2–3 Trees

Nimrat Kaur (A125012)
Srijita Verma (A125023)

International Institute of Information Technology, Bhubaneswar

December 3, 2025

Outline

Introduction

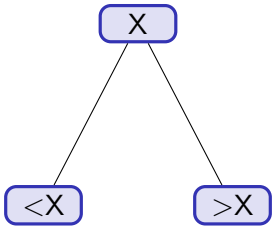
- A 2–3 Tree is a type of self-balancing search tree used in computer science for efficient data storage and retrieval
- In a 2–3 tree every node has:
  - Either 1 key and 2 children (2-node), or
  - 2 keys and 3 children (3-node)
- Leaf nodes are always at the **same level**, ensuring perfect balance.
- They form the conceptual foundation for B-Trees, B+ Trees, and 2-3-4 Trees used in databases, file systems, and indexing structures.
- They guarantee worst-case logarithmic height, making them useful in systems requiring deterministic performance.

Properties of 2–3 Trees

- Every internal node is either:
  - **2-node**: 1 key, 2 children
  - **3-node**: 2 keys, 3 children
- Keys are always stored in sorted order.
- Each node contains one or two keys, including leaf nodes.
- Operations supported are - **Searching, Insertion and Deletion**.
- All leaves appear at the same level (strict height balance).
- The height of the tree is logarithmic in the number of keys.

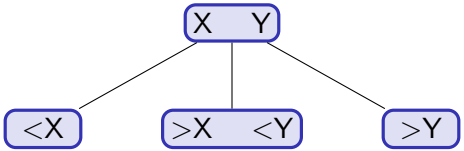
2-Node

- Contains one key.
- It has two children.



3-Node

- Contains two keys.
- It has three children.



2-3 Tree vs Binary Tree

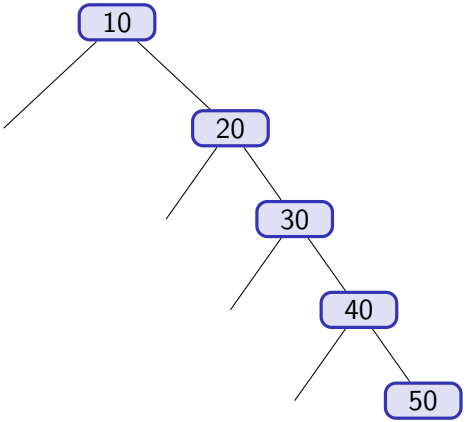
2-3 Tree

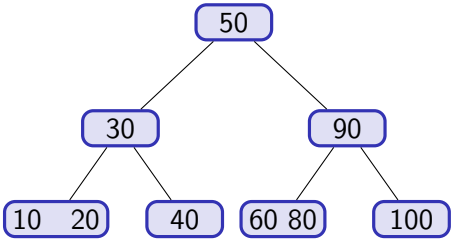
- Multiway tree (2 or 3 children).
- Always height-balanced.
- Operations take O(logn).
- Uses splits and merges to maintain balance.

Binary Tree

- Max 2 children.
- Not guaranteed balanced.
- Worst case O(n).
- Needs rotations (AVL, Red-Black) for balancing.

2-3 Tree vs Binary Tree





Height of 2–3 Tree

Deriving the Height Bound

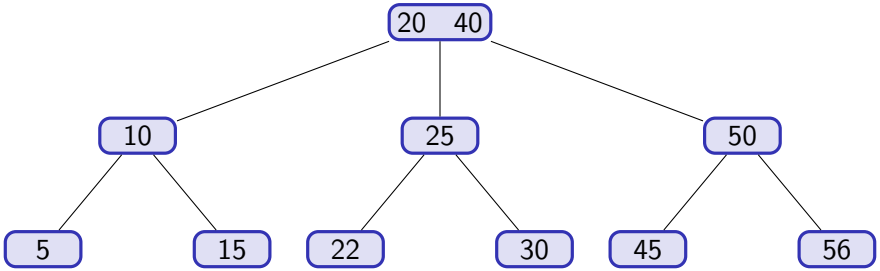
- Minimum branching (all 2-nodes): Leaves at height h= 2^h
- Maximum branching (all 3-nodes): Leaves at height h= 3^h
- Since the tree has n keys: $2^h \leq n \leq 3^h$
- Taking logs: $\log_3 n \leq h \leq \log_2 n$
- **Therefore:** h $= \Theta(\log n)$
- The Height of a 2–3 tree with n keys is: h $= O(\log n)$
- Because the tree always stays balanced.

Searching in a 2–3 Tree

- **At a 2-node:**
  - If X $=$ key $\rightarrow$ FOUND
  - If X $<$ key $\rightarrow$ go left
  - If X $>$ key $\rightarrow$ go right
- **At a 3-node with keys K1 nad K2:**
  - If X $=$ K1 or X $=$ K2 $\rightarrow$ FOUND
  - If X $<$ key $\rightarrow$ go left
  - If K1 $<$ X $<$ K2 $\rightarrow$ go middle
  - If X $>$ K2 $\rightarrow$ go right
- **Time Complexity**
  - Searching: $O(\log n)$

Searching Example

- Search 30
- Found at the right child of 25.



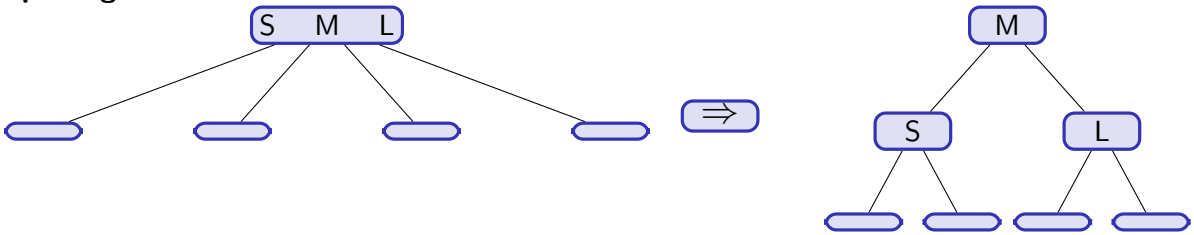
Insertion Algorithm

- To insert an item I into a 2-3 tree, first locate the leaf at which the search for I would terminate.
- Insert the new item I into the leaf.
- If the leaf now contains only two items, you are done. If the leaf contains three items, you must split it.
- **Splitting a leaf**

Insertion Algorithm

Insertion Algorithm

Splitting a root

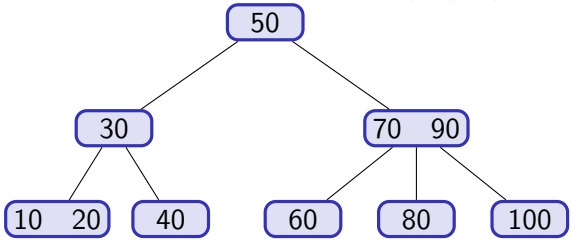


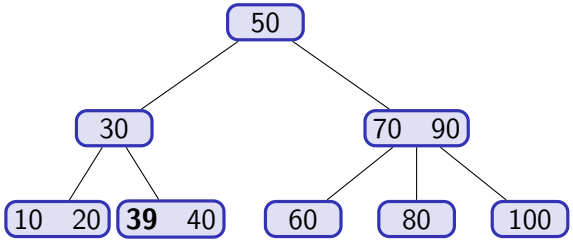
Psuedo Code

- *Search for the correct leaf where the key belongs.*
- *Insert the key into the leaf in sorted order.*
- *If the leaf now has 3 keys:*
  - Split it into two nodes.
  - Promote the middle key to the parent.
- *If the parent overflows,keep splitting upward.*
- *If the root splits, create a new root.*

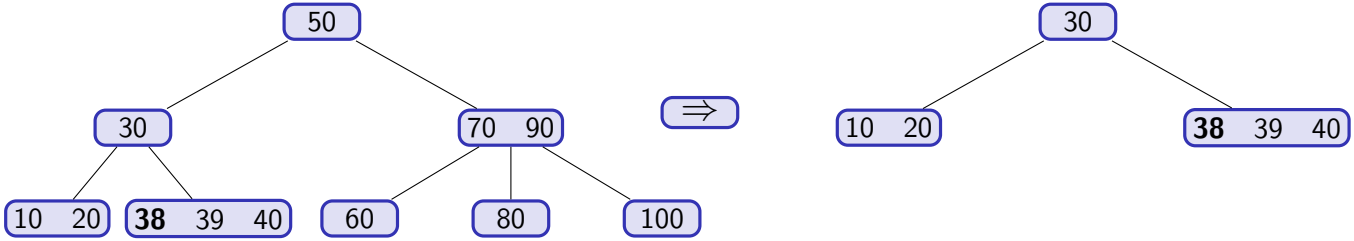
Insertion Example

Our Goal is to add the nodes- **39,38,37,36** in that order!

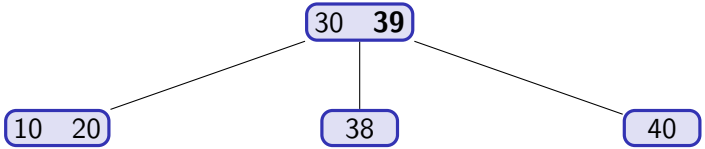


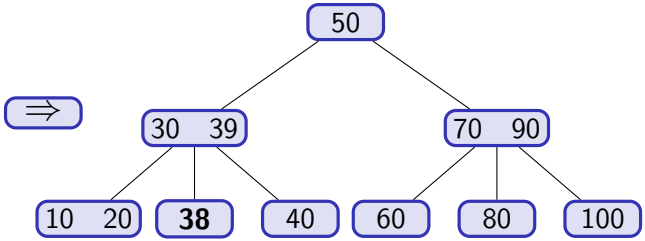


Insertion Example

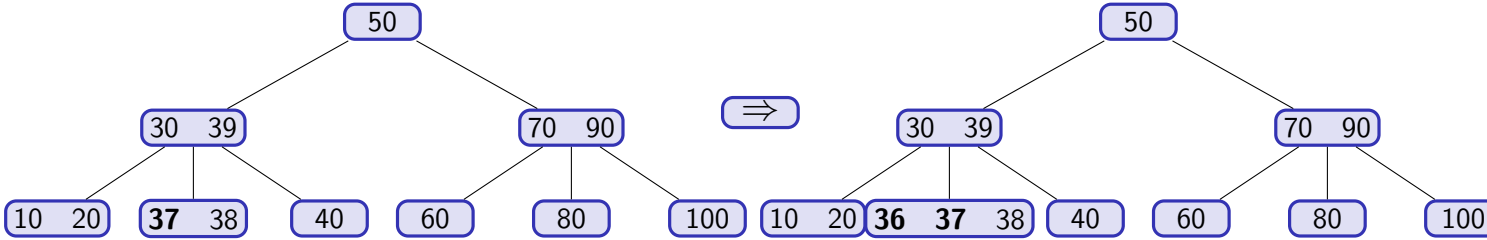


Insertion Example

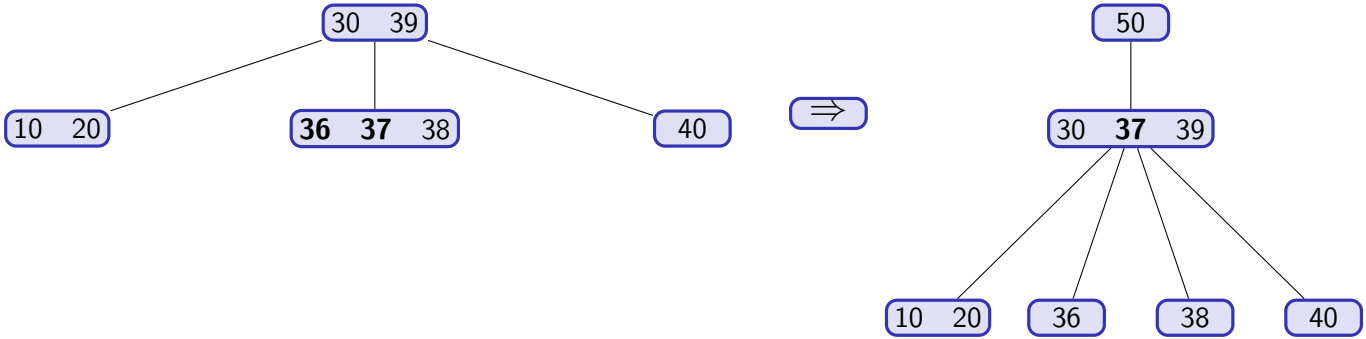




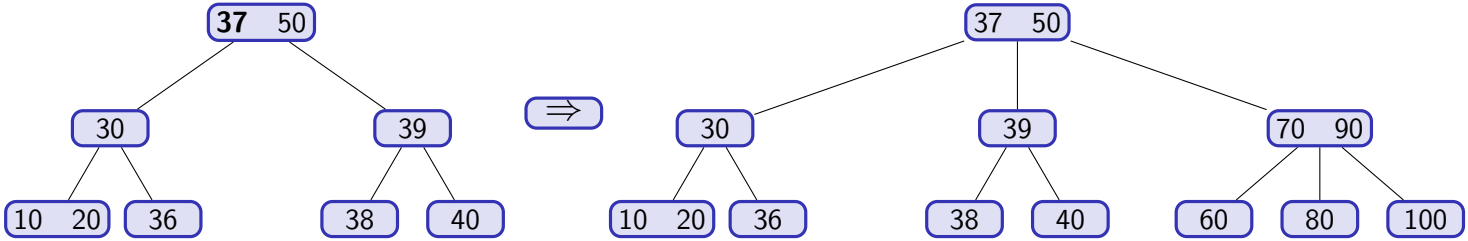
Insertion Example



Insertion Example



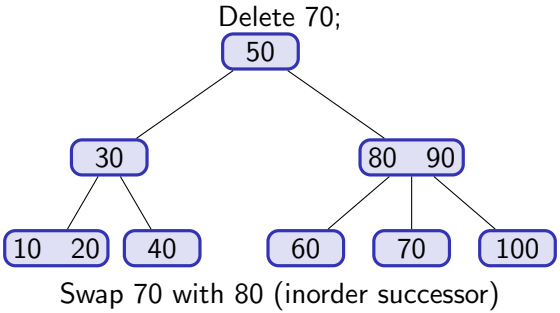
Insertion Example



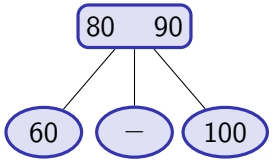
Deletion in 2–3 Tree

- If the key is in an internal node:
  - Replace it with its inorder predecessor or successor.
- Delete the key from the leaf level.
- If a node underflows (node becomes empty):
  - Borrow a key from a sibling, or
  - Merge with a sibling and push the parent key down.
- Underflow may propagate upward until the root is reached.
- If the root becomes empty, its single child becomes the new root.
- Time Complexity : $O(\log n)$

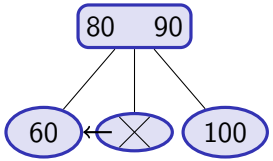
Deletion Example



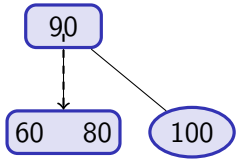
Deletion Example



(a) Delete value from leaf



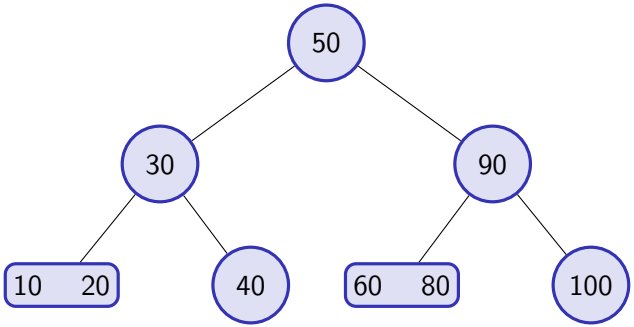
(b) Merge nodes by deleting empty leaf



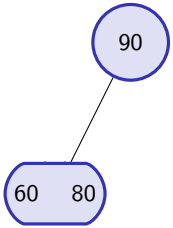
(c) Move 80 downward

Deletion Example

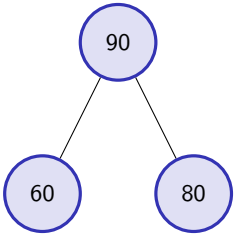
Now delete 100



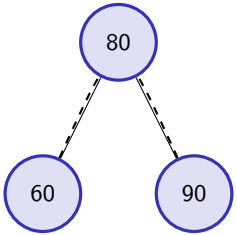
Deletion Example



(a)
Delete value from leaf

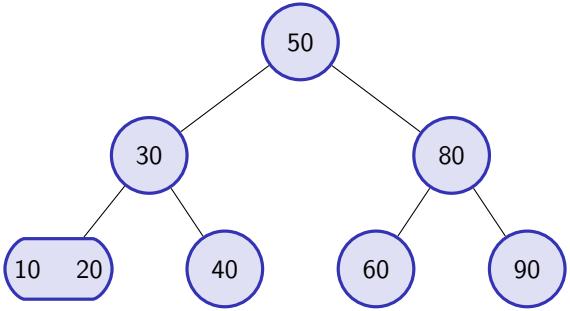


(b)
Doesn't work



(c)
Redistribute

Deletion Example



Time Complexity

Operation	Worst Case	Reason
Search	$O(\log n)$	Height is perfectly balanced.
Insert	$O(\log n)$	Only one root-to-leaf path is modified; splits propagate upward.
Delete	$O(\log n)$	Borrow/merge operations occur only along one path.

Space Complexity

Metric	Complexity	Reason
Total Space	$O(n)$	Each key stored exactly once; limited extra pointers.
Node Space	$O(1)$	Each node stores at most 2 keys and 3 pointers.

Why 2–3 Trees Aren't Used Directly

- Nodes have a fixed branching factor (2 or 3), which is too small for disk-based storage.
- B-Trees generalize the idea by storing many keys per node, sized to match disk blocks or cache lines.
- The same balancing concepts—split, merge, promote—scale naturally to large nodes.
- Larger nodes reduce:
  - Disk seeks
  - Cache misses
  - Tree height
- Because of this, real systems use B-Trees and B^+ Trees, not strict 2–3 Trees, even though the underlying principles are the same.

Conclusion

- 2–3 Trees maintain strict height balance, ensuring search, insert, and delete always run in O(log n) time.
- Splitting and merging during insertion and deletion preserve uniform depth across all leaves.
- They provide a clean theoretical model for understanding multiway search trees.
- Most importantly, they form the foundation for B-Trees and B+ Trees, which are widely used in databases, file systems, and indexing systems today.

References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.
- Mark Allen Weiss. *Data Structures and Algorithm Analysis in C++*, 4th ed. Pearson, 2013.
- Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser. *Data Structures and Algorithms in Java*, 6th ed. Wiley, 2014.
- Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1998.
- Robert Sedgewick and Kevin Wayne. *Algorithms*, 4th ed. Addison-Wesley, 2011.

Thank You :)