Worksheet - 10

Ques 1 : https://leetcode.com/problems/maximum-width-of-binary-tree/

Code :

```
class Solution {
    class Node {
        TreeNode node;
        int idx;
        Node(TreeNode node, int idx){
            this.node = node;
            this.idx = idx;
        }
    }
    public int widthOfBinaryTree(TreeNode root) {
        Queue<Node> queue = new LinkedList<>();
        queue.add(new Node(root,0));
        int max = 0;
        while(!queue.isEmpty()){
            int size = queue.size();
            int start = 0, end = 0;
            for(int i=0; i<size; i++){
                Node eachNode = queue.remove();
                int index = eachNode.idx;
                if(i==0)
                    start = index; //start and end index for each level
                if(i==size-1)
                    end = index;
                if(eachNode.node.left!=null)
                    queue.add(new Node(eachNode.node.left, 2*eachNode.idx));
                if(eachNode.node.right!=null)
                    queue.add(new Node(eachNode.node.right, 2*eachNode.idx+1));
            }
            max = Math.max(max, end - start + 1);
        }
        return max;
    }
}
```

Explanation :

In this question, the maximum width of a binary tree is defined as the number of nodes between the leftmost and rightmost nodes at that level.
Node Class : The Node class wraps each TreeNode (representing a node in the binary tree) and an associated idx (index) that simulates the position of the node in a complete binary tree.
Queue (BFS) : A queue is used for a level-order traversal (breadth-first search). For each node, it tracks both the node itself and its index in a conceptual complete binary tree.
Main Loop : For each level of the tree, the size of the queue represents the number of nodes at that level.
It tracks the index of the first and last nodes in the level (start and end).
For the left child of the current node, the index is calculated as 2*idx, and for the right child, it is 2*idx + 1, maintaining the structure of a complete binary tree.
Width calculation : The width of the current level is computed as end - start + 1 (distance between the first and last nodes). The maximum width across all levels is updated with Math.max(max, width).
Return value : Finally, the function returns the maximum width of any level found during the traversal.
Input: root = [1,3,2,5,null,null,9,6,null,7]
Output: 7
Explanation: The maximum width exists in the fourth level with length 7 (6,null,null,null,null,null,7).

Ques 2 : https://leetcode.com/problems/merge-two-2d-arrays-by-summing-values/
Code :

```
class Solution {
    public int[][] mergeArrays(int[][] nums1, int[][] nums2) {
        int max = Math.max(nums1[nums1.length - 1][0], nums2[nums2.length - 1][0]);
        int[][] ans = new int[max][2];

        for ( int[] ints : nums1 ) {
            ans[ints[0] - 1][0] = ints[0];
            ans[ints[0] - 1][1] = ans[ints[0] - 1][1] + ints[1];
        }
        for ( int[] ints : nums2 ) {
            ans[ints[0] - 1][0] = ints[0];
            ans[ints[0] - 1][1] = ans[ints[0] - 1][1] + ints[1];
        }
        int count = 0;
```

```
        for ( int[] an : ans ) if ( an[0] != 0 ) count++;

        int[][] fans = new int[count][2];
        count = 0;

        for ( int[] an : ans )
            if ( an[0] != 0 )
                fans[count++] = an;

        return fans;
    }
}
```

Explanation :

This Java code defines a method mergeArrays that takes two 2D integer arrays, nums1 and nums2, where each subarray represents a pair of integers (index and value). The purpose of the method is to merge these two arrays based on the first element of each subarray (the index) and sum the corresponding values.

1. Determine Maximum Index: It finds the maximum index from both input arrays to create an output array (ans) that can accommodate all possible indices.
2. Populate Merged Data: It iterates through each input array, using the first element of each subarray as the index to store and sum the values in the ans array.
3. Count Valid Entries: It counts how many unique indices have been populated in the ans array.
4. Prepare Final Output: It creates a new 2D array (fans) to store only the valid (non-zero) entries from ans, which holds the merged results.

Input: nums1 = [[1,2],[2,3],[4,5]], nums2 = [[1,4],[3,2],[4,1]]
Output: [[1,6],[2,3],[3,2],[4,6]]
Explanation: The resulting array contains the following:
- id = 1, the value of this id is 2 + 4 = 6.
- id = 2, the value of this id is 3.
- id = 3, the value of this id is 2.
- id = 4, the value of this id is 5 + 1 = 6.

Ques 3 : https://leetcode.com/problems/add-binary/
Code :

```
class Solution {
    public String addBinary(String a, String b) {
```

```
        StringBuilder sb = new StringBuilder();
        int carry = 0;
        int i = a.length() - 1;
        int j = b.length() - 1;

        while (i >= 0 || j >= 0 || carry == 1) {
            if (i >= 0)
                carry += a.charAt(i--) - '0';
            if (j >= 0)
                carry += b.charAt(j--) - '0';
            sb.append(carry % 2);
            carry /= 2;
        }
        return sb.reverse().toString();
    }
}
```

Explanation :

A StringBuilder object sb is initialized to store the binary result. It allows efficient appending of characters and later reversing of the final string.

Carry is initialized to 0 to handle cases where the sum of two bits exceeds 1.

Two variables, i and j, are initialized to point to the last index (least significant bit) of strings a and b, respectively. This allows processing both binary strings from right to left.

The loop continues as long as there are bits left in either string (i >= 0 || j >= 0) or there is a carry left to process (carry == 1).

If there are bits remaining in string a, add the current bit from a to carry : a.charAt(i) gets the character at index i, which is either '0' or '1'. Subtracting '0' from the character converts it to an integer (since '1' - '0' = 1 and '0' - '0' = 0). After this, i-- decrements the index to move leftward.

Similarly, for string b, if there are bits left, the current bit from b is added to carry.

After adding the bits and the carry, carry % 2 gives the current bit of the result (either 0 or 1). This bit is appended to the StringBuilder.

The carry is updated by dividing carry by 2. If the sum of bits was 2 or more, this results in carry being set to 1 for the next iteration. Otherwise, carry becomes 0.Since the binary string was built from the least significant bit (rightmost) to the most significant bit (leftmost), the StringBuilder is reversed before converting it to a string and returning the result.

Input: a = "11", b = "1"

Output: "100"

Ques 4 : https://leetcode.com/problems/happy-number/

Code :

```
class Solution{
   public boolean isHappy(int n) {
           Set<Integer> set = new HashSet<>();
       while(true){
                int sum=0;
          while(n!=0){
             sum += Math.pow(n%10,2.0);
             n = n/10;
          }
                  if(sum ==1){
             return true;
          }
           n= sum;
           if(set.contains(n)){
                 return false;
           }
           set.add(n);
           }
      }
}
```

Explaination :

HashSet to detect when the number sum is already present in set or not

find the sum of squares by using the Math.pow method. After calculating the sum of squares check if sum is equal to 1 then return true otherwise put a number equal to the sum and check if the set contains the number n or not. if yes return false otherwise add number n in the set repeat the process until the sum is equal to 1 .

Input : 31

Output: true

Input : 2

Output : false

Ques 5 : https://leetcode.com/problems/fizz-buzz/

Code :

```
class Solution {
```

```
    public List<String> fizzBuzz(int n) {
        ArrayList<String> list = new ArrayList<>();
        for(int i=1;i<=n;i++){
            if(i%3==0 && i%5==0){
                list.add("FizzBuzz");
            }
            else if(i%3==0){
                list.add("Fizz");
            }
            else if(i%5==0){
                list.add("Buzz");
            }
            else{
                list.add(Integer.toString(i));
            }
        }
        return list;
    }
}
```

Explanation :

The goal is to print numbers from 1 to n, but with the following rules:

1. If a number is divisible by 3, output "Fizz".
2. If a number is divisible by 5, output "Buzz".
3. If a number is divisible by both 3 and 5, output "FizzBuzz".
4. Otherwise, output the number itself as a string.

Input: n = 5
Output: ["1","2","Fizz","4","Buzz"]

Ques 6 :

https://leetcode.com/problems/check-if-number-is-a-sum-of-powers-of-three/description/

Code :

```
class Solution {
    public boolean checkPowersOfThree(int n) {
        while (n > 0) {
            if (n % 3 == 2) return false;
            else n /= 3;
```

```
        }
        return true;
    }
}
```

Explanation :

This is a public method that takes an integer n and returns a boolean (true or false). It checks if n can be represented as a sum of distinct powers of 3.

The loop runs as long as n is greater than 0. It processes n by repeatedly dividing it by 3

The expression n % 3 gets the remainder when n is divided by 3. In base-3:

1. If the remainder is 0, the current "digit" in base-3 is 0.
2. If the remainder is 1, the current "digit" in base-3 is 1.
3. If the remainder is 2, this means n has a "2" in its ternary representation, which makes it impossible to represent n as a sum of distinct powers of 3. Therefore, the method returns false if it encounters this condition.

If the remainder is not 2, divide n by 3 to continue checking the next "digit" in its ternary representation. This reduces n and prepares for the next iteration of the loop.

If the loop completes without encountering a remainder of 2, it means all the "digits" in the base-3 representation of n are either 0 or 1. This implies that n can be represented as a sum of distinct powers of 3, so the method returns true.

Input: n = 12

Output: true

Explanation: 12 = 31 + 32