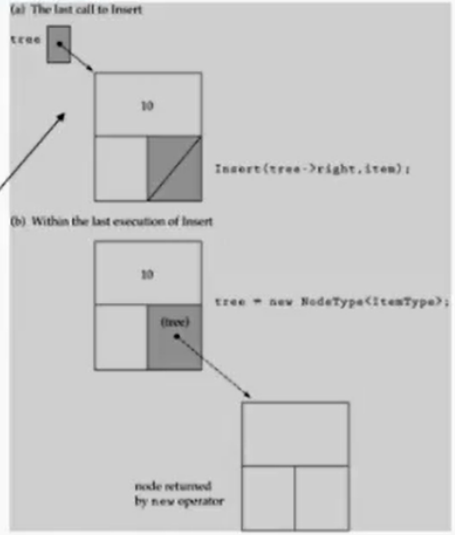
Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Insert 11

Insertion in BST (cont.)





A

Alka Jindal

Insertion in BST : Code

```
typedef struct Node {  //structure of a node
   int data;
   struct Node left;
   struct Node right;
} * Nptr;

Nptr newNode(int data, Nptr left, Nptr right) { //create a new node
   Nptr nNode = (Nptr) malloc(sizeof(struct Node));
   nNode->data = data;
   nNode->left = left;
   nNode->right =right;
   return nNode;
}
```

wab-jvws-tfr (2021-10-30 at 01:45 GMT-7)

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Insert in BST : Code (cont.)

```
Nptr insertKey(Nptr tree, int data) {
   if (tree == NULL) {
      return newNode(data, NULL, NULL);
   }
   else if (tree->data < data) {
      tree->right = insertKey(tree->right, data);
   }
   else {
      tree->left = insertKey(tree->left, data);
   }
   return tree;
}
```

A

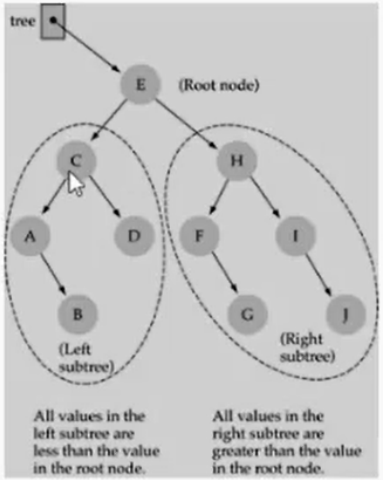
Alka Jindal

21:42 / 44:07

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Search in BST: Algorithm

```
searchBST(tree, key) {
   if tree = NULL {
      print("Not found");
      return;
   }
   else if tree->data = key
      print("Key found");
   else if tree->data >key
      searchBST(tree->left, key);
   else
      searchBST(tree->right, key);
}
```



A

Alka Jindal

Search in BST : Code

```
Nptr searchBST(Nptr tree, int key) {  //Search key in tree
   if (tree == NULL) {
      printf("\nkey not found");  //print not found if key not present
      return NULL;
   }
   else if (tree->data == key) {
      printf("\n Key found");  //print found if key not present
      return tree;
   }
   else if (tree->data < key)
      return searchBST(tree->right, key);  //Search right sub-tree
   else
     return searchBST(tree->left, key);  //Search left sub-tree
}
```

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Search in BST: Time Complexity

- Search in BST depends on the height of the tree
- In worst case,

  Height of a binary tree with n nodes = O(n)

  Thus, Time complexity of search = O(n)

- In general,

  Height of a binary tree with n nodes = $O(\log_2 n)$

  Thus, time complexity of search = $O(\log_2 n)$

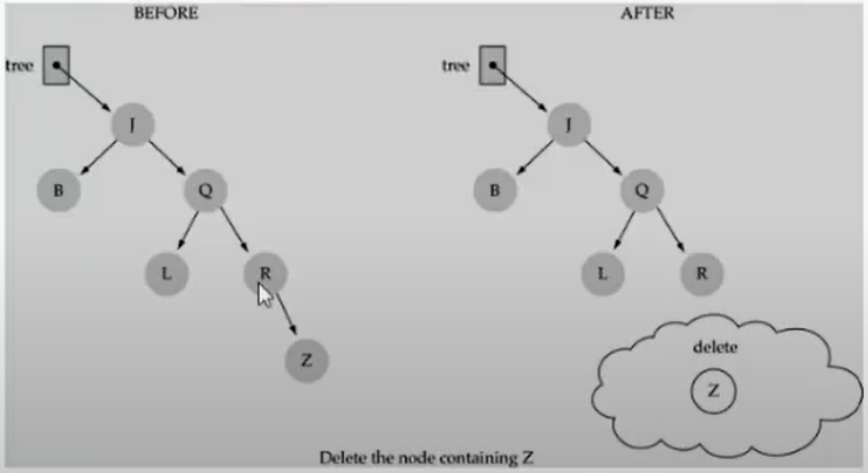
Alka Jindal

wab-jvws-tfr (2021-10-30 at 01:45 GMT-7)

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Deletion in BST: With no Children

- Simply delete the node



A

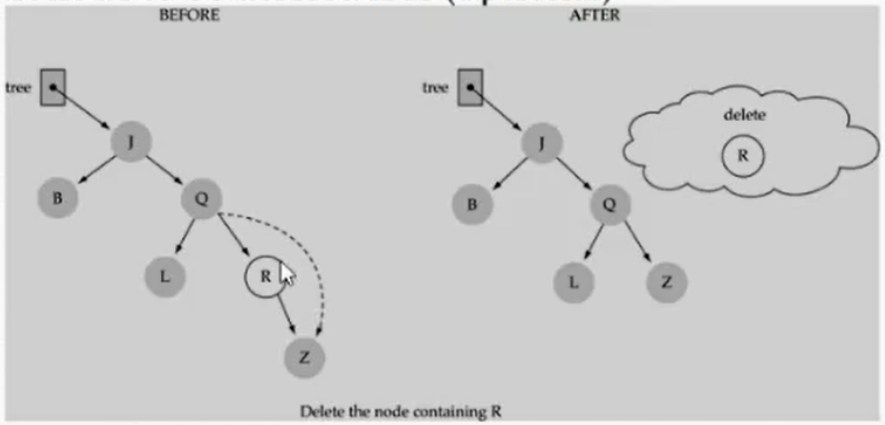
Alka Jindal

27:15 / 44:07

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Deletion in BST: With One Child

- Make parent of node to be deleted to point its child node
- Set node to be deleted free (optional)



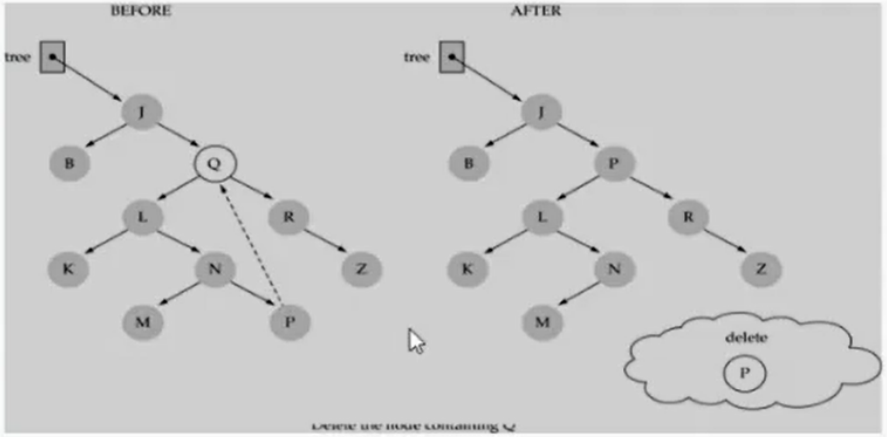
A

Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Deletion in BST : With Two Child

- Find predecessor (i.e. rightmost node in the left sub-tree) and replace data
- Delete predecessor node



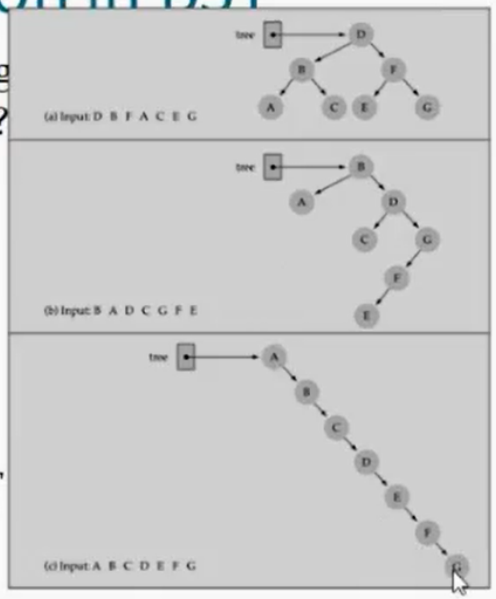
A

Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Order of Insertion in BST

- Does the order of inserting elements into tree matters?
- Yes, certain orders might produce very unbalanced trees!
- Unbalanced trees are not desirable because search time increases!
- Solution are balanced BST like AVL tree.



A

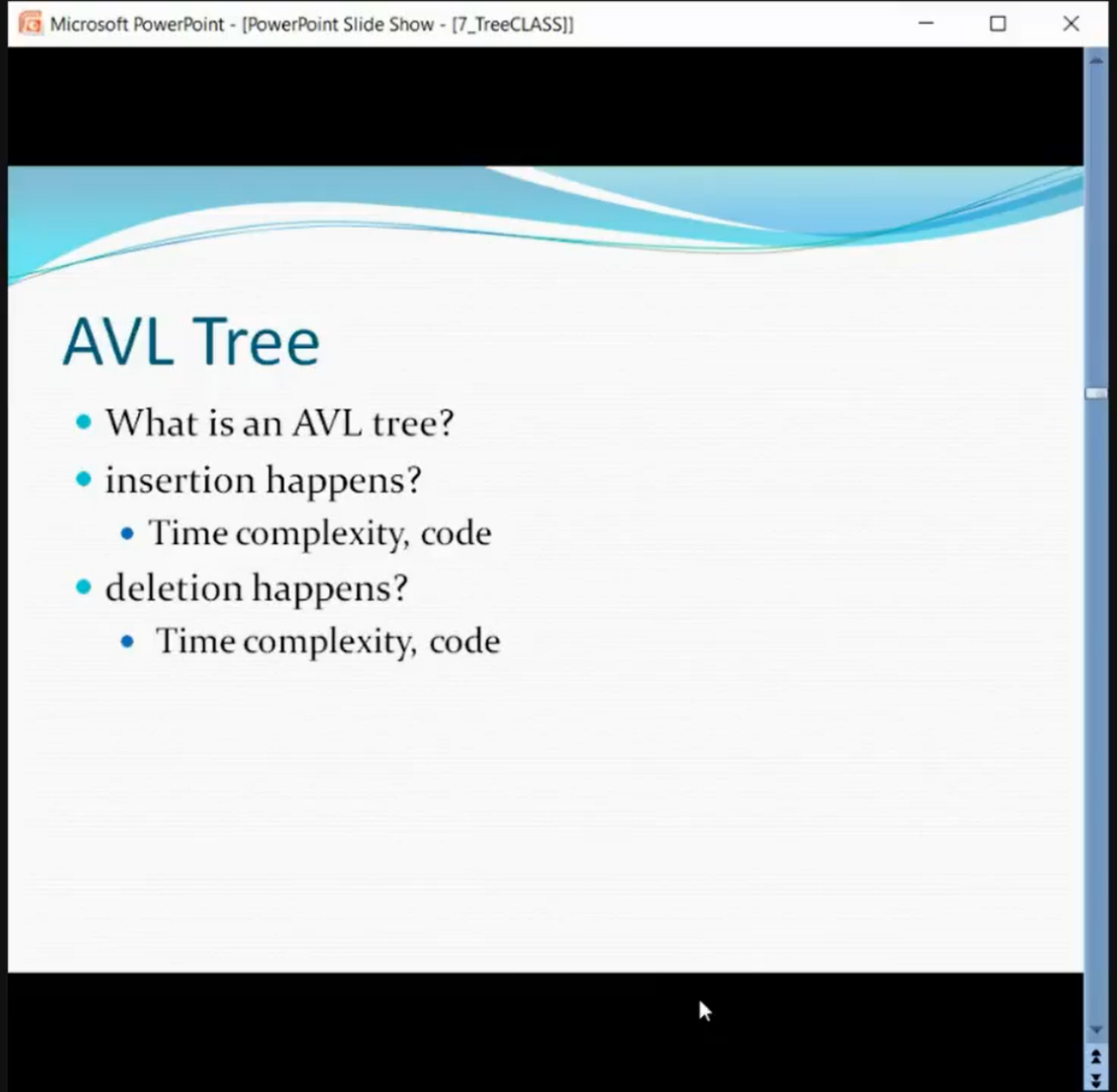
Alka Jindal

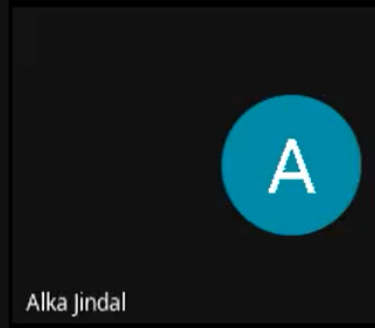
Balanced BST

- Most of the applications (eg. Search, insert, delete, minimum, maximum etc.) are dependent on height of BST
- Order of insertion of element in BST could generate unbalanced BST
- Therefore, in worst case time complexity can be **O(n)**

- If we can ensure the height of BST is O(log n) always, we can guarantee other operations will also have time complexity of O(log n)
- Balanced BST have height O(log n) always
- Eg. AVL tree, Red-black tree, 2-3 tree etc

A
Alka Jindal


Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]
AVL Tree
What is an AVL tree?
insertion happens?
Time complexity, code
deletion happens?
Time complexity, code

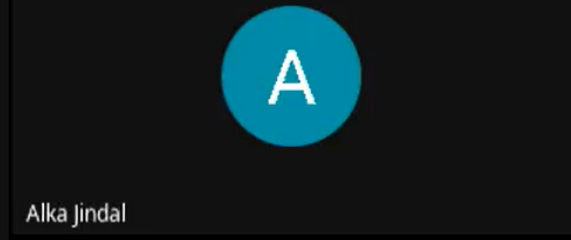

A
Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

AVL Tree

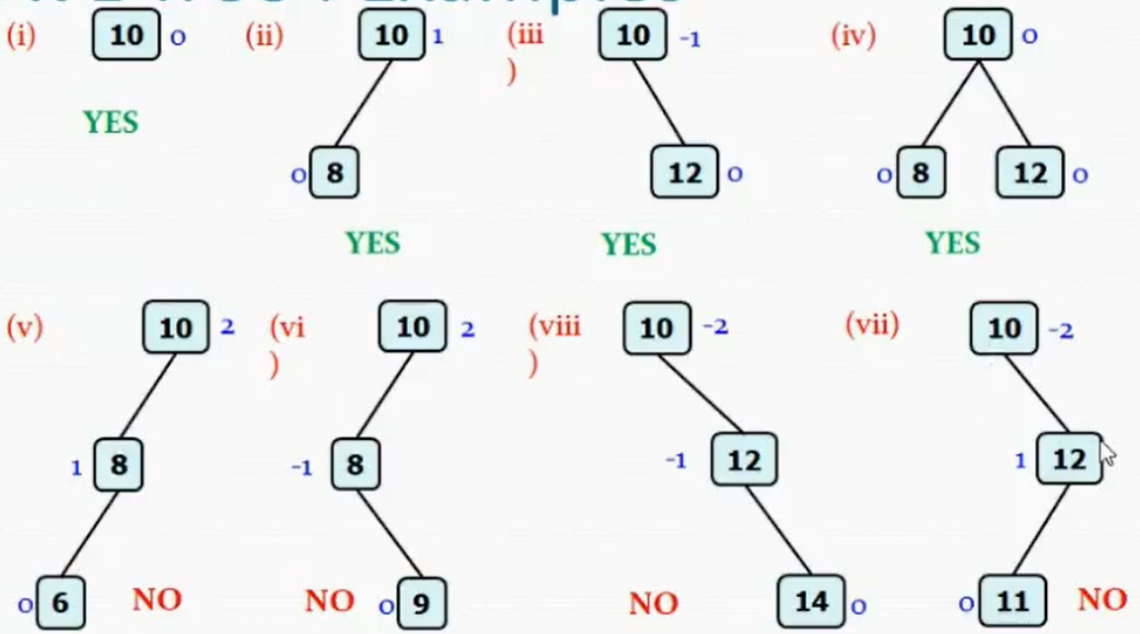
- A balanced BST named after scientist invented
- Maintains height close to minimum i.e. O(log n)
- After each insertion, check whether tree is balanced
- If not balanced, fix imbalance by rotation(s) to bring back to balance
- Balance factor of any node is difference between height of left and right sub-tree
- Tree is balanced if

-1 < balance factor of a node < 1



Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

AVL Tree : Examples



Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Press Esc to exit full screen

Insertion in AVL

Steps to insert in AVL:

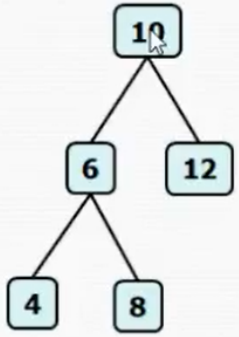
1. Insert in BST (i.e. find proper location and create node)
2. Starting from new node inserted, check upwards for the first node which is imbalance
3. Re-balance tree by performing rotation(s) based on the type of node arrangement causing imbalance
    1. Left-Left case
    2. Right-Right case
    3. Left-Right case
    4. Right-Left case

Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Left-Left Case

- Consider the AVL tree
- Insert 4 in the tree
- Travel upwards from new node to find first imbalance node i.e. 8
- If new node is left-left to imbalance node, its left-left case
- Do a right rotate at 8 to balance tree
- Now the tree is again balanced



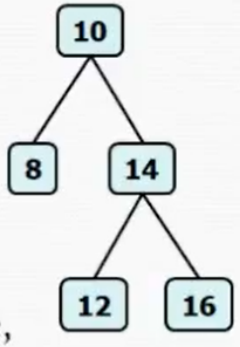
A

Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Right-Right Case

- Consider the AVL tree
- Insert **16** in the tree
- Travel upwards from new node to find first imbalance node i.e. **12**
- If new node is right-right to imbalance node, its right-right case
- Do a right rotate at **12** to balance tree
- Now the tree is again balanced



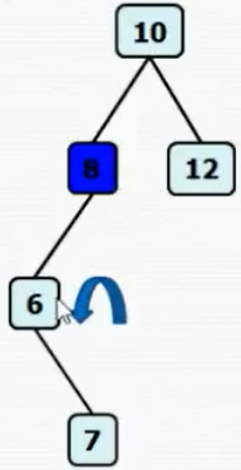
A

Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Left-Right Case

- Consider the AVL tree
- Insert 7 in the tree
- Travel upwards from new node to find first imbalance node i.e. 8
- If new node is left-right to imbalance node, its left-right case
- Two steps to balance:
  - Do a left rotate at **8->left (i.e. 6)**



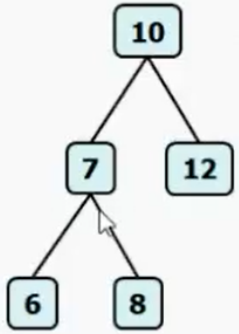
A

Alka Jindal

Microsoft PowerPoint - [PowerPoint Slide Show - [7_TreeCLASS]]

Left-Right Case

- Consider the AVL tree
- Insert 7 in the tree
- Travel upwards from new node to find first imbalance node i.e. 8
- If new node is left-right to imbalance node, its left-right case
- Two steps to balance:
  - Do a left rotate at **8->left (i.e. 6)**
  - Do a right rotate at **8**



A

Alka Jindal

Right-Left Case

- Consider the AVL tree
- Insert 13 in the tree
- Travel upwards from new node to find first imbalance node i.e. 12
- If new node is right-right to imbalance node, its right-right case
- Two steps to balance:
  - Do a rotate right at 12->right(i.e. 14)