



# DESIGN AND ANALYSIS OF ALGORITHMS

---

**Surabhi Narayan**

Department of Computer Science & Engineering

# DESIGN AND ANALYSIS OF ALGORITHMS

---

## 2-3 Trees

**Surabhi Narayan**

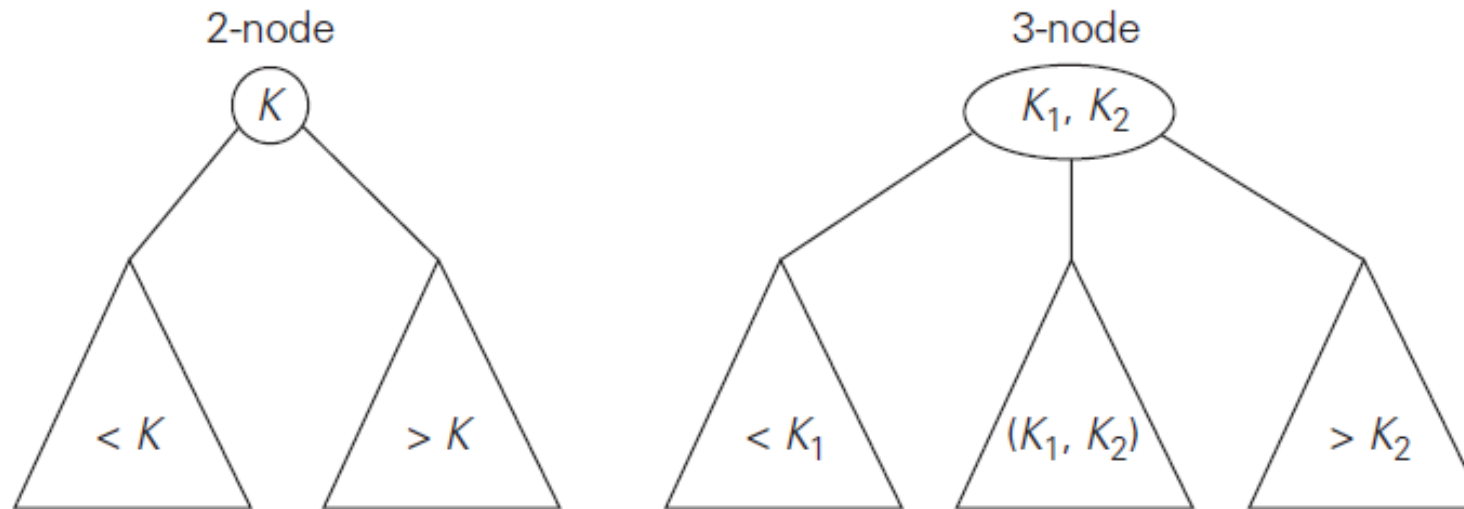
Department of Computer Science & Engineering

*2-3 tree* is a tree that can have nodes of two kinds: 2-nodes and 3-nodes.

- A *2-node* contains a single key  $K$  and has two children:  
the left child serves as the root of a subtree whose keys are less than  $K$ , and the right child serves as the root of a subtree whose keys are greater than  $K$ .
- A *3-node* contains two ordered keys  $K1$  and  $K2$  ( $K1 < K2$ ) and has three children.
- The leftmost child serves as the root of a subtree with keys less than  $K1$ , the middle child serves as the root of a subtree with keys between  $K1$  and  $K2$ , and the rightmost child serves as the root of a subtree with keys greater than  $K2$ .
- The last requirement of the 2-3 tree is that all its leaves must be on the same level. In other words, a 2-3 tree is always perfectly height-balanced: the length of a path from the root to a leaf is the same for every leaf.

Here are the properties of a 2-3 tree:

- each node has either one value or two value
- a node with one value is either a leaf node or has exactly two children (non-null). Values in left subtree  $<$  value in node  $<$  values in right subtree
- a node with two values is either a leaf node or has exactly three children (non-null). Values in left subtree  $<$  first value in node  $<$  values in middle subtree  $<$  second value in node  $<$  value in right subtree.
- all leaf nodes are at the same level of the tree



**FIGURE 6.7** Two kinds of nodes of a 2-3 tree.

**Search:** To search a key **K** in given 2-3 tree **T**, we follow the following procedure:

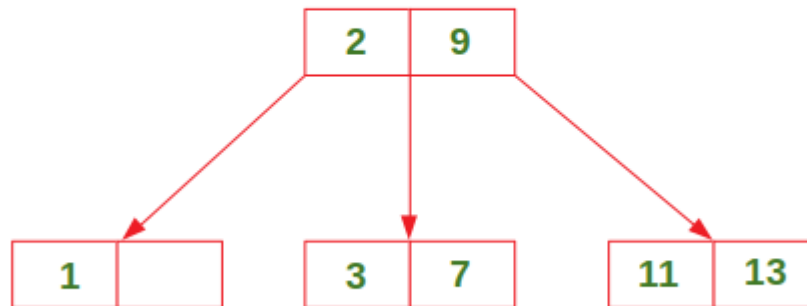
Base cases:

1. If **T** is empty, return False (key cannot be found in the tree).
2. If current node contains data value which is equal to **K**, return True.
3. If we reach the leaf-node and it doesn't contain the required key value **K**, return False.

Recursive Calls:

1. If  $\mathbf{K} < \text{currentNode.leftVal}$ , we explore the left subtree of the current node.
2. Else if  $\text{currentNode.leftVal} < \mathbf{K} < \text{currentNode.rightVal}$ , we explore the middle subtree of the current node.
3. Else if  $\mathbf{K} > \text{currentNode.rightVal}$ , we explore the right subtree of the current node.

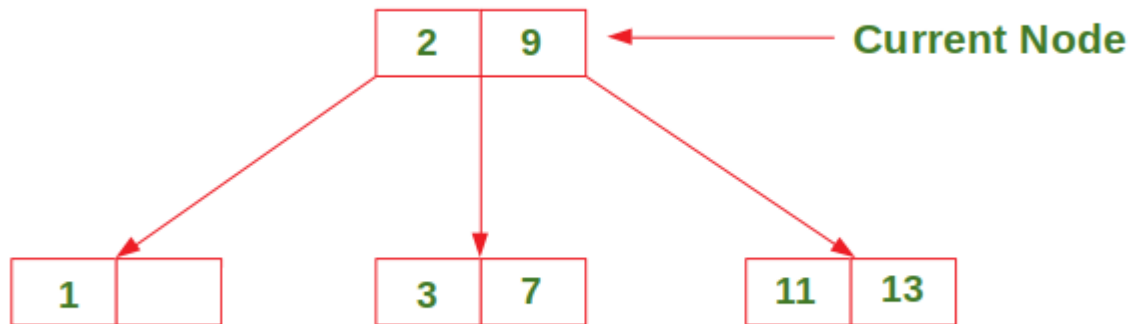
Search 5 in the following 2-3 Tree:



# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

Search 5

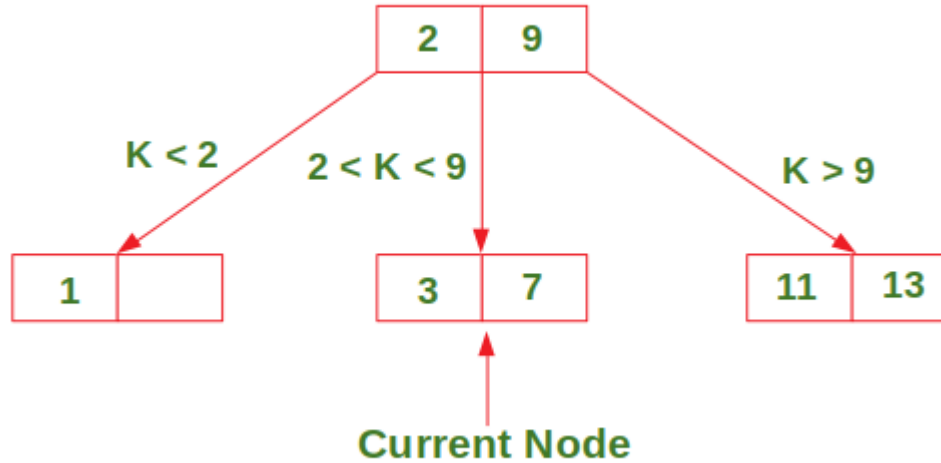




# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

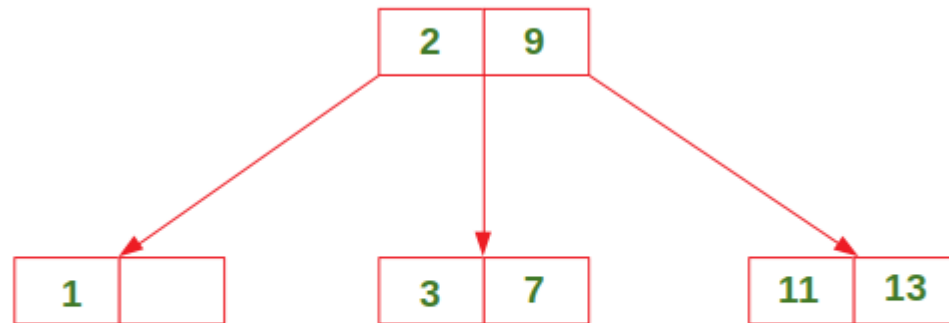
### Search 5



# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

Search 5



5 Not Found. Return False

### Insertion

The insertion algorithm into a two-three tree is quite different from the insertion algorithm into a binary search tree. In a two-three tree, the algorithm will be as follows:

- If the tree is empty, create a node and put value into the node
- Otherwise find the leaf node where the value belongs.
- If the leaf node has only one value, put the new value into the node
- If the leaf node has more than two values, split the node and promote the median of the three values to parent.
- If the parent then has three values, continue to split and promote, forming a new root node if necessary

# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

---

Insert 50

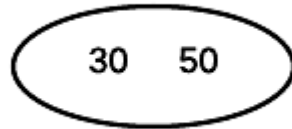


# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

---

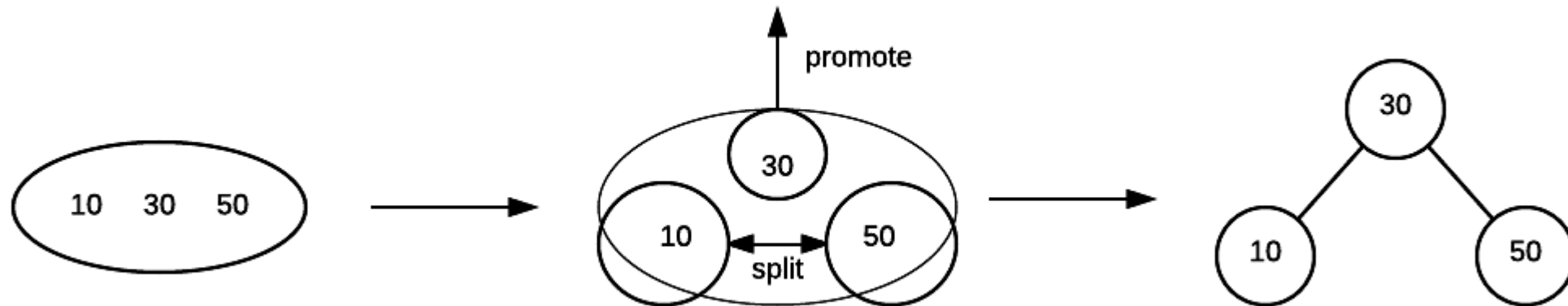
Insert 30



# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

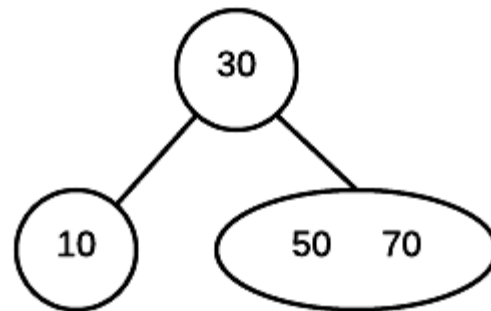
Insert 10



# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

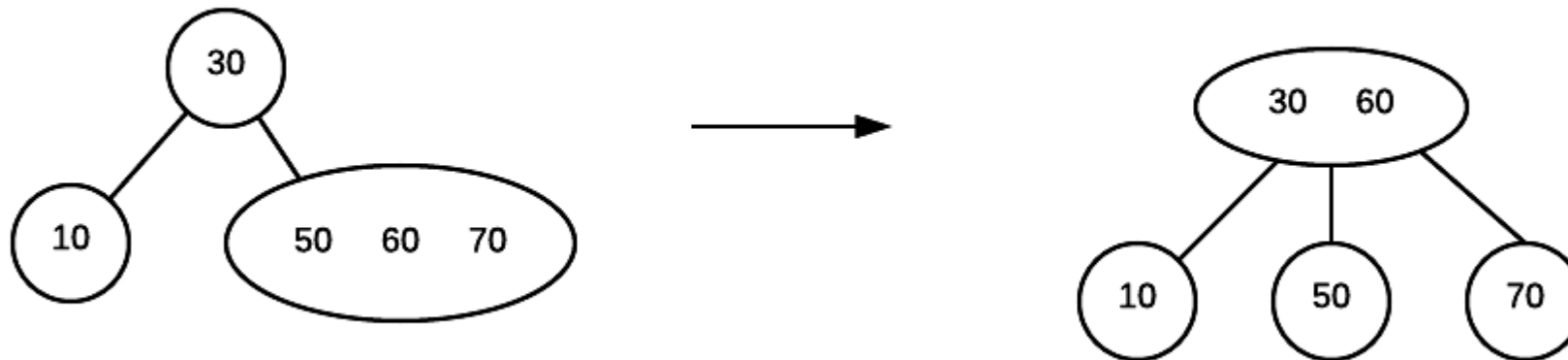
Insert 70



# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree

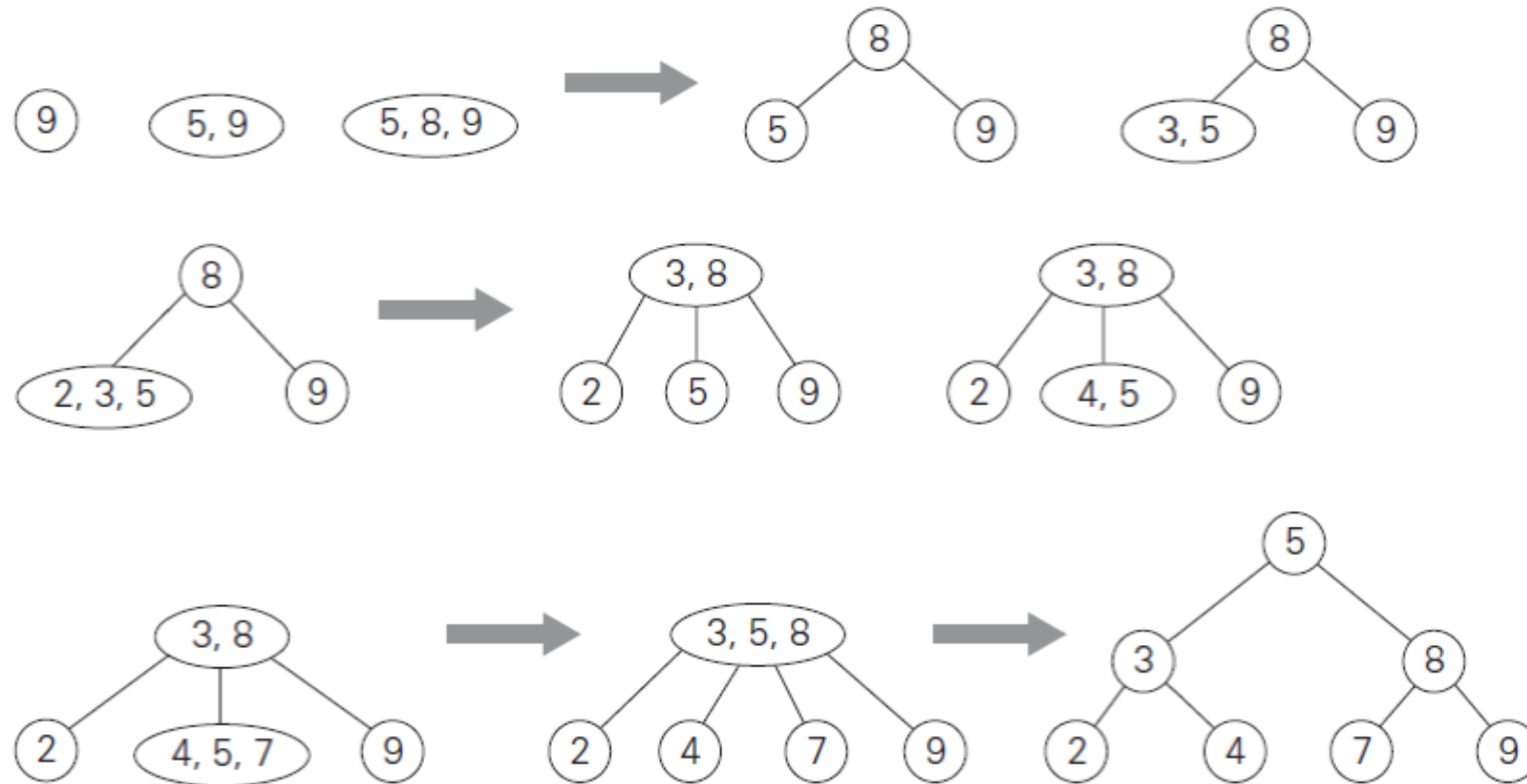
Insert 60





# DESIGN AND ANALYSIS OF ALGORITHMS

## 2-3 Tree



**FIGURE 6.8** Construction of a 2-3 tree for the list 9, 5, 8, 3, 2, 4, 7.

### Time Complexity

The property of being perfectly balanced, enables the **2-3 Tree** operations of insert, delete and search to have a time complexity of  $O(\log (n))$ .



**THANK YOU**

---

**Surabhi Narayan**

Department of Computer Science & Engineering

**[surabhinarayan@pes.edu](mailto:surabhinarayan@pes.edu)**