B-Tree

Introduction

- A **B-Tree** is a **self-balancing search tree** used in databases and file systems.
- It generalizes the idea of a **Binary Search Tree (BST)** but allows a node to have **multiple keys and multiple children**.
- It keeps data **sorted** and allows **search, insertion, and deletion in O(log n)** time.

Unlike BST/AVL/Red-Black Trees, a B-Tree is designed to work well with **disk storage / large data blocks**.

Properties of B-Tree

Let the **order of a B-Tree = m** (max number of children a node can have).

1. Each node can have at most **m children**.
2. Each node (except root) must have at least ⌈**m/2**⌉ **children**.
3. Each node can contain at most **m-1 keys**.
4. Each node (except root) must contain at least ⌈**m/2**⌉ **- 1 keys**.
5. Keys inside a node are stored in **sorted order**.
6. **All leaves appear at the same level** (tree is balanced).
7. **Search** works like in BST:
    - If key < current key → go to left child.

  - If key > current key → go to right child.

Example (B-Tree of Order 3 → 2–3 Tree)

Order = 3 means:

- Each node can have at most **2 keys and 3 children**.

Example Tree after inserting `[10, 20, 5, 6, 12, 30, 7, 17]`:

```
         [10, 20]
         /    |    \
    [5, 6, 7] [12, 17] [30]
```

Operations on B-Tree

(i) Search

- Start from root.
- Traverse keys in a node:
  - If key found → Success.
  - If key < current key → Move to left child.
  - If key > largest key → Move to rightmost child.
- Continue until found or leaf reached.
- **Time Complexity = O(log n)**.

(ii) Insertion

1. Always insert new key in a **leaf node**.
2. If the leaf has space (< m–1 keys), insert directly.
3. If the leaf is full:
    - **Split the node** around the middle key.
    - Promote the middle key to the parent.
    - Repeat split if parent also becomes full.

(iii) Deletion

More complex than insertion.
Cases:

1. If the key is in a **leaf node** → simply delete.
2. If the key is in an **internal node**:
    - Replace it with **predecessor** or **successor** key (from child subtree), then delete from child.
3. If deletion causes underflow (less than ⌈m/2⌉ – 1 keys):
    - **Borrow** a key from sibling (left or right).
    - If sibling also has minimum keys → **Merge** with sibling.

Applications of B-Tree

- **Databases & File Systems**
    - Used in **MySQL, Oracle, PostgreSQL, MongoDB**.

- **Indexing** large data (secondary storage).
- **Disk-based storage**: minimizes disk reads.
- **Search Engines** for maintaining large indexes.

Complexity

Operation	Time Complexity
Search	O(log n)
Insertion	O(log n)
Deletion	O(log n)
Traversal	O(n)

Much faster for **disk access** than AVL/Red-Black trees because of fewer height levels.