**Problem Name:** Balanced Binary tree

**Topics:**

**Companies:**

**Level:** Hard

**Language:** C++

**Problem Statement:**

**Input Format:**

**Output Format:**

**Constraints:**

**Examples:**

**Brute force Solution:**

**Explanation:**

It is intuitive to think that, for every node in the binary tree, we can check whether or not the left and right subtrees follow the required condition.

1. Start from the root and keep traversing the binary tree until the **root** becomes **NULL**
2. Retrieve the height of left and right subtrees using **height()** function
   * If the difference is more than **‘1’:**
     + return false. As the tree does not satisfy the balance condition
   * Check the balance condition for left and right subtrees recursively
3. Print the result

**Code:**

**Time Complexity**: O(N2) in the worst case(skewed tree).

**Space Complexity:** O(N) recursion uses the auxiliary stack space.

**Optimized Solution:**

**Explanation:**

We follow an approach such that we solve the problem in a **Bottom-Up** manner. Check whether the subtrees of a node are itself, balanced binary [trees](https://www.tutorialcup.com/interview/tree)(or not) and obtain the height of the binary tree at the same time, which can be generalized using recursion.

1. If the tree is **empty**, we can say it’s balanced. If not, we can follow other steps:
2. Create a helper function to return the “*height*” of a current subtree, using recursion.
3. Height Function will return:
   * -1, when it’s an unbalanced tree
   * the height otherwise.
4. In case a subtree has left or right subtree **unbalanced,**or their heights differ by more than ‘1’, return -1.
5. Otherwise, return the height of this subtree as : **1 + max(left\_height, right\_height)**

**Code:**

**Time Complexity**: O(N)

**Space Complexity:** O(N) recursion uses the auxiliary stack space.