1. Implement Insertion in an AVL Tree

Write a program to insert nodes into an AVL tree while maintaining its **balance factor.**
Perform necessary **rotations** (LL, RR, LR, RL) to keep it balanced.

Example:
Insert sequence: `10, 20, 30`
After inserting 30 → perform **Left Rotation** → Balanced Tree:

2. Implement Deletion in an AVL Tree

Delete a node from an AVL tree and rebalance the tree if required.
Test all four imbalance cases after deletion.

Example:
Delete `10` from:

```
   20
   /  \
  10  30
      \
       40
```

Perform **Left Rotation** after deletion.

3. Find Balance Factor of Every Node

For each node in the AVL tree, calculate and print the **balance factor** (Left Height − Right Height).

Example:
If for node 20, `Left = 2, Right = 1`
Balance Factor = +1

4. Perform All Four Rotations (LL, RR, LR, RL)

Demonstrate each type of rotation with examples.

Example:

- LL Rotation → Right Rotation
- RR Rotation → Left Rotation
- LR Rotation → Left-Right Rotation
- RL Rotation → Right-Left Rotation

Note: You should manually **dry run** each case.

5. Check if a Given Tree is AVL Balanced

Write a function to check if a binary tree satisfies AVL properties.
(Height difference ≤ 1 for every node.)

6. Construct an AVL Tree from a Given Sorted Array

Given a **sorted array**, build a **balanced AVL tree** from it.

Example:
Input: [1, 2, 3, 4, 5, 6, 7]
Output:

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

This is a perfectly balanced AVL tree.

Note: When a user selects this option, you've to ask about the size of the array and then create an array of that size. Ask for values from the user. And then pass that array to this function and return/construct an AVL tree.

7. Print the AVL Tree in Level Order

Perform **BFS traversal** using a queue to print the AVL tree level-by-level. It visually verifies if the tree is balanced.

Example:
For tree:

```
   30
   /  \
  20  40
 /      \
10       50
```

Output: 30 20 40 10 50