

## What is a B+ Tree?

A **B+ tree** is a self-balancing tree data structure that maintains sorted data and allows for efficient insertion, deletion, and search operations. Key properties:

1. **Each internal node can have up to  $n$  children** (order  $n$ ).
  2. **Each internal node (except the root) must have at least  $\lceil n/2 \rceil$  children.**
  3. **Leaf nodes store actual data and are linked together.**
  4. **Internal nodes store only keys and pointers to children.**
- 

## Example 1: B+ Tree of Order 2 ( $n=2$ )

We will insert: [10, 20, 5, 6, 12]

- **Step 1:** Insert 10 → No splitting needed.
- **Step 2:** Insert 20 → No splitting needed.
- **Step 3:** Insert 5 → The node overflows (it has 3 keys but max is 2), so we split.
  - **Split the node into two:**
    - Left leaf: [5]
    - Right leaf: [10, 20]
    - Promote 10 to the parent.
- **Step 4:** Insert 6 → Fits in the left leaf [5, 6], no split needed.
- **Step 5:** Insert 12 → Goes into the right leaf [10, 12, 20], overflows, so split again.
  - **Split:**

- Left leaf: [10, 12]
- Right leaf: [20]
- Promote 12 to parent.

**Final B+ Tree (n=2):**

```

    [10, 12]
   /  |  \
[5, 6] [10, 12] [20]

```

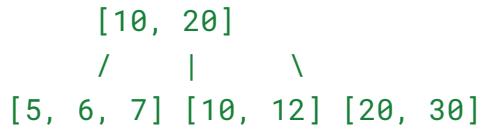
---

## Example 2: B+ Tree of Order 3 (n=3)

Inserting: [10, 20, 5, 6, 12, 30, 7]

- **Step 1-4:** Insert 10, 20, 5, 6 → No split needed yet.
- **Step 5:** Insert 12 → Leaf node exceeds 3 keys → **Split the node.**
  - Left leaf: [5, 6]
  - Right leaf: [10, 12, 20]
  - Promote 10 to parent.
- **Step 6:** Insert 30 → Fits in right leaf [10, 12, 20, 30], overflows → **Split again.**
  - Left leaf: [10, 12]
  - Right leaf: [20, 30]
  - Promote 20 to parent.
- **Step 7:** Insert 7 → Goes into [5, 6, 7], no split.

**Final B+ Tree (n=3):**

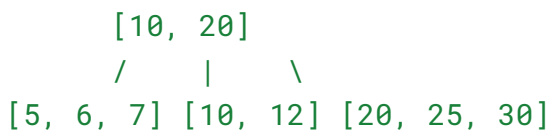


## Example 3: B+ Tree of Order 4 (n=4)

Inserting: [10, 20, 5, 6, 12, 30, 7, 25]

- The splitting process follows the same logic, but nodes can hold more keys before splitting.

**Final B+ Tree (n=4):**



Since **n=4**, splits happen less frequently, leading to a shallower tree.

---

## Steps for a Computer to Insert into a B+ Tree

A computer program would follow these **steps**:

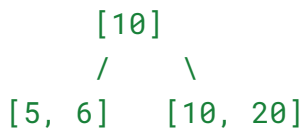
1. **Find the correct leaf node** where the key should go (similar to a binary search).
2. **Insert the key** in sorted order within the leaf.
3. **Check if the leaf node overflows** (has more than  $n-1$  keys).
  - **If yes**, split it into two and promote the middle key.
4. **Propagate the split up to parent nodes** if needed.
5. **If the root splits**, create a new root.

## Example 4: B+ Tree of Order 3 (n=3) - Root Split

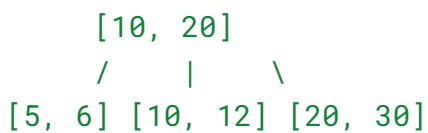
Inserting: [10, 20, 5, 6, 12, 30, 7, 25, 15]

### Step-by-step insertion

1. **Insert 10, 20, 5** → First leaf node: [5, 10, 20]
2. **Insert 6** → [5, 6, 10, 20] (Overflows, split occurs)
  - Left leaf: [5, 6]
  - Right leaf: [10, 20]
  - Promote 10 to parent → **Tree gains a root**



3. **Insert 12** → Goes into [10, 20], making [10, 12, 20]
4. **Insert 30** → [10, 12, 20, 30] (Overflows, split occurs)
  - Left leaf: [10, 12]
  - Right leaf: [20, 30]
  - Promote 20 to parent



5. **Insert 7** → Goes into [5, 6], making [5, 6, 7], no split needed.
6. **Insert 25** → Goes into [20, 30], making [20, 25, 30], no split needed.
7. **Insert 15** → Goes into [10, 12], making [10, 12, 15], no split needed.

### Final Tree (Order 3)



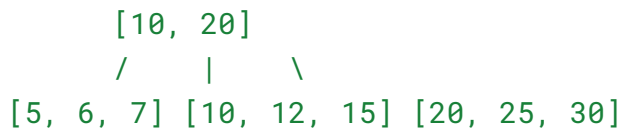
✓ Levels increased once when root split.

---

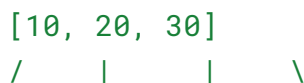
## Example 5: B+ Tree of Order 3 (n=3) - 2nd Level Split

Inserting: [10, 20, 5, 6, 12, 30, 7, 25, 15, 40, 50]

Starting from the previous tree:



8. **Insert 40** → Goes into [20, 25, 30], making [20, 25, 30, 40] (Overflow, split)
  - Left leaf: [20, 25]
  - Right leaf: [30, 40]
  - Promote 30 to parent



[5, 6, 7] [10, 12, 15] [20, 25] [30, 40]

9. **Insert 50** → Goes into [30, 40], making [30, 40, 50], no split needed.

### Final Tree (Order 3)

```
      [10, 20, 30]
     /   |   |   \
[5, 6, 7] [10, 12, 15] [20, 25] [30, 40, 50]
```

✓ Levels increased once when root split.

---

## Example 6: B+ Tree of Order 3 (n=3) - Root Splits Again (Third Level)

Inserting: [10, 20, 5, 6, 12, 30, 7, 25, 15, 40, 50, 60, 70]

Starting from:

```
      [10, 20, 30]
     /   |   |   \
[5, 6, 7] [10, 12, 15] [20, 25] [30, 40, 50]
```

10. **Insert 60** → Goes into [30, 40, 50], making [30, 40, 50, 60] (Overflow, split)

- Left leaf: [30, 40]
- Right leaf: [50, 60]
- Promote 50 to parent

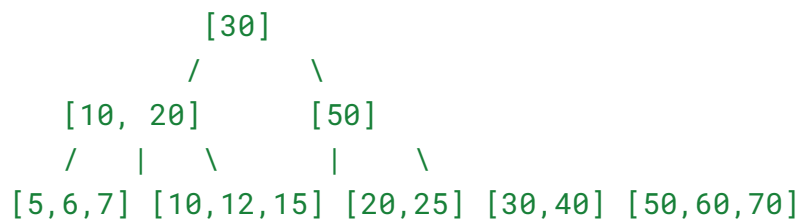
```
      [10, 20, 30, 50]
     /   |   |   |   \
```

[5,6,7] [10,12,15] [20,25] [30,40] [50,60]

11. **Insert 70** → Goes into [50, 60], making [50, 60, 70], no split needed.

12. Now the root [10, 20, 30, 50] has 4 keys, exceeding 3, so it splits!

- Left internal node: [10, 20]
- Right internal node: [50]
- Promote 30 to **NEW ROOT**



✓ Tree increased to three levels.

---

## Summary

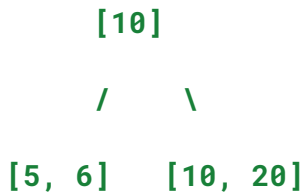
- **Root splits once** → Tree increases to 2 levels.
- **Internal node splits, forcing root to split again** → Tree increases to 3 levels.
- **Each time root splits, height increases.**

## Example 4: B+ Tree of Order 3 (n=3) - Root Split

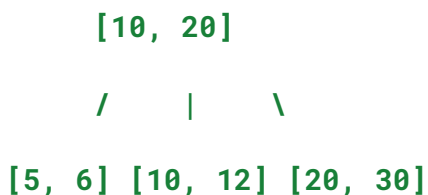
Inserting: [10, 20, 5, 6, 12, 30, 7, 25, 15]

## Step-by-step insertion

1. Insert 10, 20, 5 → First leaf node: [5, 10, 20]
2. Insert 6 → [5, 6, 10, 20] (Overflows, split occurs)
  - Left leaf: [5, 6]
  - Right leaf: [10, 20]
  - Promote 10 to parent → Tree gains a root



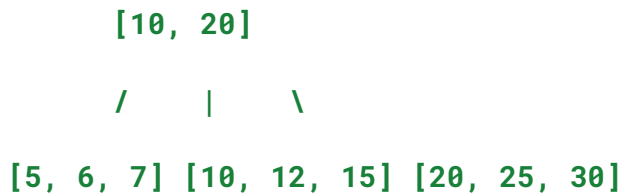
3. Insert 12 → Goes into [10, 20], making [10, 12, 20]
4. Insert 30 → [10, 12, 20, 30] (Overflows, split occurs)
  - Left leaf: [10, 12]
  - Right leaf: [20, 30]
  - Promote 20 to parent





5. Insert 7 → Goes into [5, 6], making [5, 6, 7], no split needed.
6. Insert 25 → Goes into [20, 30], making [20, 25, 30], no split needed.
7. Insert 15 → Goes into [10, 12], making [10, 12, 15], no split needed.

Final Tree (Order 3)



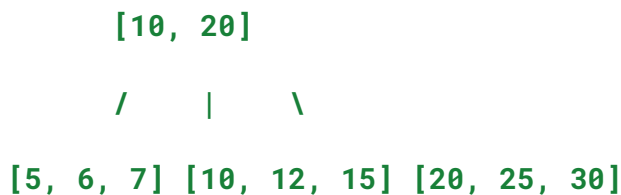
✓ Levels increased once when root split.

---

## Example 5: B+ Tree of Order 3 (n=3) - 2nd Level Split

Inserting: [10, 20, 5, 6, 12, 30, 7, 25, 15, 40, 50]

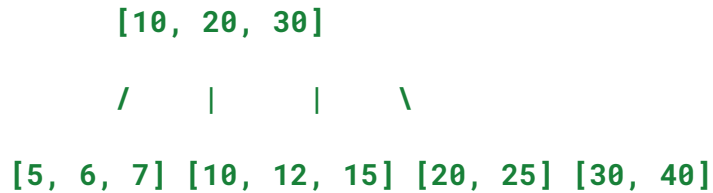
Starting from the previous tree:



8. Insert 40 → Goes into [20, 25, 30], making [20, 25, 30, 40] (Overflow, split)

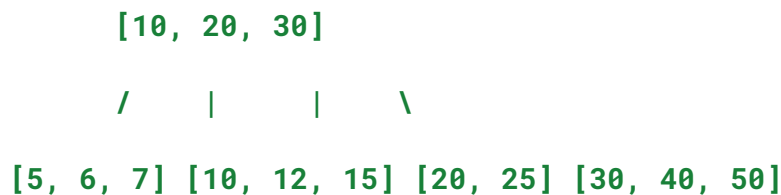
- Left leaf: [20, 25]

- Right leaf: [30, 40]
- Promote 30 to parent



9. Insert 50 → Goes into [30, 40], making [30, 40, 50], no split needed.

Final Tree (Order 3)



✓ Levels increased once when root split.

## Example 6: B+ Tree of Order 3 (n=3) - Root Splits Again (Third Level)

Inserting: [10, 20, 5, 6, 12, 30, 7, 25, 15, 40, 50, 60, 70]

Starting from:

```
[10, 20, 30]
```

/       |       |       \  
 [5, 6, 7] [10, 12, 15] [20, 25] [30, 40, 50]

10. Insert 60 → Goes into [30, 40, 50], making [30, 40, 50, 60] (Overflow, split)

- Left leaf: [30, 40]
- Right leaf: [50, 60]
- Promote 50 to parent

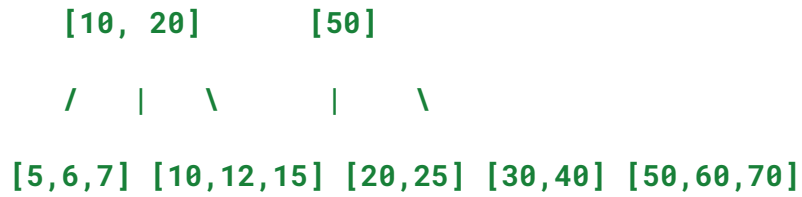
[10, 20, 30, 50]  
 /       |       |       |       \  
 [5,6,7] [10,12,15] [20,25] [30,40] [50,60]

11. Insert 70 → Goes into [50, 60], making [50, 60, 70], no split needed.

12. Now the root [10, 20, 30, 50] has 4 keys, exceeding 3, so it splits!

- Left internal node: [10, 20]
- Right internal node: [50]
- Promote 30 to NEW ROOT

[30]  
 /                \  
 [10, 20] [50, 60, 70]



✓ Tree increased to three levels.

---

## Summary

- Root splits once → Tree increases to 2 levels.
- Internal node splits, forcing root to split again → Tree increases to 3 levels.
- Each time root splits, height increases.