1. Given the root of a binary tree, return *the maximum width of the given tree*.

The maximum width of a tree is the maximum width among all levels.

The width of one level is defined as the length between the end-nodes (the leftmost and rightmost non-null nodes), where the null nodes between the end-nodes that would be present in a complete binary tree extending down to that level are also counted into the length calculation.

It is guaranteed that the answer will in the range of a 32-bit signed integer.

Sol:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int widthOfBinaryTree(TreeNode root) {
        Deque<Pair<TreeNode, Integer>> q = new ArrayDeque<>();
        q.offer(new Pair<>(root, 1));
        int ans = 0;
        while (!q.isEmpty()) {
            ans = Math.max(ans, q.peekLast().getValue() - q.peekFirst().getValue() + 1);
            for (int n = q.size(); n > 0; --n) {
```

```
            var p = q.pollFirst();
            root = p.getKey();
            int i = p.getValue();
            if (root.left != null) {
                q.offer(new Pair<>(root.left, i << 1));
            }
            if (root.right != null) {
                q.offer(new Pair<>(root.right, i << 1 | 1));
            }
        }
    }
    return ans;
  }
}
```

2. You are given two 2D integer arrays nums1 and nums2.

- **nums1[i] = [id_i, val_i] indicate that the number with the id id_i has a value equal to val_i.**
- **nums2[i] = [id_i, val_i] indicate that the number with the id id_i has a value equal to val_i.**

Each array contains unique ids and is sorted in ascending order by id.

Merge the two arrays into one array that is sorted in ascending order by id, respecting the following conditions:

- **Only ids that appear in at least one of the two arrays should be included in the resulting array.**
- **Each id should be included only once and its value should be the sum of the values of this id in the two arrays. If the id does not exist in one of the two arrays then its value in that array is considered to be 0.**

Return *the resulting array*. The returned array must be sorted in ascending order by id.

Sol: public class Solutions {

```
    public static List<int[]> mergeArrays(int[][] nums1, int[][] nums2) {
        List<int[]> result = new ArrayList<>();
        int i = 0, j = 0;

        while (i < nums1.length && j < nums2.length) {
            if (nums1[i][0] == nums2[j][0]) {
                // IDs are the same, sum the values
                result.add(new int[] {nums1[i][0], nums1[i][1] + nums2[j][1]});
                i++;
                j++;
            } else if (nums1[i][0] < nums2[j][0]) {
                // nums1 ID is smaller, add it to result
                result.add(nums1[i]);
                i++;
            } else {
                // nums2 ID is smaller, add it to result
                result.add(nums2[j]);
                j++;
            }
        }

        // Add remaining elements from nums1 if any
        while (i < nums1.length) {
            result.add(nums1[i]);
            i++;
        }

        // Add remaining elements from nums2 if any
        while (j < nums2.length) {
```

```
            result.add(nums2[j]);
            j++;
        }

        return result;
    }

    public static void main(String[] args) {
        int[][] nums1 = {{1, 2}, {2, 3}, {4, 5}};
        int[][] nums2 = {{1, 4}, {3, 2}, {4, 1}};

        List<int[]> mergedArray = mergeArrays(nums1, nums2);

        // Print the result
        for (int[] arr : mergedArray) {
            System.out.println("[" + arr[0] + ", " + arr[1] + "]");
        }
    }
}
```

3. Given two binary strings a and b, return *their sum as a binary string*.

Sol: class Solution {

```
    public String addBinary(String a, String b) {
        var sb = new StringBuilder();
        int i = a.length() - 1, j = b.length() - 1;
        for (int carry = 0; i >= 0 || j >= 0 || carry > 0; --i, --j) {
            carry += (i >= 0 ? a.charAt(i) - '0' : 0) + (j >= 0 ? b.charAt(j) - '0' : 0);
            sb.append(carry % 2);
            carry /= 2;
        }
        return sb.reverse().toString();
    }
}
```

4. Write an algorithm to determine if a number n is happy.

A happy number is a number defined by the following process:

- **Starting with any positive integer, replace the number by the sum of the squares of its digits.**
- **Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.**
- **Those numbers for which this process ends in 1 are happy.**

Return true *if* n *is a happy number, and* false *if not*.

Sol: class Solution {

```
    public boolean isHappy(int n) {
        Set<Integer> vis = new HashSet<>();
        while (n != 1 && !vis.contains(n)) {
            vis.add(n);
            int x = 0;
            while (n != 0) {
                x += (n % 10) * (n % 10);
```

```
            n /= 10;
        }
        n = x;
    }
    return n == 1;
  }
}
```

5. Given an integer n, return *a string array* answer *(1-indexed) where*:

- **answer[i] == "FizzBuzz" if i is divisible by 3 and 5.**
- **answer[i] == "Fizz" if i is divisible by 3.**
- **answer[i] == "Buzz" if i is divisible by 5.**
- **answer[i] == i (as a string) if none of the above conditions are true.**

Sol:

```
class Solution {
  public List<String> fizzBuzz(int n) {
    List<String> ans = new ArrayList<>();
    for (int i = 1; i <= n; ++i) {
      String s = "";
      if (i % 3 == 0) {
        s += "Fizz";
      }
      if (i % 5 == 0) {
        s += "Buzz";
      }
      if (s.length() == 0) {
        s += i;
      }
      ans.add(s);
    }
```

```
        return ans;
    }
}
```

6. Given an integer n, return true *if it is possible to represent* n *as the sum of distinct powers of three.* Otherwise, return false.

An integer y is a power of three if there exists an integer x such that y == 3^x.

Sol:

```
class Solution {
    public boolean checkPowersOfThree(int n) {
        while (n > 0) {
            if (n % 3 > 1) {
                return false;
            }
            n /= 3;
        }
        return true;
    }
}
```