


## 2. FizzBuzz (LeetCode 412) [Asked by LinkedIn]

```
Java Auto
1 class Solution {
2     public List<String> fizzBuzz(int n) {
3         List<String> answer = new ArrayList<>();
4         for (int i = 1; i <= n; i++) {
5             if (i % 3 == 0 && i % 5 == 0) { // i is divisible by 3 & 5: FizzBuzz
6                 answer.add("Fizzbuzz");
7             } else if (i % 3 == 0) { // i is divisible by 3
8                 answer.add("Fizz");
9             } else if (i % 5 == 0) { // i is divisible by 5
10                answer.add("Buzz");
11            } else {
12                answer.add(i + "");
13            }
14        }
15        return answer;
16    }
17 } // TC: O(n), SC: O(n) I
```



It's through an error

Hint : `answer.add("fizzBuzz");`

## 3. Single Number (LeetCode 136) [Asked by Amazon]


Brute force approach:

```
> Problem List < > < Run < Submit < < 80 <
Code
Java Auto
1 class Solution {
2     public int singleNumber(int[] nums) {
3         if (nums.length == 1) return nums[0];
4         Set<Integer> set = new HashSet<>();
5         for (int i = 0; i < nums.length; i++) {
6             if (set.contains(nums[i])) {
7                 set.remove(nums[i]);
8             } else {
9                 set.add(nums[i]);
10            }
11        }
12        return set.iterator().next();
13    }
14 } // TC: O(n), SC: O(n) I
```

—INSERT—

Saved to local

Testcase > Result



Optimal Solution:


```
> Problem List < > < Run < Submit < < 80 <
Code
Java Auto
1 class Solution {
2     public int singleNumber(int[] nums) {
3         int singleNum = 0;
4         for (int num : nums) {
5             singleNum = singleNum ^ num;
6         }
7         return singleNum;
8     }
9 }
10 // TC: O(n), SC: O(1)
```

—INSERT—

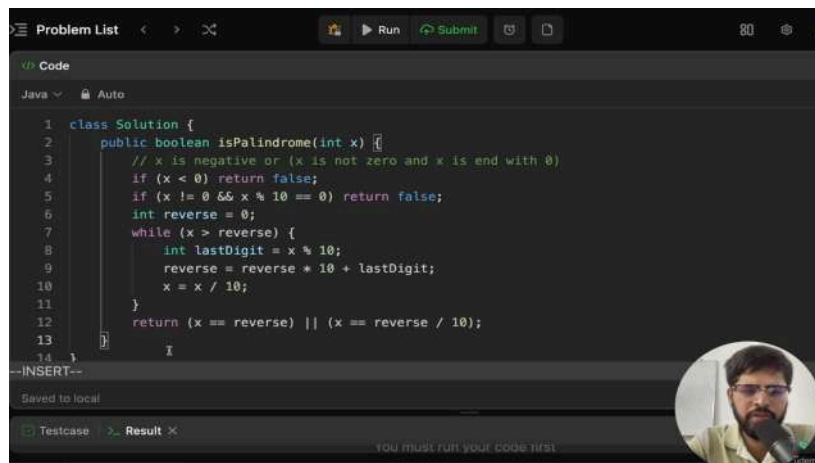
Saved to local

Testcase > Result

You must run your code first

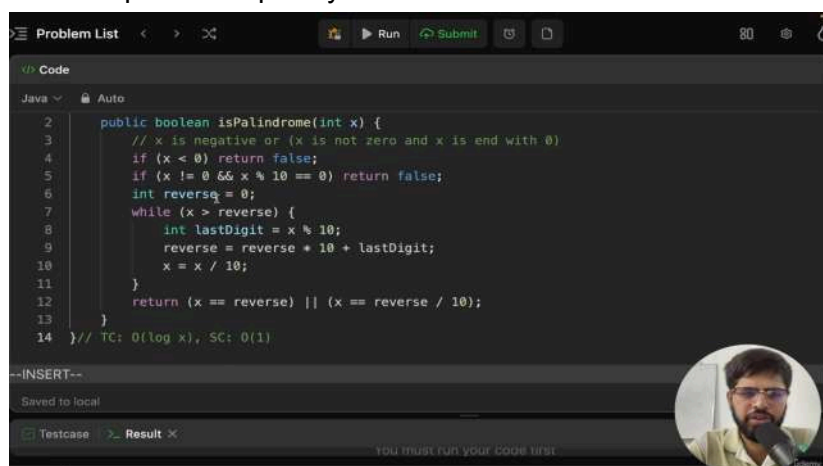


## 4. Palindrome Number (LeetCode 9) [Asked by Facebook/Bloomberg]



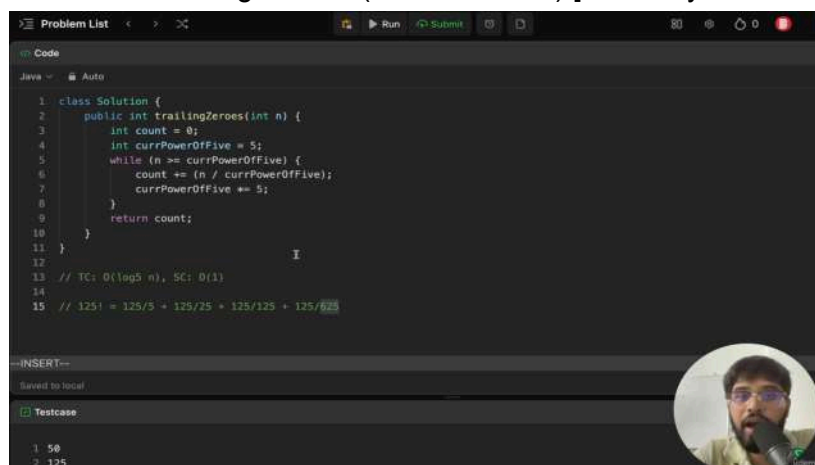
```
1 class Solution {
2     public boolean isPalindrome(int x) {
3         // x is negative or (x is not zero and x is end with 0)
4         if (x < 0) return false;
5         if (x != 0 && x % 10 == 0) return false;
6         int reverse = 0;
7         while (x > reverse) {
8             int lastDigit = x % 10;
9             reverse = reverse * 10 + lastDigit;
10            x = x / 10;
11        }
12        return (x == reverse) || (x == reverse / 10);
13    }
14 }
```

Time & Space Complexity:



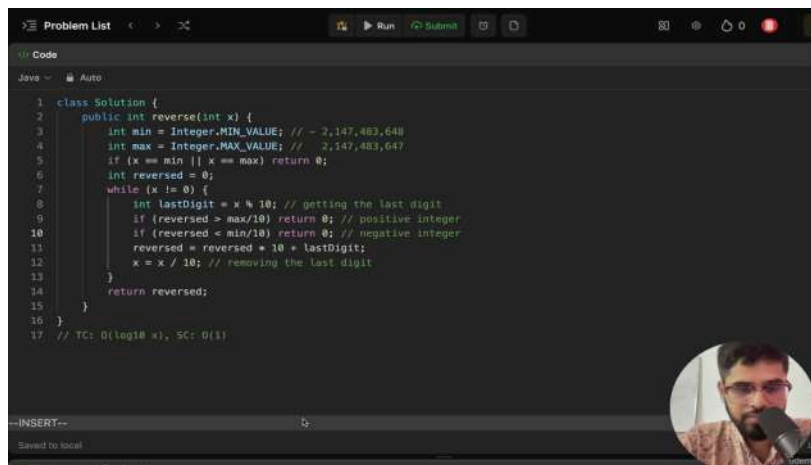
```
2     public boolean isPalindrome(int x) {
3         // x is negative or (x is not zero and x is end with 0)
4         if (x < 0) return false;
5         if (x != 0 && x % 10 == 0) return false;
6         int reverse = 0;
7         while (x > reverse) {
8             int lastDigit = x % 10;
9             reverse = reverse * 10 + lastDigit;
10            x = x / 10;
11        }
12        return (x == reverse) || (x == reverse / 10);
13    }
14 } // TC: O(log x), SC: O(1)
```

5. Factorial Trailing Zeroes (LeetCode 172) [Asked by Microsoft]



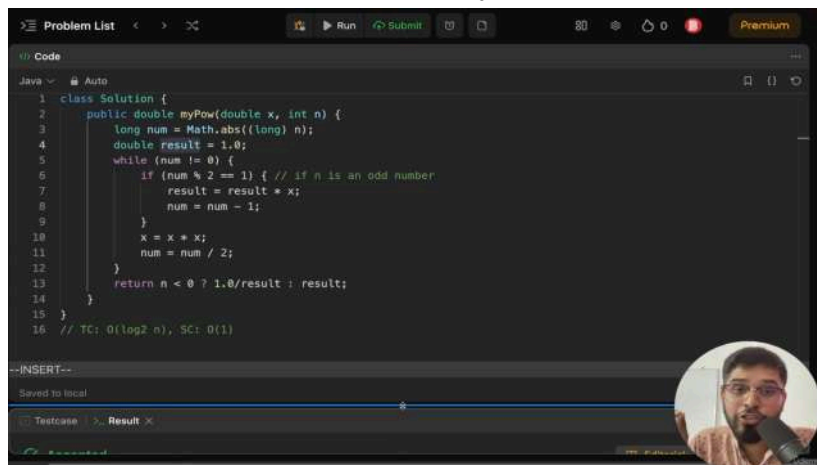
```
1 class Solution {
2     public int trailingZeroes(int n) {
3         int count = 0;
4         int currPowerOfFive = 5;
5         while (n >= currPowerOfFive) {
6             count += (n / currPowerOfFive);
7             currPowerOfFive *= 5;
8         }
9         return count;
10    }
11 }
12 // TC: O(log5 n), SC: O(1)
13 // 125! = 125/5 + 125/25 + 125/125 + 125/625
```

6. Reverse Integer (LeetCode 7) [Asked by Facebook]



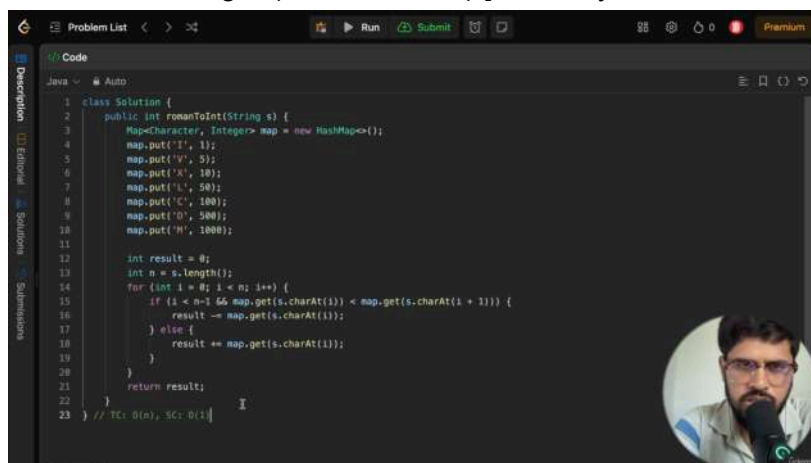
```
1 class Solution {
2     public int reverse(int x) {
3         int min = Integer.MIN_VALUE; // - 2,147,483,648
4         int max = Integer.MAX_VALUE; // 2,147,483,647
5         if (x == min || x == max) return 0;
6         int reversed = 0;
7         while (x != 0) {
8             int lastDigit = x % 10; // getting the last digit
9             if (reversed > max/10) return 0; // positive integer
10            if (reversed < min/10) return 0; // negative integer
11            reversed = reversed * 10 + lastDigit;
12            x = x / 10; // removing the last digit
13        }
14        return reversed;
15    }
16 }
17 // TC: O(log10 x), SC: O(1)
```

## 7. Pow(x, n) (LeetCode 50) [Asked by Facebook/Amazon/LinkedIn]



```
1 class Solution {
2     public double myPow(double x, int n) {
3         long num = Math.abs((long) n);
4         double result = 1.0;
5         while (num != 0) {
6             if (num % 2 == 1) { // if n is an odd number
7                 result = result * x;
8                 num = num - 1;
9             }
10            x = x * x;
11            num = num / 2;
12        }
13        return n < 0 ? 1.0/result : result;
14    }
15 }
16 // TC: O(log2 n), SC: O(1)
```

## 8. Roman to Integer (LeetCode 344) [Asked by MicroSoft/Uber]



```
1 class Solution {
2     public int romanToInt(String s) {
3         Map<Character, Integer> map = new HashMap<>();
4         map.put('I', 1);
5         map.put('V', 5);
6         map.put('X', 10);
7         map.put('L', 50);
8         map.put('C', 100);
9         map.put('D', 500);
10        map.put('M', 1000);
11
12        int result = 0;
13        int n = s.length();
14        for (int i = 0; i < n; i++) {
15            if (i < n-1 && map.get(s.charAt(i)) < map.get(s.charAt(i+1))) {
16                result -= map.get(s.charAt(i));
17            } else {
18                result += map.get(s.charAt(i));
19            }
20        }
21        return result;
22    }
23 } // TC: O(n), SC: O(1)
```

## 9. Integer to Roman

```

1 class Solution {
2     public String intToRoman(int num) {
3         int[] storeInt = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
4         String[] storeRoman = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};
5         String romanNumeral = "";
6         for (int i = 0; i < storeInt.length; i++) {
7             while (num >= storeInt[i]) {
8                 String symbol = storeRoman[i];
9                 romanNumeral += symbol;
10                num -= storeInt[i];
11            }
12        }
13        return romanNumeral;
14    }
15 }
16 // TC: O(13 * log n), SC: O(13)

```

## 10.Reverse a String

```

1 class Solution {
2     public void reverseString(char[] s) {
3         int left = 0, right = s.length - 1;
4         while (left < right) {
5             char tmp = s[left];
6             s[left] = s[right];
7             s[right] = tmp;
8             left++;
9             right--;
10        }
11    }
12 } // TC: O(n), SC: O(1)

```

## 9. Longest Common Prefix (LeetCode 14) [Asked by Facebook]

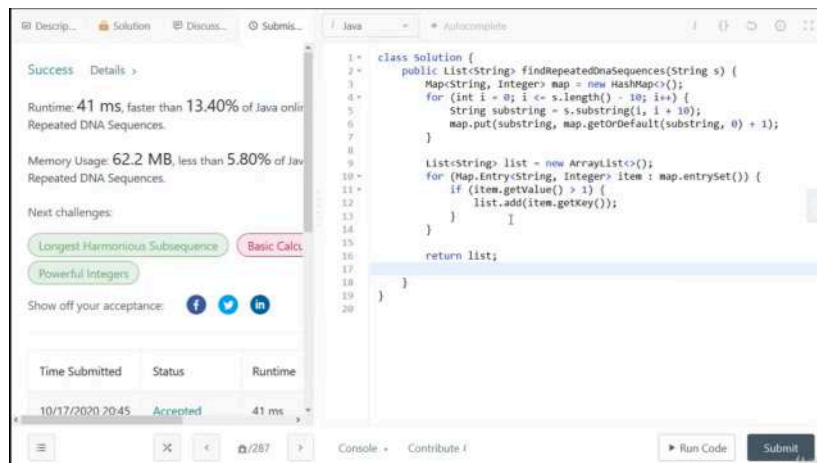
```

1 class Solution {
2     public String longestCommonPrefix(String[] strs) {
3         if (strs.length == 0) return "";
4         String prefix = strs[0];
5         for (int i = 1; i < strs.length; i++) {
6             while (strs[i].indexOf(prefix) != 0) {
7                 prefix = prefix.substring(0, prefix.length() - 1);
8                 if (prefix.isEmpty()) return "";
9             }
10        }
11        return prefix;
12    }
13 }
14
15 /*
16 TC: O(m*n), SC: O(1)
17 prefix = "fl"
18 ["flower", "flow", "flight"] */

```

## 10. Repeated DNA Sequences (LeetCode 187) [Asked by Google]

HashMap:



Set ds:

11. Valid Anagram (LeetCode 242) [Asked by Facebook]

12. Longest Palindromic Substring (LeetCode 5) [Asked by Facebook]

```

1 class Solution {
2     int start = 0, end = 0;
3     public String longestPalindrome(String s) {
4         for (int i = 0; i < s.length(); i++) {
5             expandAroundCenter(s, i, i);
6             expandAroundCenter(s, i, i + 1);
7         }
8         return s.substring(start, end + 1);
9     }
10    private void expandAroundCenter(String s, int left, int right) {
11        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
12            left--;
13            right++;
14        }
15        left = left + 1; // from left to right => palindromic substring
16        right = right - 1;
17        if (end - start + 1 < right - left + 1) {
18            start = left;
19            end = right;
20        }
21    }
22 }
23

```

13. Longest Substring without Repeating Characters (LeetCode 3) [Asked by Google]

```

1 class Solution {
2     public int lengthOfLongestSubstring(String s) {
3         int ans = 0;
4         Map<Character, Integer> map = new HashMap<>(); // n
5         int left = 0;
6         for (int right = 0; right < s.length(); right++) { // n
7             char ch = s.charAt(right);
8             if (!map.containsKey(ch)) {
9                 map.put(ch, right);
10            } else {
11                left = Math.max(left, map.get(ch) + 1);
12                map.put(ch, right);
13            }
14            ans = Math.max(ans, right - left + 1); // Left to right
15        }
16        return ans;
17    }
18 }
19 // TC: O(n), SC: O(n)

```

14. Reverse Words in a String (LeetCode 151) [Asked by Facebook/Amazon]

15. Integer to English Words (LeetCode 273) [Facebook, Microsoft]

16. Binary Search (LeetCode 704) [Asked by Infosys,Oracle]

```
1 class Solution {
2     public int search(int[] nums, int target) {
3         int left = 0;
4         int right = nums.length - 1;
5         while (left <= right) {
6             int mid = left + (right - left) / 2; // (left + right) / 2
7             if (nums[mid] == target) {
8                 return mid;
9             }
10            if (target > nums[mid]) {
11                left = mid + 1;
12            } else {
13                right = mid - 1;
14            }
15        }
16        return -1;
17    }
18 } // TC: O(log n), SC: O(1)
```

17. Search in Rotated Sorted Array (LeetCode 33) [Asked by Facebook]

```
1 class Solution {
2     public int search(int[] nums, int target) {
3         int left = 0;
4         int right = nums.length - 1;
5         while (left <= right) {
6             int mid = left + (right - left) / 2;
7             if (nums[mid] == target) return mid;
8             if (nums[left] <= nums[mid]) { // left to right is sorted
9                 if (nums[left] <= target && target < nums[mid])
10                    right = mid - 1;
11                 else
12                    left = mid + 1;
13            } else { // mid to right is sorted
14                if (nums[mid] < target && target <= nums[right])
15                    left = mid + 1;
16                else
17                    right = mid - 1;
18            }
19        }
20        return -1;
21    }
22 } // TC: O(log n), SC: O(1)
```

18. Find Minimum in Rotated Sorted Array (LeetCode 153) [Asked by Facebook]

```
1 class Solution {
2     public int findMin(int[] nums) {
3         if (nums.length == 1) return nums[0];
4         if (nums.length == 2) return Math.min(nums[0], nums[1]);
5         if (nums[0] < nums[nums.length - 1]) return nums[0];
6         int left = 0;
7         int right = nums.length - 1;
8         while (left <= right) {
9             int mid = left + (right - left) / 2; // (left + right) / 2
10            // the array is decreasing at mid + 1
11            if (nums[mid] > nums[mid+1]) return nums[mid+1];
12            // the array is decreasing at mid
13            if (nums[mid-1] > nums[mid]) return nums[mid];
14            // discard the sorted part > increasing part
15            if (nums[left] < nums[mid])
16                left = mid + 1;
17            else
18                right = mid - 1;
19        }
20        return 0;
21    }
22 } // TC: O(log n), SC: O(1)
```

19. Two Sum (LeetCode 1) [Asked by Google]

```

class Solution {
    public int[] twoSum(int[] nums, int target) {
        int[] ans = new int[2];
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) { // n
            int diff = target - nums[i];
            if (map.containsKey(diff)) {
                ans[0] = i;
                ans[1] = map.get(diff);
                break;
            } else {
                map.put(nums[i], i);
            }
        }
        return ans;
    }
} // TC: O(n), SC: O(n)

```

## 20. Move Zeroes (LeetCode 283) [Asked by Facebook]

```

1. class Solution {
2.     public void moveZeroes(int[] nums) {
3.         if (nums.length == 1) return;
4.         int left = 0;
5.         for (int right = 0; right < nums.length; right++) {
6.             if (nums[right] != 0) {
7.                 nums[left] = nums[right];
8.                 left++;
9.             }
10.        }
11.        // from index: left to nums.length-1, with 0's
12.        for (int i = left; i < nums.length; i++) {
13.            nums[i] = 0;
14.        }
15.    }
16. } // TC: O(n), SC: O(1)
17.

```

## 21. Best Time to Buy And Sell Stock(LeetCode 121) [Asked by Facebook/Microsoft]

```

1. class Solution {
2.     public int maxProfit(int[] prices) {
3.         if (prices.length == 1) return 0;
4.
5.         int profit = 0;
6.         int left = 0;
7.         for (int right = 1; right < prices.length; right++) {
8.             if (prices[right] > prices[left]) {
9.                 profit = Math.max(profit, prices[right] - prices[left]);
10.            } else {
11.                left = right;
12.            }
13.        }
14.        return profit;
15.    }
16. }
17.
18. // TC: O(n), SC: O(1)
19.

```

## 22. Ransom Note (LeetCode 383) [Asked by Amazon]



```
1 class Solution {
2     public boolean canConstruct(String ransomNote, String magazine) {
3         int[] frequency = new int[26];
4         for (int i = 0; i < magazine.length(); i++) {
5             char c = magazine.charAt(i);
6             frequency[c - 'a']++;
7         }
8         for (int i = 0; i < ransomNote.length(); i++) {
9             char c = ransomNote.charAt(i);
10            if (frequency[c - 'a'] == 0) return false;
11            frequency[c - 'a']--;
12        }
13        return true;
14    }
15 } // TC: O(m + n), SC: O(1)
```

### 23. Contains Duplicate (LeetCode 217) [Asked by Google]

```
1 class Solution {
2     public boolean containsDuplicate(int[] nums) {
3         Set<Integer> set = new HashSet<>();
4         for (int i = 0; i < nums.length; i++) {
5             if (set.contains(nums[i])) {
6                 return true;
7             } else {
8                 set.add(nums[i]);
9             }
10        }
11        return false;
12    }
13 } // TC: O(n), SC: O(n)
```

### 24. Length of Last Word (LeetCode 58) [Asked by Amazon]

```
1 class Solution {
2     public int lengthOfLastWord(String s) {
3         int right = s.length() - 1;
4         while (right >= 0 && s.charAt(right) == ' ') {
5             right--;
6         }
7         int left = right;
8         while (left >= 0 && s.charAt(left) != ' ') {
9             left--;
10        }
11        return right - left;
12    }
13 } // TC: O(n), SC: O(1)
```

### 25. Best Time to Buy and Sell Stock II (LeetCode 122) [Asked by Google]



```

1 * class Solution {
2 *     public int maxProfit(int[] prices) {
3 *         int profit = 0;
4 *         for (int i = 1; i < prices.length; i++) {
5 *             if (prices[i-1] < prices[i])
6 *                 profit += prices[i] - prices[i-1];
7 *             }
8 *         return profit;
9 *     }
10 } // TC: O(n), SC: O(1)

```

## 26. Rotate Array (LeetCode 189) [Asked by Microsoft]

Success Details >

Runtime: 0 ms, faster than 100.00% of Java solutions for Rotate Array.

Memory Usage: 39.4 MB, less than 12.69% of Java solutions for Rotate Array.

Next challenges: [Reverse Words in a String II](#)

Show off your acceptance: [Facebook](#) [Twitter](#) [LinkedIn](#)

Time Submitted	Status	Runtime
10/15/2020 21:14	Accepted	0 ms

```

1 * class Solution {
2 *     public void rotate(int[] nums, int k) {
3 *         k = k % nums.length;
4 *         reverse(nums, 0, nums.length - 1);
5 *         reverse(nums, 0, k - 1);
6 *         reverse(nums, k, nums.length - 1);
7 *     }
8 *     private void reverse(int[] nums, int start, int end) {
9 *         while (start < end) {
10 *             int tap = nums[start];
11 *             nums[start] = nums[end];
12 *             nums[end] = tap;
13 *             start++;
14 *             end--;
15 *         }
16 *     }
17 * }
18 * // TC: O(3n) = O(n)
19 * // SC: O(1)

```

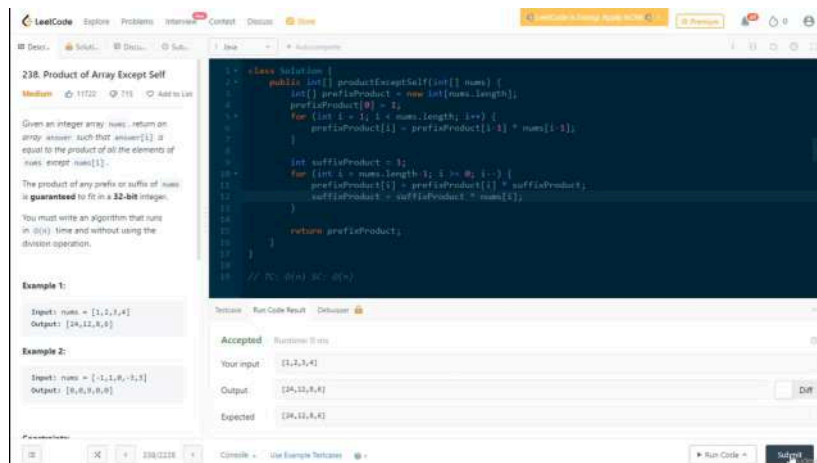
## 27. Jump Game (LeetCode 55) [Asked by Google]

```

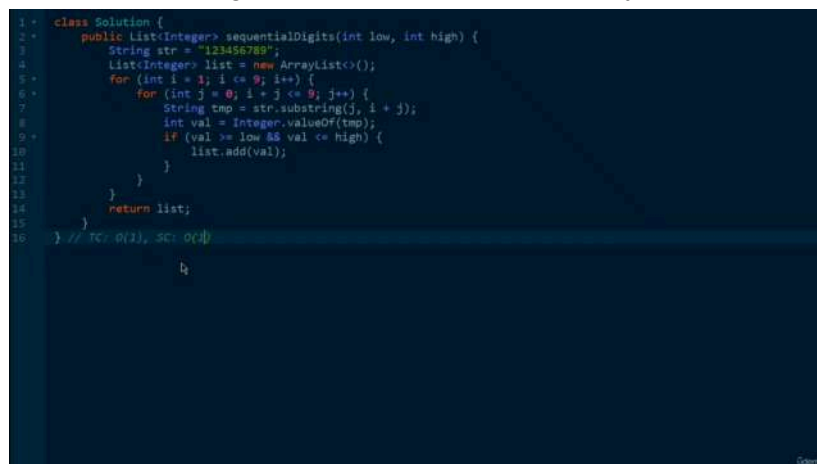
1 * class Solution {
2 *     public boolean canJump(int[] nums) {
3 *         int reachable = 0;
4 *         for (int i = 0; i < nums.length && i <= reachable; i++) {
5 *             reachable = Math.max(reachable, i + nums[i]);
6 *             if (reachable >= nums.length - 1)
7 *                 return true;
8 *         }
9 *         return false;
10 }
11 } // TC: O(n), SC: O(1)

```

## 28. Product of Array Except Self (LeetCode 238) [Asked by Facebook]



## 29. Sequential Digits (LeetCode 1291) [Asked by Facebook]



## 30. Find All Disappeared Numbers in an Array (LeetCode 448) [Asked by Google]



## 31. Find All Duplicates Numbers in an Array (Leetcode 442) [Asked by Facebook]

```

1 * class Solution {
2 *     public List<Integer> findDuplicates(int[] nums) {
3 *         List<Integer> list = new ArrayList<>();
4 *         for (int i = 0; i < nums.length; i++) {
5 *             int index = Math.abs(nums[i]) - 1;
6 *             if (nums[index] < 0) {
7 *                 list.add(Math.abs(nums[i]));
8 *             }
9 *             if (nums[index] > 0) {
10 *                 nums[index] = -1 * nums[index];
11 *             }
12 *         }
13 *         return list;
14 *     }
15 * } // TC: O(n), SC: O(1)

```

### 32. First Missing Positive (LeetCode 41) [Asked by Microsoft]

```

1 * class Solution {
2 *     public int firstMissingPositive(int[] nums) {
3 *         // preprocess the array
4 *         for (int i = 0; i < nums.length; i++) {
5 *             if (nums[i] <= 0) {
6 *                 nums[i] = nums.length + 1;
7 *             }
8 *         }
9 *         // marking indices
10 *         for (int i = 0; i < nums.length; i++) {
11 *             int index = Math.abs(nums[i]) - 1;
12 *             if (index < nums.length && nums[index] > 0) {
13 *                 nums[index] = -1 * nums[index];
14 *             }
15 *         }
16 *         // scan the array
17 *         for (int i = 0; i < nums.length; i++) {
18 *             if (nums[i] > 0) {
19 *                 return i + 1;
20 *             }
21 *         }
22 *         return nums.length + 1;
23 *     }
24 * } // TC: O(n), SC: O(1)

```

Testcase Run Code Result Debugger

### 33. Next Permutation (LeetCode 31) [Asked by Google]

```

1 * class Solution {
2 *     public void nextPermutation(int[] nums) {
3 *         int i = nums.length - 2;
4 *         while (i >= 0 && nums[i] >= nums[i + 1]) {
5 *             i--;
6 *         }
7 *         if (i >= 0) {
8 *             int j = nums.length - 1;
9 *             while (nums[i] >= nums[j]) {
10 *                 j--;
11 *             }
12 *             swap(nums, i, j);
13 *             reverse(nums, i + 1, nums.length - 1);
14 *         }
15 *         public void swap(int[] nums, int i, int j) {
16 *             int temp = nums[i];
17 *             nums[i] = nums[j];
18 *             nums[j] = temp;
19 *         }
20 *         public void reverse(int[] nums, int i, int j) {
21 *             while (i < j) {
22 *                 swap(nums, i, j);
23 *                 i++;
24 *                 j--;
25 *             }
26 *         }
27 *     }
28 * } // TC: O(n), SC: O(1)

```

2.13% of Java  
less than 49.32%  
Palindrome Perm  
Reach the Kth S  
f t in  
Run  
ted 1 m  
ted 1 m

### 34. Largest Subarray With 0 Sum [Asked by Microsoft]

```

class GFG
{
    int maxLen(int arr[], int n)
    {
        int sum = 0, len = 0;
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < n; i++) {
            sum += arr[i];
            if (sum == 0) {
                len = i + 1;
            } else if (map.containsKey(sum)) {
                len = Math.max(len, i - map.get(sum));
            } else {
                map.put(sum, i);
            }
        }
        return len;
    }
} // TC: O(n), SC: O(n)

```

### 35. Subarray Product Less than K (LeetCode 713) [Asked by Goldman Sachs]

```

1. class Solution {
2.     public int numSubarrayProductLessThanK(int[] nums, int k) {
3.         int prod = 1, count = 0, left = 0;
4.         for (int right = 0; right < nums.length; right++) {
5.             prod *= nums[right];
6.             while (left <= right && prod >= k) {
7.                 prod /= nums[left];
8.                 left++;
9.             }
10.            count += right - left + 1;
11.        }
12.        return count;
13.    }
14.} // O(n), O(1)

```

### 36. K-Diff Pairs in an Array (LeetCode 532) [Asked by Amazon]

```

1. class Solution {
2.     public int FindPairs(int[] nums, int k) {
3.         HashMap<Integer, Integer> map = new HashMap<>();
4.         for (int num : nums) {
5.             map.put(num, map.getOrDefault(num, 0) + 1);
6.         }
7.         int count = 0;
8.         for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
9.             if (k == 0) {
10.                if (entry.getValue() >= 2) {
11.                    count++;
12.                }
13.            } else {
14.                if (map.containsKey(entry.getKey() + k)) {
15.                    count++;
16.                }
17.            }
18.        }
19.        return count;
20.    }
21.} // TC: O(n), SC: O(n)

```

### 37. Two Sum II - Input Array Is Sorted (LeetCode 167) [Asked by Amazon]

```

1 class Solution {
2     public int[] twoSum(int[] numbers, int target) {
3         int left = 0, right = numbers.length - 1;
4         while (left < right) {
5             int sumOfCurrPair = numbers[left] + numbers
6         [right];
7             if (sumOfCurrPair == target) {
8                 break;
9             } else if (sumOfCurrPair > target) {
10                right--;
11            } else {
12                left++;
13            }
14            return new int[] {left + 1, right + 1};
15        }
16    }
17    // TC: O(n), SC: O(1)
18 }

```

### 38. Is Subsequence (LeetCode 392) [Asked by Google]

```

1
2
3 class Solution {
4     public boolean isSubsequence(String s, String t) {
5         int p1 = 0, p2 = 0;
6         while (p1 < s.length() && p2 < t.length()) {
7             if (s.charAt(p1) == t.charAt(p2)) {
8                 p1++;
9                 p2++;
10            } else {
11                p2++;
12            }
13        }
14        return p1 == s.length();
15    }
16 } // TC: O(n), SC: O(1)

```

### 39. Remove Duplicates from Sorted Array (LeetCode 26) [Asked by Facebook/Microsoft]

```

1
2
3 class Solution {
4     public int removeDuplicates(int[] nums) {
5         int left = 0, right = 0; // 0 ~ left
6         while (right < nums.length) {
7             if (nums[left] != nums[right]) {
8                 left++;
9                 nums[left] = nums[right];
10            }
11            right++;
12        }
13        return left + 1;
14    }
15 } // TC: O(n), SC: O(1)
16
17
18

```

### 40. Sort Colors (LeetCode 75) [Asked by Facebook]

```

1 * class Solution {
2 *     public void sortColors(int[] nums) {
3 *         int left = 0, mid = 0, right = nums.length - 1;
4 *         while (mid <= right) {
5 *             if (nums[mid] == 0) {
6 *                 swap(left, mid, nums);
7 *                 left++;
8 *                 mid++;
9 *             } else if (nums[mid] == 1) {
10 *                 mid++;
11 *             } else {
12 *                 swap(mid, right, nums);
13 *                 right--;
14 *             }
15 *         }
16 *     }
17 *     private void swap(int i, int j, int[] nums) {
18 *         int tmp = nums[i];
19 *         nums[i] = nums[j];
20 *         nums[j] = tmp;
21 *     }
22 * } // TC: O(n), SC: O(1)

```

#### 41. Valid Palindrome (LeetCode 125) [Asked by Facebook]

```

class Solution {
    public boolean isPalindrome(String s) {
        int p1 = 0, p2 = s.length() - 1;
        while (p1 <= p2) {
            char c1 = s.charAt(p1);
            char c2 = s.charAt(p2);
            if (Character.isLetterOrDigit(c1) == false) p1++;
            else if (Character.isLetterOrDigit(c2) == false) p2--;
            else {
                if (Character.toLowerCase(c1) != Character.toLowerCase(c2)) {
                    return false;
                }
                p1++;
                p2--;
            }
        }
        return true;
    }
} // TC: O(n), SC: O(1)

```

#### 42. Merge Sorted Array (LeetCode 80) [Asked by Microsoft]

### 88. Merge Sorted Array

Easy 4718 437 Add to List Share

You are given two integer arrays `nums1` and `nums2`, sorted in non-decreasing order, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to 0 and should be ignored. `nums2` has a length of `n`.

**Example 1:**

Input: `nums1 = [1,2,3,0,0]`, `m = 3`, `nums2 = [2,5,6]`, `n = 3`  
Output: `[1,2,2,3,5,6]`  
Explanation: The arrays are merged as `[1,2,3]` and `[2,5,6]`. The result of the merge is `[1,2,2,3,5,6]` with the underlined elements coming from `nums1`.

```

1 * class Solution {
2 *     public void merge(int[] nums1, int m, int[] nums2, int n) {
3 *         int p1 = m;
4 *         int p2 = 0;
5 *         for (int i = nums1.length - 1; i >= 0; i--) {
6 *             int valA = p1 >= m ? nums1[p1] : Integer.MIN_VALUE;
7 *             int valB = p2 >= n ? nums2[p2] : Integer.MIN_VALUE;
8 *             if (valA > valB) {
9 *                 nums1[i] = valA;
10 *                 p1++;
11 *             } else {
12 *                 nums1[i] = valB;
13 *                 p2++;
14 *             }
15 *         }
16 *     }
17 * } // TC: O(m+n), SC: O(1)

```

#### 43. 3 Sum (LeetCode 15) [Asked by Facebook]

```

2
3 class Solution {
4     public List<List<Integer>> threeSum(int[] nums) {
5         Arrays.sort(nums);
6         List<List<Integer>> triplets = new ArrayList<>();
7         for (int i = 0; i <= nums.length - 3; i++) {
8             if (i == 0 || nums[i] != nums[i - 1]) {
9                 int left = i + 1, right = nums.length - 1;
10                int target = 0 - nums[i];
11                while (left < right) {
12                    if (nums[left] + nums[right] == target) {
13                        List<Integer> triplet = new ArrayList<>();
14                        triplet.add(nums[i]);
15                        triplet.add(nums[left]);
16                        triplet.add(nums[right]);
17                        triplets.add(triplet);
18
19                        while (left < nums.length - 1 && nums[left] == nums[left + 1]) left++;
20                        while (right > 0 && nums[right] == nums[right - 1]) right--;
21                        left++;
22                        right--;
23                    } else if (nums[left] + nums[right] < target) {
24                        left++;
25                    } else {
26                        right--;
27                    }
28                }
29            }
30        }
31        return triplets;
32    }
33 } // TC: O(n log n) + O(n^2), SC: O(1)

```

#### 44. Valid Parentheses (LeetCode 20) [Facebook]

**Success** Details +

Runtime 3 ms, faster than 47.94% of Java online submissions for Valid Parentheses.

Memory Usage 41.8 MB, less than 62.20% of Java online submissions for Valid Parentheses.

Next challenges:

Remove Invalid Parentheses | Check if String is Valid After Substring Removal

Show off your acceptance:

Time Submitted	Status	Runtime	Memory
05/05/2022 22:23	Accepted	3 ms	41.8 MB
05/05/2022 21:47	Accepted	2 ms	40.1 MB
05/05/2022 21:46	Wrong Answer	N/A	N/A
05/05/2022 21:46	Compile Error	N/A	N/A

```

1 class Solution {
2     public boolean isValid(String s) {
3         HashMap<Character, Character> map = new HashMap<>();
4         map.put('(', ')');
5         map.put('[', ']');
6         map.put('{', '}');
7         Stack<Character> stack = new Stack<>();
8         for (int i = 0; i < s.length(); i++) {
9             char curr = s.charAt(i);
10            if (map.containsKey(curr)) {
11                char pop = stack.pop();
12                if (pop != map.get(curr)) {
13                    return false;
14                }
15            } else {
16                stack.push(curr);
17            }
18        }
19        return stack.isEmpty();
20    }
21 } // TC: O(n), SC: O(n)

```

#### 45. Asteroid Collisions (LeetCode 735) [Asked by Lyft]

```

1 class Solution {
2     public int[] asteroidCollision(int[] asteroids) {
3         Stack<Integer> st = new Stack<>();
4         for (int asteroid : asteroids) {
5             if (asteroid > 0) {
6                 st.push(asteroid);
7             } else {
8                 while (!st.isEmpty() && st.peek() > 0 && Math.abs(asteroid) > st.peek()) {
9                     st.pop();
10                }
11                if (st.isEmpty() || st.peek() < 0) {
12                    st.push(asteroid);
13                } else if (st.peek() + asteroid == 0) {
14                    st.pop();
15                }
16            }
17        }
18        int[] ans = new int[st.size()];
19        for (int i = ans.length - 1; i >= 0; i--) {
20            ans[i] = st.pop();
21        }
22        return ans;
23    }
24 } // TC: O(n), SC: O(n)

```

#### 46. Longest Valid Parentheses (LeetCode 32) [Asked by Google]



```

1 * class Solution {
2 *     public int longestValidParentheses(String s) {
3         int ans = 0;
4         Stack<Integer> stack = new Stack<>();
5         stack.push(-1);
6         for (int i = 0; i < s.length(); i++) {
7             char ch = s.charAt(i);
8             if (ch == '(') {
9                 stack.push(i);
10            } else {
11                stack.pop();
12                if (stack.size() == 0) {
13                    stack.push(i);
14                } else {
15                    ans = Math.max(ans, i - stack.peek());
16                }
17            }
18        }
19        return ans;
20    }
21 } // TC: O(n), SC: O(n) I

```

#### 47. Decode String (LeetCode 394) [Asked by Google]

```

3     Stack<Integer> numStack = new Stack<>();
4     Stack<String> strStack = new Stack<>();
5     StringBuilder sb = new StringBuilder();
6     int len = s.length();
7     for (int i = 0; i < len; i++) {
8         char ch = s.charAt(i);
9         if (Character.isDigit(ch)) {
10             int num = ch - '0';
11             while (i + 1 < len && Character.isDigit(s.charAt(i+1))) {
12                 num = num * 10 + s.charAt(i+1) - '0';
13                 i++;
14             }
15             numStack.push(num);
16         } else if (ch == '[') {
17             strStack.push(sb.toString());
18             sb = new StringBuilder();
19         } else if (ch == ']') {
20             int k = numStack.pop();
21             StringBuilder tmp = new StringBuilder(strStack.pop());
22             for (int j = 0; j < k; j++) { // k times
23                 tmp.append(sb);
24             }
25             sb = tmp;
26         } else {
27             sb.append(ch);
28         }
29     }
30     return sb.toString();
31 }
32 } // TC: O(n * k), SC: O(n)

```

#### 48. Largest Rectangle in Histogram (LeetCode 84) [Asked by Google]

```

class Solution {
    public int largestRectangleArea(int[] heights) {
        int maxArea = 0;
        Stack<Integer> stack = new Stack<>();
        stack.push(0);
        for (int i = 1; i <= heights.length; i++) {
            int curr = (i == heights.length) ? -1 : heights[i];
            while (!stack.isEmpty() && curr <= heights[stack.peek()]) {
                int height = heights[stack.pop()];
                int width = stack.isEmpty() ? i : i - stack.peek() - 1;
                int currArea = height * width;
                maxArea = Math.max(maxArea, height * width);
            }
            stack.push(i);
        }
        return maxArea;
    }
}
// TC: O(n), SC: O(n)

```

#### 49. Kaden's Algorithm (Maximum Subarray :: LeetCode 53) [Asked by Google]

```

2
3 class Solution {
4     public int maxSubArray(int[] nums) {
5         int curr = nums[0];
6         int max = nums[0];
7         for (int i = 1; i < nums.length; i++) {
8             curr = Math.max(nums[i], nums[i] + curr);
9             max = Math.max(max, curr);
10        }
11        return max;
12    }
13 } // TC: O(n), SC: O(1)

```

50. Boyer-Moore Voting Algorithm (Majority Element- LeetCode 169) [Asked by Amazon]

51. Merge Sort Algorithm [Asked by Amazon, Microsoft]

52. Count Inversions -- Using Merge Sort [Asked by Amazon, Microsoft]

53. Quick Sort Algorithm [Asked by Amazon, Microsoft]

The screenshot displays a LeetCode submission interface. On the left, the 'Output Window' shows a success message: 'Problem Solved Successfully'. Below this, statistics are listed: 'Test Cases Passed: 67/87', 'Your Total Score: 425', 'Total Time Taken: 0.74', 'Correct Submission Count: 3', and 'Attempts No.: 3'. On the right, the code editor shows the Java implementation of the Quick Sort algorithm. The code includes a 'class Solution' with a 'quickSort' method that recursively sorts an array by partitioning it around a pivot. The partitioning logic is detailed, including the selection of a pivot and the swapping of elements to maintain order.

54. Kth Largest/Smallest -- Quick Select (LeetCode 215) [Asked by Google, Microsoft]

55. [OLD] Quick Sort :: Sort an Array (LeetCode 912)

56. Set Matrix Zeroes (LeetCode 73) [Asked by Facebook]

57. Range Sum Query 2D - Immutable (LeetCode 304) [Asked by Amazon]

```

1 class NumMatrix {
2
3     int[][] dp;
4     public NumMatrix(int[][] matrix) {
5         dp = new int[matrix.length + 1][matrix[0].length + 1];
6         for (int r = 0; r < matrix.length; r++) {
7             for (int c = 0; c < matrix[0].length; c++) {
8                 dp[r + 1][c + 1] = dp[r][c + 1] + dp[r + 1][c] + matrix[r][c] - dp[r][c];
9             }
10        }
11    } // TC: O(m*n), SC: O(m*n)
12
13    public int sumRegion(int row1, int col1, int row2, int col2) {
14        return dp[row2 + 1][col2 + 1] - dp[row1][col2 + 1] - dp[row2 + 1][col1] + dp[row1][col1];
15    } // TC: O(1), SC: O(1)
16 }
17
18 /**
19  * Your NumMatrix object will be instantiated and called as such:
20  * NumMatrix obj = new NumMatrix(matrix);
21  * int param_1 = obj.sumRegion(row1,col1,row2,col2);
22  */

```

58. Where Will the Ball Fall (LeetCode 1706) [Asked by Amazon]

```

1 class Solution {
2     public int[] findBall(int[][] grid) {
3         int[] result = new int[grid[0].length];
4         Arrays.fill(result, -1);
5         for (int col = 0; col < grid[0].length; col++) {
6             int currCol = col;
7             for (int currRow = 0; currRow < grid.length; currRow++) {
8                 int nextCol = currCol + grid[currRow][currCol];
9                 if (nextCol < 0 || nextCol > grid[0].length - 1) break;
10                if (grid[currRow][currCol] == grid[currRow][nextCol]) break;
11                if (currRow == grid.length - 1) {
12                    result[col] = nextCol;
13                }
14                currCol = nextCol;
15            }
16        }
17        return result;
18    } // TC: O(M), SC: O(1)
19 }

```

59. Rotate Image (LeetCode 48) [Asked by Microsoft]

60. Middle of the Linked List

```

1 // 60. Middle of the Linked List
2
3 class Solution {
4     public ListNode middleNode(ListNode head) {
5         // base case -- if linked list contains only one node
6         if (head.next == null) return head;
7         ListNode slow = head, fast = head;
8         while (fast != null && fast.next != null) {
9             slow = slow.next;
10            fast = fast.next.next;
11        }
12        return slow;
13    }
14 }
15 // TC: O(n), SC: O(1)

```

60. Delete Node in a Linked List (LeetCode 237) [Asked by Amazon, Microsoft]

```

7 * }F
8 */
9 class Solution {
10 public void deleteNode(ListNode node) {
11     node.val = node.next.val;
12     node.next = node.next.next;
13 }
14 }

```

## 61. Merge Two Sorted Lists (LeetCode 21) [Asked by Microsoft]

```

1 * class Solution {
2 * public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
3 *     ListNode newHead = new ListNode(-1);
4 *     ListNode p1 = list1, p2 = list2, curr = newHead;
5 *     while (p1 != null && p2 != null) {
6 *         if (p1.val < p2.val) {
7 *             curr.next = p1;
8 *             curr = curr.next;
9 *             p1 = p1.next;
10 *         } else {
11 *             curr.next = p2;
12 *             curr = curr.next;
13 *             p2 = p2.next;
14 *         }
15 *     }
16 *     if (p1 == null) curr.next = p2;
17 *     if (p2 == null) curr.next = p1;
18 *     return newHead.next;
19 * }
20 * } // TC: O(m + n), SC: O(1)

```

## 62. Reverse Linked List ::4 Ways:: (LeetCode 206) [Asked by Google, Facebook]

## 63. Reverse Nodes in k-group (LeetCode 25) [Asked by Microsoft]

Success Details

Runtime: 0 ms, faster than 100.00% of Java Nodes in k-Group.

Memory Usage: 41.5 MB, less than 97.59% Reverse Nodes in k-Group.

Next challenges:

Reverse Nodes in a Linked List

Reverse Nodes in k-Group

Show off your acceptance:

Time Submitted	Status	Run
05/10/2022 08:15	Accepted	0 m
05/09/2022 22:29	Accepted	1 m
05/09/2022 21:51	Accepted	0 m
12/27/2021 01:28	Accepted	0 m

```

1 * class Solution {
2 * public ListNode reverseKGroup(ListNode head, int k) {
3 *     if (k == 1) return head;
4 *     int count = k;
5 *     ListNode curr = head;
6 *     while (curr != null && count > 0) {
7 *         count--;
8 *         curr = curr.next;
9 *     }
10 *     if (count > 0) return head;
11 *     ListNode prev = reverseKGroup(curr, k);
12 *     ListNode current = head;
13 *     for (int i = 0; i < k; i++) { // k times
14 *         ListNode next = current.next;
15 *         current.next = prev;
16 *         prev = current;
17 *         current = next;
18 *     }
19 *     return prev;
20 * }
21 * } // TC: O(n*k), SC: O(n/k)

```

## 64. Remove Duplicates from Sorted List ::O(N):: (LeetCode 83) [Asked by Microsoft]

**Success** Details

Runtime: 3 ms, faster than 8.44% of Java online submissions for Remove Duplicates from Sorted List.

Memory Usage: 42.1 MB, less than 60.42% of Java online submissions for Remove Duplicates from Sorted List.

Next challenges:

Remove Duplicates from Sorted List II

Time Submitted	Status	Runtime	Memory
01/28/2023 12:06	Accepted	3 ms	42.1 MB
01/28/2023 12:06	Compile Error	N/A	N/A
01/28/2023 11:51	Accepted	3 ms	42 MB
01/28/2023 11:48	Wrong Answer	N/A	N/A
08/22/2021 07:02	Accepted	0 ms	38.1 MB

```

1 // Definition for singly-linked list.
2 public class ListNode {
3     int val;
4     ListNode next;
5     ListNode() {}
6     ListNode(int val) { this.val = val; }
7     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
8 }
9
10
11 class Solution {
12     public ListNode deleteDuplicates(ListNode head) {
13         if (head == null) return null;
14         ListNode curr = head;
15         Stack<ListNode> stack = new Stack<>();
16         while (curr != null) {
17             if (stack.isEmpty()) {
18                 stack.push(curr);
19             } else {
20                 if (stack.peek().val != curr.val) {
21                     stack.push(curr);
22                 }
23             }
24             curr = curr.next;
25         }
26         ListNode newHead = null;
27         while (!stack.isEmpty()) {
28             ListNode tmp = stack.pop();
29             tmp.next = newHead;
30             newHead = tmp;
31         }
32         return newHead;
33     }
34 } // TC: O(n), SC: O(n)

```

## 65. Remove Duplicates from Sorted List :: $O(1)$ :: (LeetCode 83) [Asked by Microsoft]

```

10
11 class Solution {
12     public ListNode deleteDuplicates(ListNode head) {
13         if (head == null) return null;
14         ListNode dummyHead = new ListNode(-1);
15         dummyHead.next = head;
16         ListNode curr = head, prev = dummyHead;
17         while (curr != null) {
18             if (curr.next != null && curr.val == curr.next.val) {
19                 while (curr.next != null && curr.val == curr.next.val) {
20                     curr = curr.next;
21                 }
22                 prev.next = curr;
23                 prev = curr;
24             } else {
25                 prev = curr;
26             }
27             curr = curr.next;
28         }
29         return dummyHead.next;
30     }
31 } // TC: O(n), SC: O(1)

```

## 66. Rotate List (LeetCode 61) [Asked by Microsoft]

```

1
2 public class Solution {
3     public ListNode rotateRight(ListNode head, int k) {
4         if (head == null) return null;
5         int size = 0;
6         ListNode curr = head;
7         while (curr.next != null) {
8             size++;
9             curr = curr.next;
10        }
11        size += 1;
12        curr.next = head; // tail -> head
13        k = k % size;
14        curr = head;
15        for (int i = 1; i < size - k; i++) {
16            curr = curr.next;
17        }
18        ListNode newHead = curr.next;
19        curr.next = null;
20        return newHead;
21    }
22 } // TC: O(n), SC: O(1)
23

```

## 67. Reorder List (LeetCode 143) [Asked by Amazon, LinkedIn, Microsoft]

## 68. Palindrome Linked List (LeetCode 234) [Asked by Microsoft]

## 69. Code (Resource)

## 70. Tree Traversal Techniques (Level Order, Zigzag, Preorder, Inorder, Postorder)

## 71. Level Order Traversal Binary Tree (LeetCode 102) [Asked by Amazon, Microsoft]

```
14 *  
15 */  
16 class Solution {  
17     public List<List<Integer>> levelOrder(TreeNode root) {  
18         List<List<Integer>> traversal = new ArrayList<>();  
19         if (root == null) return traversal;  
20         Queue<TreeNode> queue = new LinkedList<>();  
21         queue.add(root);  
22         while (!queue.isEmpty()) {  
23             int size = queue.size();  
24             List<Integer> currLevel = new ArrayList<>();  
25             while (size != 0) {  
26                 TreeNode tmp = queue.poll();  
27                 currLevel.add(tmp.val);  
28                 if (tmp.left != null) queue.add(tmp.left);  
29                 if (tmp.right != null) queue.add(tmp.right);  
30                 size--;  
31             }  
32             traversal.add(currLevel);  
33         }  
34         return traversal;  
35     }  
36 } // TC: O(n), SC: O(n)
```

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [3,9,20,null,null,15,7]

## 72. Binary Tree Zigzag Level Order Traversal (LeetCode 103) [Asked by Microsoft]

```
15 *  
16 */  
17 class Solution {  
18     public List<List<Integer>> zigzagLevelOrder(TreeNode root) {  
19         List<List<Integer>> traversal = new ArrayList<>();  
20         if (root == null) return traversal;  
21         Queue<TreeNode> queue = new LinkedList<>();  
22         queue.add(root);  
23         int level = 0;  
24         while (!queue.isEmpty()) {  
25             int size = queue.size();  
26             List<Integer> currLevel = new ArrayList<>();  
27             while (size != 0) {  
28                 TreeNode tmp = queue.poll();  
29                 if (level % 2 == 0) currLevel.add(tmp.val);  
30                 else if (level % 2 == 1) currLevel.add(0, tmp.val);  
31                 if (tmp.left != null) queue.add(tmp.left);  
32                 if (tmp.right != null) queue.add(tmp.right);  
33                 size--;  
34             }  
35             level++;  
36             traversal.add(currLevel);  
37         }  
38         return traversal;  
39 } // TC: O(n), SC: O(n)
```

## 73. Binary Tree Preorder Traversal (LeetCode 144) [Asked by Microsoft]

```
14 *  
15 */  
16 class Solution {  
17     public List<Integer> preorderTraversal(TreeNode root) {  
18         List<Integer> preorder = new ArrayList<>();  
19         if (root == null) return preorder;  
20         Stack<TreeNode> stack = new Stack<>();  
21         stack.push(root);  
22         TreeNode curr = root;  
23         while (!stack.isEmpty()) {  
24             curr = stack.pop();  
25             while (curr != null) {  
26                 preorder.add(curr.val);  
27                 if (curr.right != null)  
28                     stack.push(curr.right);  
29                 curr = curr.left;  
30             }  
31         }  
32         return preorder;  
33     }  
34 }  
35 // TC: O(n), SC: O(n)  
36  
37 --
```

Console

#### 74. Binary Tree Inorder Traversal (LeetCode 94) [Asked by Microsoft]

```
17 public List<Integer> inorderTraversal(TreeNode root) {
18     List<Integer> inorder = new ArrayList<>();
19     if (root == null) return inorder;
20     Stack<TreeNode> stack = new Stack<>();
21     TreeNode curr = root;
22     while (!stack.isEmpty() || curr != null) {
23         while (curr != null) {
24             stack.push(curr);
25             curr = curr.left;
26         }
27         // L, Root, R
28         curr = stack.pop();
29         inorder.add(curr);
30         curr = curr.right;
31     }
32     return inorder;
33 }
34 // TC: O(n), SC: O(n)
```

It's through an error.

Hint: `inorder.add(curr.val)`

#### 75. Binary Tree Post Order Traversal (LeetCode 145) [Asked by Microsoft]

```
16 class Solution {
17     public List<Integer> postorderTraversal(TreeNode root) {
18         List<Integer> postorder = new ArrayList<>();
19         if (root == null) return postorder;
20         Stack<TreeNode> stack = new Stack<>();
21         stack.push(root);
22         TreeNode curr = root;
23         while (!stack.isEmpty()) {
24             curr = stack.pop();
25             // Ro, R, L — reverse preorder traversal
26             while (curr != null) {
27                 postorder.add(curr.val);
28                 if (curr.left != null)
29                     stack.push(curr.left);
30                 curr = curr.right;
31             }
32             // L, R, Ro
33             Collections.reverse(postorder);
34             return postorder;
35         }
36     }
37 }
38 // TC: O(n), SC: O(n)
```

#### 76. [Morris Traversal] Preorder Traversal (LeetCode 144) [Asked by Microsoft]

#### 77. [Morris Traversal] Inorder Traversal (LeetCode 94) [Asked by Microsoft]

```
17 public List<Integer> inorderTraversal(TreeNode root) {
18     List<Integer> inorder = new ArrayList<>();
19     if (root == null) return inorder;
20     TreeNode curr = root;
21     while (curr != null) {
22         if (curr.left == null) { // L, Ro, R
23             inorder.add(curr.val);
24             curr = curr.right;
25         } else {
26             TreeNode predecessor = curr.left; // rightMostNode
27             while (predecessor.right != null && predecessor.right != curr) {
28                 predecessor = predecessor.right;
29             }
30             if (predecessor.right == null) { // we have to create a thread
31                 predecessor.right = curr;
32                 curr = curr.left;
33             } else { // we have a thread
34                 predecessor.right = null;
35                 inorder.add(curr.val);
36                 curr = curr.right;
37             }
38         }
39     }
40     return inorder;
41 }
42 // TC: O(n), SC: O(1)
```



## 78. Minimum Depth of Binary Tree (LeetCode 111) [Asked by Meta, Amazon, Microsoft]

```
11 * this.left = left;
12 * this.right = right;
13 * }
14 * }
15 */
16 class Solution {
17     public int minDepth(TreeNode root) {
18         if (root == null) return 0;
19         return dfs(root);
20     }
21     public int dfs(TreeNode root) {
22         if (root == null) return Integer.MAX_VALUE;
23         if (root.left == null && root.right == null) return 1;
24         int left = dfs(root.left);
25         int right = dfs(root.right);
26         return 1 + Math.min(left, right);
27     }
28 }
29 // TC: O(n), SC: O(n) | I
30
31
32
33
```

## 79. Maximum Depth of Binary Tree (LeetCode 104) [Asked by Meta, Microsoft, Amazon]

```
12 * this.right = right;
13 * }
14 * }
15 */
16 class Solution {
17     public int maxDepth(TreeNode root) {
18         if (root == null) return 0;
19         return dfs(root);
20     }
21     public int dfs(TreeNode root) {
22         if (root == null) return Integer.MIN_VALUE; // invalid path
23         if (root.left == null && root.right == null) return 1; // leaf node
24         int left = dfs(root.left);
25         int right = dfs(root.right);
26         return 1 + Math.max(left, right);
27     }
28 }
29 // TC: O(n), SC: O(n)
30
31
32
33
34
```

## 80. Diameter of Binary Tree (LeetCode 543) [Asked by Microsoft, Amazon]

Gfg:

```
109 Node left;
110 Node right;
111 Node(int data) {
112     this.data = data;
113     left = null;
114     right = null;
115 }
116
117
118 class Solution { // diameter: num nodes in the longest path
119     int DIAMETER = 0;
120     int diameter(Node root) {
121         if (root == null) return DIAMETER;
122         helper(root);
123         return DIAMETER;
124     }
125     int helper(Node root) {
126         // base case
127         if (root == null) return 0;
128         // recursive case
129         int left = helper(root.left);
130         int right = helper(root.right);
131         DIAMETER = Math.max(DIAMETER, 1 + left + right);
132         return 1 + Math.max(left, right);
133     }
134 }
135
136 // TC: O(n), SC: O(h)
137
138
139
140
141
142
143
```

Leetcode:

Accepted

Runtime: 0 ms  
Beats 100.00% of users with Java

Next question: 61. Rotate List

More challenges: 1522. Diameter of N-Ary Tree, 2246. Longest

Status: Accepted, Language: Java, Runtime: 0 ms, Memory: 41.1 MB

```

14 + }
15 +/
16 class Solution {
17     int DIAMETER = 0;
18     public int diameterOfBinaryTree(TreeNode root) {
19         if (root == null) return DIAMETER;
20         helper(root);
21         return DIAMETER;
22     }
23     int helper(TreeNode root) {
24         // base case
25         if (root == null) return 0;
26         // recursive case
27         int left = helper(root.left);
28         int right = helper(root.right);
29         DIAMETER = Math.max(DIAMETER, left + right);
30         return 1 + Math.max(left, right);
31     }
32 }
33 // TC: O(n), SC: O(n)
34
35
36
--INSERT--

```

### 81. Flatten Binary Tree to Linked List (LeetCode 114) [Asked by Microsoft]

Accepted

Runtime: 0 ms  
Beats 100.00% of users with Java

Next question: 1. Two Sum

More challenges: 430. Flatten a Multilevel Doubly Linked List, 1660. Correct

Status: Accepted, Language: Java, Runtime: 0 ms, Memory: 40.9 MB

```

14 + }
15 +/
16 class Solution {
17     public void flatten(TreeNode root) {
18         TreeNode curr = root;
19         while (curr != null) {
20             if (curr.left == null) {
21                 curr = curr.right;
22             } else {
23                 TreeNode rightMostChild = curr.left;
24                 while (rightMostChild.right != null) {
25                     rightMostChild = rightMostChild.right;
26                 }
27                 rightMostChild.right = curr.right;
28                 curr.right = curr.left;
29                 curr.left = null;
30                 curr = curr.right;
31             }
32         }
33     }
34 }
35 // TC: O(n), SC: O(1)
36
37
38
39
40
41
42
--INSERT--

```

### 82. Lowest Common Ancestor of Binary Tree (LeetCode 236) [Asked by Google, Microsoft]

Accepted

Runtime: 0 ms  
Beats 100.00% of users with Java

Next question: 1. Two Sum

More challenges: 430. Flatten a Multilevel Doubly Linked List, 1660. Correct

Status: Accepted, Language: Java, Runtime: 0 ms, Memory: 40.9 MB

```

10 class Solution {
11     public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
12         if (root == p || root == q) return root;
13         return dfs(root, p, q);
14     }
15     public TreeNode dfs(TreeNode root, TreeNode p, TreeNode q) {
16         // base case
17         if (root == null) return null;
18         if (root == p || root == q) return root;
19         if (root.left == null && root.right == null) return null;
20         // recursive case
21         TreeNode leftLCA = dfs(root.left, p, q);
22         TreeNode rightLCA = dfs(root.right, p, q);
23         // if (leftLCA == null && rightLCA != null) {
24         //     return rightLCA;
25         // }
26         // if (leftLCA != null && rightLCA == null) {
27         //     return leftLCA;
28         // }
29         // if (leftLCA == null && rightLCA == null) {
30         //     return null;
31         // }
32         if (leftLCA == null) return rightLCA;
33         if (rightLCA == null) return leftLCA;
34         return root;
35     }
36 }
37 // TC: O(n), SC: O(n)
38
39
40
--INSERT--

```

### 83. Populating Next Right Pointers in Each Node (LeetCode 116) [Asked by Meta]

```

24 class Solution {
25     public Node connect(Node root) {
26         if (root == null) return null;
27         Node levelStart = root;
28         while (levelStart != null) {
29             Node curr = levelStart;
30             while (curr != null) {
31                 if (curr.left != null) {
32                     curr.left.next = curr.right;
33                     if (curr.right != null) {
34                         curr.right.next = curr.next.left;
35                     }
36                 }
37                 curr = curr.next;
38             }
39             levelStart = levelStart.left;
40         }
41         return root;
42     }
43 }
44 // TC: O(n), SC: O(1)
45
--INSERT--

```

84. Populating Next Right Pointers in Each Node II (LeetCode 117) [Asked by Meta]

```

22 //
23
24 class Solution {
25     public Node connect(Node root) {
26         if (root == null) return null;
27         Node levelStart = root;
28         while (levelStart != null) {
29             Node curr = levelStart; // travel through current level
30             Node dummy = new Node(0); // to keep track the first node in the next level
31             Node prev = dummy; // to connect node in the next level
32             while (curr != null) {
33                 if (curr.left != null) {
34                     prev.next = curr.left;
35                     prev = prev.next;
36                 }
37                 if (curr.right != null) {
38                     prev.next = curr.right;
39                     prev = prev.next;
40                 }
41                 curr = curr.next;
42             }
43             levelStart = dummy.next;
44             dummy.next = null;
45         }
46         return root;
47     }
48 }
49 // TC: O(n), SC: O(1)
50
--INSERT--

```

85. Same Tree (LeetCode 100) [Asked by Amazon, Microsoft]

```

1 class Solution {
2     public boolean isSameTree(TreeNode p, TreeNode q) {
3         return dfs(p, q);
4     }
5     public boolean dfs(TreeNode root1, TreeNode root2) {
6         if (root1 == null && root2 == null) return true;
7         if (root1 == null || root2 == null) return false;
8         if (root1.val != root2.val) return false;
9         boolean left = dfs(root1.left, root2.left);
10        boolean right = dfs(root1.right, root2.right);
11        return left && right;
12    }
13 } // TC: O(n), SC: O(n)

```

86. Flip/Invert Binary Tree (LeetCode 226) [Asked Google, Amazon]

87. Inorder Successor of BST (LeetCode 285) [Asked by Microsoft/Amazon]

Problems Courses Jobs Events POTD

Output Window

Compilation Results Custom Input

Problem Solved Successfully

You get marks only for the first correct submission if you solve the problem without viewing the full solution.

Test Cases Passed: 1163 / 1185

Your Total Score: 754

Total Time Taken: 0.97

Correct Submission Count: 6

Attempts No: 9

Next Suggested Problem(s):

- Kth nodes from end of linked list
- Sum of leaf nodes in BST
- Children Sum Parent

Geek Tip:

- Post your approach and explanations in the comments section.
- Go through the problem editorial and solutions submitted by others to know about alternate approaches to this problem. Keep coding!

```

120 Node(int x){
121     data=x;
122     left=null;
123 }
124
125
126 class Solution {
127     public Node insertSuccessor(Node root, Node x) {
128         Node successor = null;
129         Node curr = root;
130         while (curr != null) {
131             if (curr.data > x.data) {
132                 successor = curr;
133                 curr = curr.left;
134             } else if (curr.data == x.data) {
135                 curr = curr.right;
136             }
137         }
138         return successor;
139     }
140 }
141 // TC: O(h), SC: O(1)
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159

```

88. Insert into a Binary Search Tree (LeetCode 701) [Asked by Amazon, Microsoft]

Problem List

701. Insert into a Binary Search Tree

Medium

You are given the root node of a binary search tree (BST) and a value to insert into the tree. Return the root node of the BST after the insertion. It is guaranteed that the new value does not exist in the original BST. Notice that there may exist multiple valid ways for the insertion, as long as the tree remains a BST insertion. You can return any of them.

Example 1:

Input: root = [4,2,7,1,3], val = 5  
Output: [4,2,7,1,3,5]  
Explanation: Another accepted tree

```

1 //
2
3 class Solution {
4     public Node insertIntoBST(Node root, int val) {
5         // base case
6         if (root == null) return new Node(val);
7         // recursive case
8         if (root.val < val) {
9             root.left = insertIntoBST(root.left, val);
10        } else if (root.val > val) {
11            root.right = insertIntoBST(root.right, val);
12        }
13        return root;
14    }
15 }
16
17

```

89. Sorted to Binary Search Tree (LeetCode 108) [Asked by Microsoft, Meta]

90. Construct BST from Preorder Traversal (LeetCode 1008) [Asked by Amazon]

91. Kth Largest Element in an Array (LeetCode 215) [Asked by Microsoft, Amazon]

```

2
3 class Solution {
4     public int findKthLargest(int[] nums, int k) {
5         PriorityQueue<Integer> minHeap = new PriorityQueue<>();
6         for (int i = 0; i < k; i++) {
7             minHeap.add(nums[i]);
8         }
9         for (int i = k; i < nums.length; i++) {
10            if (minHeap.peek() < nums[i]) {
11                minHeap.poll();
12                minHeap.add(nums[i]);
13            }
14        }
15        return minHeap.peek();
16    }
17 } // TC: O(k + (n-k)*log(k)), SC: O(k)

```

## 92. Find Median in a Data Stream

**SUCCESS** Details >

Runtime: **169 ms**, faster than **26.36%** of Java online submissions for Find Median from Data Stream.

Memory Usage: **62.2 MB**, less than **52.30%** of Java online submissions for Find Median from Data Stream.

Next challenges:

- Find Window Median
- Find NK Average
- Sequentially Optimal Rank Tracker

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
03/16/2023 11:31	Accepted	169 ms	62.2 MB	java
03/16/2023 09:38	Accepted	170 ms	61.7 MB	java
07/21/2022 00:02	Accepted	318 ms	121.6 MB	java
07/07/2022 16:51	Accepted	289 ms	120.7 MB	java
07/07/2022 15:53	Accepted	217 ms	124.8 MB	java

```

1
2
3 class MedianFinder {
4
5     PriorityQueue<Integer> minheap;
6     PriorityQueue<Integer> maxheap;
7
8     public MedianFinder() {
9         minheap = new PriorityQueue<>(a, b) -> b - a);
10        maxheap = new PriorityQueue<>(a, b) -> a - b);
11    }
12
13    public void addNum(int num) {
14        minheap.add(num);
15        minheap.size() <= maxheap.size() ?
16            if (minheap.size() < maxheap.size()) {
17                minheap.offer(maxheap.poll());
18            }
19            // TC: O(log n), SC: O(log n)
20    }
21
22    public double findMedian() {
23        if ((minheap.size() + maxheap.size()) % 2 == 1) return minheap.peek();
24        return (minheap.peek() + maxheap.peek()) / 2.0;
25    }
26    // TC: O(1), SC: O(1)
27 }
28 /**
29  *
30  */
31 }
```

Testcase Run Code Result Debugger

Runtime: 2 ms

Accepted

Your input  
[{"num":1,"val":1}, {"num":2,"val":1}, {"num":3,"val":1}]

Output  
[null,null,null,1.50000,null,1.66667]

Expected  
[null,null,null,1.50000,null,1.66667]

### 93. Implement Trie (LeetCode 208) :: HashMap :: [Asked by Amazon, Microsoft]

#### 94. String Permutation - (Non Duplicates) [Asked by Google]

### 95. String Permutation - (Duplicates) [Asked by Google]

**Problems** Courses Get Hint Comments UPVOTED 🔍 🏠 ⚙️

---

⌵ Problem
✎ Editorial
📄 Submissions
💬 Comments

### Permutations of a given string 🔒

Medium Accuracy: 49.85% Submissions: 71019 Points: 4

You are given a string S. The task is to print all unique permutations of the given string in lexicographically sorted order.

**Example 1:**

```
Input: ABC
Output:
ABC ACB BAC BCA CAB CBA
Explanation:
Given string ABC has permutations in 6 ways, which are:
```

**Output Window**

Compilation Results Custom Input

**Problem Solved Successfully**

You get marks only for the first correct submission if you solve the problem without clearing the full solution.

Test Cases Passed: <b>5/5</b>	Your Total Score: <b>317</b>
Total Time Taken: <b>0.23</b>	Correct Submission Count:

```

1 // Permutation Using Recursion
2
3 class Solution {
4 public:
5     void find_permutation(string s) {
6         if(s.length() == 0) {
7             return;
8         }
9         sort(s.begin(), s.end());
10        int left = 0, right = s.length() - 1;
11        while(left < right) swap(s[left], s[right]);
12        if(s[0] != s[s.length()-1]) {
13            String temp = s.substr(0, s.length()-1);
14            find_permutation(temp);
15        } else {
16            return;
17        }
18    }
19 };
20
21 #include<iostream>
22 using namespace std;
23
24 int main() {
25     string s;
26     cin >> s;
27     Solution obj;
28     obj.find_permutation(s);
29 }
```

[Solution](#) [Report Error](#) [Feedback](#)

### 96. Permutations -- Array (LeetCode 46) [Asked by Google]

```

1 * class Solution {
2 *     public static void dfs(int[] nums, List<List<Integer>> list, List<Integer> perm) {
3 *         if (perm.size() == nums.length) {
4 *             list.add(new ArrayList<>(perm));
5 *         }
6 *         for (int i = 0; i < nums.length; i++) {
7 *             if (perm.contains(nums[i])) continue;
8 *             perm.add(nums[i]);
9 *             dfs(nums, list, perm);
10 *             perm.remove(perm.size() - 1);
11 *         }
12 *     }
13 *     public List<List<Integer>> permute(int[] nums) {
14 *         Arrays.sort(nums);
15 *         List<List<Integer>> list = new ArrayList<>();
16 *         dfs(nums, list, new ArrayList<>());
17 *         return list;
18 *     }
19 * } // TC:  $O(n * \log n) + O(n * n!)$ , SC:  $O(n)$ 

```

### 97. Combinations (LeetCode 77) [Asked by Google/Facebook/Amazon]

```
1 class Solution {
2     public static void dfs(List<List<Integer>> list, List<Integer> comb, int n, int k, int start) {
3         if (comb.size() == k) {
4             list.add(new ArrayList<>(comb));
5             return;
6         }
7         for (int i = start; i <= n; i++) {
8             comb.add(i);
9             dfs(list, comb, n, k, i + 1);
10            comb.remove(comb.size() - 1);
11        }
12    }
13    public List<List<Integer>> combine(int n, int k) {
14        List<List<Integer>> list = new ArrayList<>();
15        dfs(list, new ArrayList<>(), n, k, 1);
16        return list;
17    }
18 }
```

### 98. Combination Sum (LeetCode 39) [Asked by Google]

```
1 class Solution {
2     public static void dfs(int[] nums, List<List<Integer>> list, List<Integer> comb, int target, int sum, int start) {
3         if (sum == target) {
4             list.add(new ArrayList<>(comb));
5             return;
6         } else if (sum > target) return;
7
8         for (int i = start; i < nums.length; i++) {
9             comb.add(nums[i]);
10            dfs(nums, list, comb, target, sum + nums[i], i);
11            comb.remove(comb.size() - 1);
12        }
13    }
14    public List<List<Integer>> combinationSum(int[] candidates, int target) {
15        List<List<Integer>> list = new ArrayList<>();
16        dfs(candidates, list, new ArrayList<>(), target, 0, 0);
17        return list;
18    }
19 }
```

### 99. Generate Parentheses (LeetCode 22) [Asked Facebook, Microsoft]

Success: Details

Runtime: 3 ms, faster than 54.81% of Java for Generate Parentheses.

Memory Usage: 44.2 MB, less than 21.32% for Generate Parentheses.

Next challenges:

Check if a Parentheses String Can Be Valid

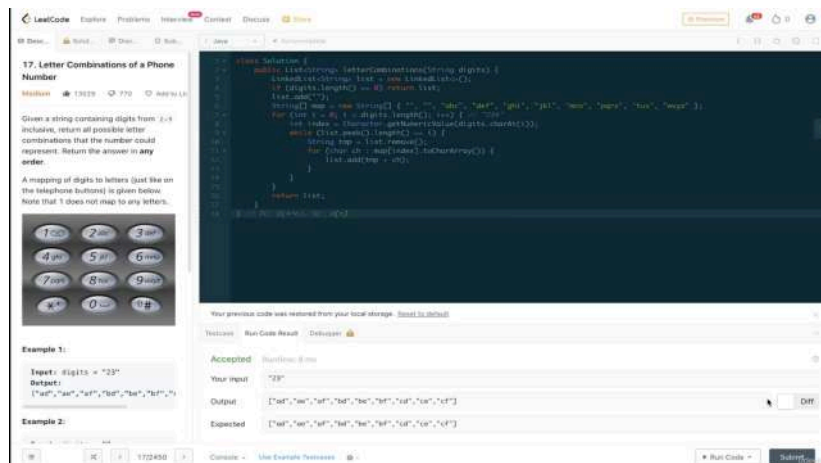
Show off your acceptance:

Time Submitted	Status	Runtime
10/29/2022 16:50	Accepted	3 ms
10/29/2022 16:46	Accepted	1 ms
10/29/2022 16:37	Accepted	3 ms
10/29/2022 15:50	Accepted	1 ms
10/29/2022 15:34	Accepted	1 ms
10/24/2022 00:59	Accepted	3 ms
07/29/2020 15:56	Accepted	1 ms

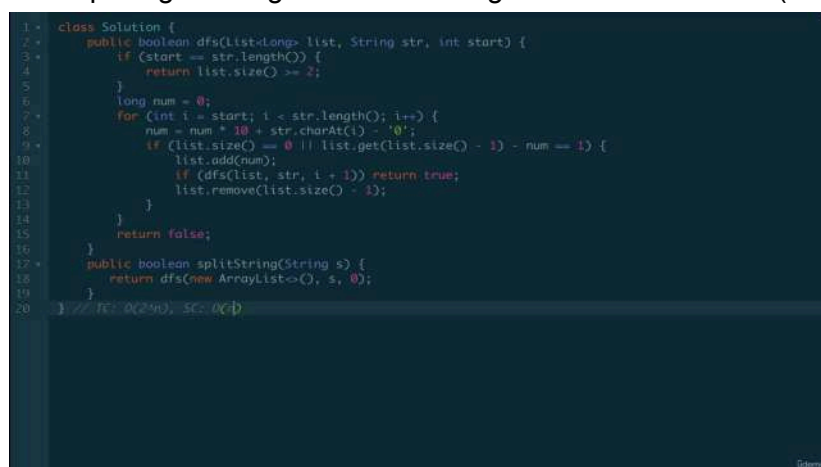
```
1 class Solution {
2     public static void dfs(List<String> list, int n, String str, int open, int close) {
3         if (str.length() == 2 * n) {
4             list.add(str);
5             return;
6         }
7         if (open < n) {
8             dfs(list, n, str + "(", open + 1, close);
9         }
10        if (close < open) {
11            dfs(list, n, str + ")", open, close + 1);
12        }
13    }
14    public List<String> generateParenthesis(int n) {
15        List<String> list = new ArrayList<>();
16        dfs(list, n, "", 0, 0);
17        return list;
18    }
19 }
```

### 100. Letter Combinations of a Phone Number (LeetCode 17) [Asked by Google]

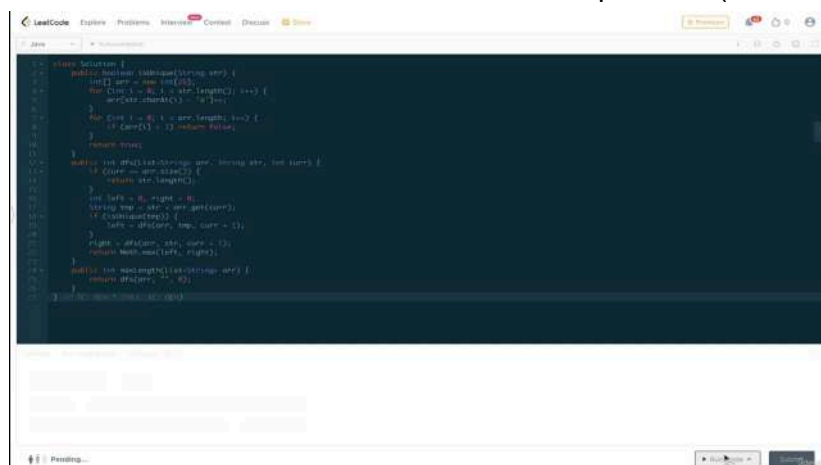




## 101. Splitting a String Into Descending Consecutive Values (1849) [Asked by Google]



## 102. Max Len of a Concatenated Str with Unique Char (LeetCode 1239) [Asked by Google]



## 103. Partition to K Equal Sum Subsets (LeetCode 698) [Asked by Microsoft]



LeetCode Explorer Problems Interview Contests Discuss Store

104. Rat in a Maze

```

class Solution {
    public boolean dfs(int[] name, boolean[] visited, int k, int targetSum, int currSum, int index) {
        if (k == 0) return true;
        if (targetSum == currSum) return true;
        if (dfs(name, visited, k - 1, targetSum, k, name.length - 1)) return true;
        for (int i = index; i < name.length - 1; i++) {
            if (visited[i]) continue;
            if (i < name.length - 1) {
                if (name[i] > 0) {
                    if (currSum + name[i] > targetSum) continue;
                    visited[i] = true;
                    if (dfs(name, visited, k, targetSum, currSum + name[i], i + 1)) return true;
                    visited[i] = false;
                }
            }
        }
        return false;
    }
    public boolean canPartitionSubarray(int[] name, int k) {
        int sum = 0;
        for (int num : name) sum += num;
        if (sum % k != 0) return false;
        int targetSum = sum / k;
        return dfs(name, new boolean[name.length], k, targetSum, 0, name.length - 1);
    }
}

```

Time Submitted: 10/30/2022 00:29 Status: Accepted

10/29/2022 23:26 Accepted

10/29/2022 22:31 Accepted

10/27/2022 08:24 Time Limit Exceeded

10/27/2022 01:03 Runtime Error

10/27/2022 01:01 Accepted

10/27/2022 01:02 Runtime Error

10/27/2022 00:58 Time Limit Exceeded

10/26/2022 12:42 Accepted

Accepted: Runtime: 0 ms

Your input: [1,3,3,1,5,2,1]

Output: true

Expected: true

## 104. Rat in a Maze [Asked by Amazon/Microsoft]

```

1 // C++ program to find all paths from
2 // (0, 0) to (n-1, m-1) in a maze
3
4 #include <iostream>
5 #include <vector>
6 #include <string>
7
8 using namespace std;
9
10 // DFS function to find all paths
11 // from (0, 0) to (n-1, m-1)
12 void dfs(int r, int c, int n, int m,
13         vector<string> &res,
14         vector<vector<bool>> &visited) {
15     // If we reach the destination
16     if (r == n - 1 && c == m - 1) {
17         res.push_back("");
18         return;
19     }
20     // Mark the current cell as visited
21     visited[r][c] = true;
22     // Try to move in four directions
23     if (r < n - 1 && !visited[r + 1][c])
24         dfs(r + 1, c, n, m, res, visited);
25     if (c < m - 1 && !visited[r][c + 1])
26         dfs(r, c + 1, n, m, res, visited);
27     if (r > 0 && !visited[r - 1][c])
28         dfs(r - 1, c, n, m, res, visited);
29     if (c > 0 && !visited[r][c - 1])
30         dfs(r, c - 1, n, m, res, visited);
31     // Unmark the current cell
32     visited[r][c] = false;
33 }
34
35 // Function to find all paths
36 vector<string> findPaths(int n, int m) {
37     vector<vector<bool>> visited(n, vector<bool>(m, false));
38     vector<string> res;
39     dfs(0, 0, n, m, res, visited);
40     return res;
41 }
42
43 // Driver code
44 int main() {
45     int n = 4, m = 4;
46     vector<string> res = findPaths(n, m);
47     for (string s : res)
48         cout << s << " ";
49     return 0;
50 }

```

## 105. M-Coloring [Asked by Amazon]

```

32
33 class solve {
34 public:
35     bool isSafe(vector<vector<int>> &graph, int color, int currColor, int r) {
36         for (int c = 0; c < graph[0].size(); c++) {
37             if (graph[r][c] == currColor) return false;
38         }
39         return true;
40     }
41     bool dfs(vector<vector<int>> &graph, int color, int m, int r) {
42         if (r == graph.size()) return true;
43         for (int i = 1; i <= m; i++) {
44             if (isSafe(graph, color, i, r)) {
45                 color[r] = i;
46                 if (dfs(graph, color, m, r + 1)) return true;
47                 color[r] = 0;
48             }
49         }
50         return false;
51     }
52     bool graphColoring(vector<vector<int>> &graph, int m, int n) {
53         int color[n];
54         return dfs(graph, color, m, 0);
55     }
56 }
57 // TC: O(n * m^n), SC: O(n)

```

## 106. BFS & DFS Traversal of Graph [Asked by Samsung]

## 107. Number of Islands [Asked by Microsoft, Apple] (LeetCode 200)

**Success** Details

Runtime: 12 ms, faster than 30.02% of Java submissions  
 Memory Usage: 57.5 MB, less than 39.5% of Java submissions

Next challenges:

Recommended Questions: [Number of Connected Components in a Undirected Graph](#), [Number of Connected Components in a Directed Graph](#), [Find All Groups of Farmland](#), [Count Unreachable Pairs of Nodes in a Directed Graph](#)

Show off your acceptance:

Time Submitted	Status	Run
11/09/2022 12:08	Accepted	1
03/19/2021 23:26	Accepted	1
03/14/2021 08:09	Accepted	2
08/11/2020 06:22	Accepted	1
08/06/2020 16:21	Compile Error	16

```

class Solution {
    public List<List<Integer>> findAllPaths(int[][] grid, int src, int tar) {
        if (src == tar) return List.of(List.of(src));
        List<List<Integer>> list = new ArrayList<>();
        Queue<List<Integer>> queue = new LinkedList<>();
        List<Integer> path = new ArrayList<>();
        path.add(src);
        queue.add(path);
        while (!queue.isEmpty()) {
            path = queue.poll();
            int last = path.get(path.size() - 1);
            if (last == tar) {
                list.add(path);
                continue;
            }
            for (Integer adj : grid[last]) {
                if (!path.contains(adj)) {
                    List<Integer> newPath = new ArrayList<>(path);
                    newPath.add(adj);
                    queue.add(newPath);
                }
            }
        }
        return list;
    }
}
  
```

Accepted Runtime: 0 ms

Your input: `[[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20],[21,22,23,24,25]]`

Output: `1`

Expected: `1`

## 108. All Paths From Source to Target (LeetCode 797) [Asked by Google, Bloomberg]

**Success** Details

Runtime: 12 ms, faster than 30.02% of Java submissions  
 Memory Usage: 57.5 MB, less than 39.5% of Java submissions

Next challenges:

Recommended Questions: [Number of Connected Components in a Undirected Graph](#), [Number of Connected Components in a Directed Graph](#), [Find All Groups of Farmland](#), [Count Unreachable Pairs of Nodes in a Directed Graph](#)

Show off your acceptance:

Time Submitted	Status	Run
11/09/2022 12:08	Accepted	1
03/19/2021 23:26	Accepted	1
03/14/2021 08:09	Accepted	2
08/11/2020 06:22	Accepted	1
08/06/2020 16:21	Compile Error	16

```

class Solution {
    public List<List<Integer>> findAllPaths(int[][] grid, int src, int tar) {
        if (src == tar) return List.of(List.of(src));
        List<List<Integer>> list = new ArrayList<>();
        Queue<List<Integer>> queue = new LinkedList<>();
        List<Integer> path = new ArrayList<>();
        path.add(src);
        queue.add(path);
        while (!queue.isEmpty()) {
            path = queue.poll();
            int last = path.get(path.size() - 1);
            if (last == tar) {
                list.add(path);
                continue;
            }
            for (Integer adj : grid[last]) {
                if (!path.contains(adj)) {
                    List<Integer> newPath = new ArrayList<>(path);
                    newPath.add(adj);
                    queue.add(newPath);
                }
            }
        }
        return list;
    }
}
  
```

Accepted Runtime: 0 ms

Your input: `[[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20],[21,22,23,24,25]]`

Output: `1`

Expected: `1`

## 109. Clone a Graph [Asked by Facebook] (LeetCode 133)

**Success** Details

Runtime: 12 ms, faster than 30.02% of Java submissions  
 Memory Usage: 57.5 MB, less than 39.5% of Java submissions

Next challenges:

Recommended Questions: [Number of Connected Components in a Undirected Graph](#), [Number of Connected Components in a Directed Graph](#), [Find All Groups of Farmland](#), [Count Unreachable Pairs of Nodes in a Directed Graph](#)

Show off your acceptance:

Time Submitted	Status	Run
11/09/2022 12:08	Accepted	1
03/19/2021 23:26	Accepted	1
03/14/2021 08:09	Accepted	2
08/11/2020 06:22	Accepted	1
08/06/2020 16:21	Compile Error	16

```

class Solution {
    public Node cloneGraph(Node node) {
        if (node == null) return null;
        Map<Node, Node> map = new HashMap<>();
        Node newNode = new Node(node.val);
        map.put(node, newNode);
        Queue<Node> queue = new LinkedList<>();
        queue.add(node);
        while (!queue.isEmpty()) {
            Node curr = queue.poll();
            List<Node> newNeighbors = map.get(curr).neighbors;
            for (Node n : curr.neighbors) {
                if (!map.containsKey(n)) {
                    Node tmp = new Node(n.val);
                    map.put(n, tmp);
                    queue.add(n);
                }
                newNeighbors.add(map.get(n));
            }
        }
        return newNode;
    }
}
  
```

Accepted Runtime: 0 ms

Your input: `[[1,2,3],[2,4,5],[4,1]]`

Output: `1`

Expected: `1`

## 110. Bipartite Graph [Asked by Facebook, Microsoft] (LeetCode 785)

```

1 * class Solution {
2 *     public boolean BFS(int[][] graph, int[] colors, int node) {
3 *         Queue<Integer> queue = new LinkedList<>();
4 *         queue.add(node);
5 *         colors[node] = 1; // 1, -1
6 *         while (!queue.isEmpty()) {
7 *             int curr = queue.poll();
8 *             for (int adj : graph[curr]) {
9 *                 if (colors[adj] == 0) {
10 *                     colors[adj] = -1 * colors[curr];
11 *                     queue.add(adj);
12 *                 } else if (colors[curr] == colors[adj]) {
13 *                     return false;
14 *                 }
15 *             }
16 *         }
17 *         return true;
18 *     }
19 *     public boolean isBipartite(int[][] graph) {
20 *         int v = graph.length;
21 *         int[] colors = new int[v];
22 *         for (int node = 0; node < v; node++) {
23 *             if (colors[node] == 0) {
24 *                 if (BFS(graph, colors, node) == false) {
25 *                     return false;
26 *                 }
27 *             }
28 *         }
29 *         return true;
30 *     }
31 * } // TC: O(V * E), SC: O(V)

```

111. Topological Sort -- Kahn's Algorithm -- (Asked by Amazon, Microsoft)

The screenshot shows a coding practice interface with a 'Problem Solved Successfully' message. On the left, a summary box displays: 'Test Cases Passed: 73/73', 'Your Total Score: 342', 'Total Time Taken: 1.75', and 'Attempts No.: 2'. On the right, the code for Kahn's Algorithm is visible, including the calculation of in-degrees and the use of a queue to process nodes.

112. Fibonacci Number (LeetCode 509) [Asked by Amazon, Microsoft]

```

1 * class Solution {
2 *     public int fib(int n) {
3 *         if (n <= 1) return n;
4 *         int first = 0, second = 1;
5 *         for (int i = 2; i <= n; i++) {
6 *             int tmp = first + second;
7 *             first = second;
8 *             second = tmp;
9 *         }
10 *         return second;
11 *     }
12 * } // TC: O(n), SC: O(1)

```

113. Climbing Stairs (LeetCode 70) [Asked by Google, Microsoft, Amazon]

```

1 * class Solution {
2 *     public int climbStairs(int n) {
3 *         if (n <= 2) return n;
4 *         int[] dp = new int[n + 1];
5 *         dp[1] = 1;
6 *         dp[2] = 2;
7 *         for (int i = 3; i <= n; i++) {
8 *             dp[i] = dp[i - 1] + dp[i - 2];
9 *         }
10 *        return dp[n];
11 *    }
12 * } // TC: O(n), SC: O(n)

```

Constant space complexity:

```

1 * class Solution {
2 *     public int climbStairs(int n) {
3 *         if (n <= 2) return n;
4 *         int first = 1, second = 2;
5 *         for (int i = 1; i <= n-2; i++) {
6 *             int tmp = first + second;
7 *             first = second;
8 *             second = tmp;
9 *         }
10 *        return second;
11 *    }
12 * } // TC: O(n), SC: O(1)

```

114. Unique Path (LeetCode 62) [Asked by Amazon, Microsoft]

115. Partition Equal Subset Sum (LeetCode 416) [Asked by Google, Microsoft]

```

32
33 * class Solution{
34 *     static int equalPartition(int N, int arr[]) {
35 *         int total = Arrays.stream(arr).sum();
36 *         if (total % 2 == 1) return 0;
37 *
38 *         int target = total / 2;
39 *         boolean[] dp = new boolean[target + 1];
40 *         dp[0] = true;
41 *
42 *         for (int i = 0; i < arr.length; i++) {
43 *             for (int curr = dp.length - 1; curr >= 1; curr--) {
44 *                 if (curr >= arr[i]) {
45 *                     dp[curr] = dp[curr] || dp[curr - arr[i]];
46 *                 }
47 *             }
48 *         }
49 *         return dp[target] == true ? 1 : 0;
50 *     }
51 * } // TC: O(n * target), SC: O(target)

```

116. House Robber (LeetCode 198) [Asked by Google, Amazon]

Brute force solution:

```

2
3 * class Solution {
4 *     public int rob(int[] nums) {
5 *         if (nums.length == 1) return nums[0];
6 *         int[] dp = new int[nums.length + 1];
7 *         dp[0] = nums[0];
8 *         dp[1] = Math.max(nums[0], nums[1]);
9 *         for (int i = 2; i < nums.length; i++) {
10 *             int rob = nums[i] + dp[i - 2];
11 *             int notRob = dp[i - 1];
12 *             dp[i] = Math.max(rob, notRob);
13 *         }
14 *         return dp[nums.length - 1];
15 *     }
16 * } // TC: O(n), SC: O(n) - O(1) I

```

Optimal Solution:

```

1
2
3 * class Solution {
4 *     public int rob(int[] nums) {
5 *         if (nums.length == 1) return nums[0];
6 *         int prev1 = nums[0];
7 *         int prev2 = Math.max(nums[0], nums[1]);
8 *         for (int i = 2; i < nums.length; i++) {
9 *             int curr = Math.max(nums[i] + prev1, prev2);
10 *             prev1 = prev2;
11 *             prev2 = curr;
12 *         }
13 *         return prev2;
14 *     }
15 * } // TC: O(n), SC: O(1)

```

117. Coin Change (LeetCode 322) [Asked by Amazon,Microsoft]

```

1
2
3
4 * class Solution {
5 *     public int coinChange(int[] coins, int amount) {
6 *         int[] dp = new int[amount + 1];
7 *         for (int i = 1; i <= amount; i++) {
8 *             int min = Integer.MAX_VALUE;
9 *             for (int j = 0; j < coins.length; j++) {
10 *                 if (i >= coins[j] && dp[i - coins[j]] != -1) {
11 *                     min = Math.min(min, dp[i - coins[j]] + 1);
12 *                 }
13 *             }
14 *             dp[i] = (min == Integer.MAX_VALUE) ? -1 : min;
15 *         }
16 *         return dp[amount];
17 *     }
18 * } // TC: (n * amount), SC: O(amount)

```

118. Decode Ways (LeetCode 91) [Asked by Meta,Microsoft,Amazon]

```

class Solution {
    public int numDecodings(String s) {
        if (s.charAt(0) == '0') return 0;
        int n = s.length();

        int p = 1, pp = 1;
        for (int i = n - 1; i >= 0; i--) {
            int curr = s.charAt(i) == '0' ? 0 : p;
            if (i < n - 1 && (s.charAt(i) == '1' || s.charAt(i) == '2' && s.charAt(i + 1) <= '6'))
                curr += pp;
            pp = p;
            p = curr;
        }
        return p;
    }
} // TC: O(n), SC: O(1)

```

Hint : if(i<n-1 && (s.charAt(i) == '1' ||.....'6')){

### 119. Word Break (LeetCode 139) [Asked by Microsoft]

```

1 * class Solution {
2 *     public boolean wordBreak(String s, List<String> wordDict) {
3 *         boolean[] dp = new boolean[s.length() + 1];
4 *         dp[0] = true;
5 *         Set<String> set = new HashSet<>(wordDict);
6 *         for (int i = 1; i <= s.length(); i++) { // n
7 *             for (int j = 0; j < i; j++) { // n
8 *                 String suffix = s.substring(j, i); // n
9 *                 if (set.contains(suffix) && dp[j] == true) {
10 *                     dp[i] = true;
11 *                     break;
12 *                 }
13 *             }
14 *         }
15 *         return dp[s.length()];
16 *     }
17 * }
18 // TC: O(n^3 + m), SC: O(n + m)
19

```

### 120. Edit Distance (LeetCode 72) [Asked by Google, Amazon, Microsoft]

```

3 * class Solution {
4 *     public int minDistance(String word1, String word2) {
5 *         int m = word1.length(), n = word2.length();
6 *         if (m == 0) return n;
7 *         if (n == 0) return m;
8 *         int[][] dp = new int[m + 1][n + 1];
9 *         for (int i = 0; i <= m; i++) dp[i][0] = i; // row
10 *         for (int i = 0; i <= n; i++) dp[0][i] = i; // column
11 *         for (int i = 1; i <= m; i++) {
12 *             for (int j = 1; j <= n; j++) {
13 *                 char c1 = word1.charAt(i - 1);
14 *                 char c2 = word2.charAt(j - 1);
15 *                 if (c1 == c2) {
16 *                     dp[i][j] = dp[i - 1][j - 1];
17 *                 } else {
18 *                     // insert, delete, replace
19 *                     int insert = dp[i][j - 1] + 1;
20 *                     int delete = dp[i - 1][j] + 1;
21 *                     int replace = dp[i - 1][j - 1] + 1;
22 *                     dp[i][j] = 1 + Math.min(insert, Math.min(delete, replace));
23 *                 }
24 *             }
25 *         }
26 *         return dp[m][n];
27 *     }
28 * } // TC: O(m * n), SC: O(m * n)
29
30

```

### 121. Interleaving String (LeetCode 97) [Asked by Google, Microsoft, Amazon]

```

1 class Solution {
2     public boolean isInterleave(String s1, String s2, String s3) {
3         int n1 = s1.length(), n2 = s2.length(), n3 = s3.length();
4         if (n1 + n2 != n3) return false;
5         boolean[][] dp = new boolean[n1 + 1][n2 + 1];
6         dp[n1][n2] = true;
7         for (int i = n1; i >= 0; i--) {
8             for (int j = n2; j >= 0; j--) {
9                 if (i < n1 && s1.charAt(i) == s3.charAt(i + j) && dp[i + 1][j] == true) {
10                     dp[i][j] = true;
11                 }
12                 if (j < n2 && s2.charAt(j) == s3.charAt(i + j) && dp[i][j + 1] == true) {
13                     dp[i][j] = true;
14                 }
15             }
16         }
17         return dp[0][0];
18     }
19 }
20
21 // TC: O(S*T), SC: O(S*T)

```

122. Longest Increasing Subsequence(LeetCode 300) [Asked by Google, Microsoft]

```

2
3 class Solution {
4     public int lengthOfLIS(int[] nums) {
5         int n = nums.length;
6         if (n == 1) return 1;
7         int[] dp = new int[n];
8         dp[0] = 1;
9         int ans = 0;
10        for (int i = 1; i < n; i++) {
11            int len = 0;
12            for (int j = 0; j < i; j++) {
13                if (nums[j] < nums[i]) {
14                    len = Math.max(len, dp[j]);
15                }
16            }
17            dp[i] = 1 + len;
18            ans = Math.max(ans, dp[i]);
19        }
20        return ans;
21    }
22    // TC: O(n^2), SC: O(n)

```

123. Longest Common Subsequence (LeetCode 1143)

```

1 class Solution {
2     public int longestCommonSubsequence(String text1, String text2) {
3         int[][] dp = new int[text1.length() + 1][text2.length() + 1];
4         for (int i = 1; i <= dp.length; i++) {
5             for (int j = 1; j <= dp[0].length; j++) {
6                 if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
7                     dp[i][j] = 1 + dp[i - 1][j - 1];
8                 } else {
9                     dp[i][j] = Math.max(dp[i][j - 1], dp[i - 1][j]);
10                }
11            }
12        }
13        return dp[text1.length()][text2.length()];
14    }
15    // TC: O(m * n), SC: O(m * n)

```

124. Longest Palindromic Subsequence (LeetCode 516) [Asked by Google, Amazon]



