📌 Preorder & Inorder Se Binary Tree Banana

🔹 Problem Statement:

Ek preorder aur ek inorder traversal ka array diya hai, inko use karke ek binary tree construct karna hai.

🔥 Approach & Logic

1️⃣Root Identify Karna (Preorder Property)

- preorder ka **first element** hamesha **root** hota hai.
- Example:
- preorder = [3, 9, 20, 15, 7]
- inorder = [9, 3, 15, 20, 7]

Yahan 3 first element hai, toh yeh **root** hoga.

2️⃣Left aur Right Subtree Find Karna (Inorder Property)

- inorder me **root ke pehle wale elements** left subtree me honge.
- **Root ke baad wale elements** right subtree me honge.
- Example:
- inorder = [9, 3, 15, 20, 7]
-             ↑
-             (Root at index 1)
    - Left Subtree: [9]
    - Right Subtree: [15, 20, 7]

3️⃣Recursion Se Subtree Build Karna

- Left aur right subtree ke liye preorder aur inorder ke respective parts pass karke recursion se solve karenge.
- LeftCount aur RightCount ka use subtree ke size track karne ke liye karenge.

🔄 Dry Run Example 1

Input:

preorder = [3, 9, 20, 15, 7]

inorder = [9, 3, 15, 20, 7]

Step 1: Root Node

- preorder[0] = 3 → Root
- inorder me 3 ka index = 1
- Left Subtree: [9]
- Right Subtree: [15, 20, 7]

```
   3
  / \
 ?   ?
```

Step 2: Left Subtree

- preorder[1] = 9 → Left Child
- inorder me 9 ka index = 0
- Left aur Right Subtree dono empty hain.

```
   3
  / \
  9   ?
```

Step 3: Right Subtree

- preorder[2] = 20
- inorder me 20 ka index = 3
- Left Subtree: [15]
- Right Subtree: [7]

```
   3
  / \
  9   20
     /  \
   ?    ?
```

Step 4: Left Subtree of 20

- preorder[3] = 15
- inorder me 15 ka index = 2

```
   3
  / \
  9   20
     /  \
   15   ?
```

Step 5: Right Subtree of 20

- preorder[4] = 7
- inorder me 7 ka index = 4

```
   3
  / \
 9   20
    /  \
   15   7
```

🔄 Dry Run Example 2

Input:

preorder = [1, 2, 4, 5, 3, 6, 7]

inorder = [4, 2, 5, 1, 6, 3, 7]

Step 1: Root Node

- preorder[0] = 1 → Root
- inorder me 1 ka index = 3
- Left Subtree: [4, 2, 5]
- Right Subtree: [6, 3, 7]

```
   1
  / \
 ?   ?
```

Step 2: Left Subtree of 1

- preorder[1] = 2
- inorder me 2 ka index = 1
- Left: [4], Right: [5]

```
   1
  / \
 2   ?
/ \
4  5
```

Step 3: Right Subtree of 1

- preorder[4] = 3
- inorder me 3 ka index = 5
- Left: [6], Right: [7]

```
     1
    / \
   2   3
  / \  / \
 4   5 6   7
```

📝 Code Implementation

```
class Solution {
public:
    TreeNode* build(vector<int>& pre, int prelo, int prehi,
                    vector<int>& in, int inlo, int inhi) {
        if (inlo > inhi) return NULL;

        TreeNode* root = new TreeNode(pre[prelo]);

        if (prelo == prehi) return root;

        int i = inlo;
        while (i <= inhi) {
            if (in[i] == pre[prelo]) break;
            i++;
        }

        int LeftCount = i - inlo;
        int RightCount = inhi - i;

        root->left = build(pre, prelo + 1, prelo + LeftCount, in, inlo, i - 1);
        root->right = build(pre, prelo + LeftCount + 1, prehi, in, i + 1, inhi);

        return root;
    }

    TreeNode* buildTree(vector<int>& pre, vector<int>& in) {
        int n = pre.size();
        return build(pre, 0, n - 1, in, 0, n - 1);
    }
};
```

🛡️ Key Takeaways

1. **Preorder ka pehla element root hota hai.**
2. **Inorder se left aur right subtree identify hote hain.**
3. **Recursion se subtree build hoti hai.**
4. **Index ka use subtree ke sizes track karne ke liye hota hai.**
5. **Agar inorder ka lookup hashmap se kare toh complexity O(N) ho sakti hai.**

Ye ekdum simple aur samajhne layak Hinglish version hai! 🚀