# Section 1

## 1. Array Manipulation: Three Sum

**Question:** Given an array of integers, return all unique triplets [a,b,c][a, b, c][a,b,c] in the array such that a+b+c=0a + b + c = 0a+b+c=0. The solution set must not contain duplicate triplets.

**Example:** Input: nums = [-1, 0, 1, 2, -1, -4]. Output: [[-1, 0, 1], [-1, -1,

2]].

void main() {

  List<int> nums = [-1, 0, 1, 2, -1, -4];

  List<List<int>> result = threeSum(nums);

  print(result); // Output: [[-1, 0, 1], [-1, -1, 2]]

}

List<List<int>> threeSum(List<int> nums) {

  nums.sort(); // Sort the array

  List<List<int>> result = [];

  for (int i = 0; i < nums.length - 2; i++) {

    // Skip duplicates for i

    if (i > 0 && nums[i] == nums[i - 1]) continue;

    int j = i + 1;

    int k = nums.length - 1;

    while (j < k) {

      int sum = nums[i] + nums[j] + nums[k];

      if (sum < 0) {

        j++; // Increase j to increase sum

      } else if (sum > 0) {

        k--; // Decrease k to decrease sum

      } else {

        result.add([nums[i], nums[j], nums[k]]);

        // Skip duplicates for j and k

        while (j < k && nums[j] == nums[j + 1]) j++;

        while (j < k && nums[k] == nums[k - 1]) k--;

        j++;

        k--;

      }

    }

  }

  return result;

}

## 2. String Manipulation: Longest Palindromic Substring

**Question: Given a string s, find the longest palindromic substring in s.**

**Example: Input: "babad". Output: "bab" or "aba".**

String longestPalindromicSubstring(String s) {

  if (s.isEmpty) return "";

  int start = 0, end = 0;

  for (int i = 0; i < s.length; i++) {

    int len1 = expandAroundCenter(s, i, i);

    int len2 = expandAroundCenter(s, i, i + 1);

    int len = len1 > len2 ? len1 : len2;

    if (len > end - start) {

      start = i - (len - 1) ~/ 2;

      end = i + len ~/ 2;

    }

  }

  return s.substring(start, end + 1);

}

int expandAroundCenter(String s, int left, int right) {

  while (left >= 0 && right < s.length && s[left] == s[right]) {

    left--;

    right++;

  }

  return right - left - 1;

}

void main() {

  String s = "babad";

  // ignore: avoid\_print

  print("Longest palindromic substring: ${longestPalindromicSubstring(s)}");

}

## 3. Tree Traversal: Binary Tree Zigzag Level Order Traversal

**Question: Given a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).**

**Example: Input: [3,9,20,null,null,15,7]. Output: [[3],[20,9],[15,7]].**

import 'dart:collection';

// TreeNode class definition

class TreeNode {

  int val;

  TreeNode? left;

  TreeNode? right;

  TreeNode(this.val, {this.left, this.right});

}

// Function to perform Zigzag Level Order Traversal

List<List<int>> zigzagLevelOrder(TreeNode? root) {

  List<List<int>> result = [];

  if (root == null) return result;

  Queue<TreeNode> queue = Queue();

  queue.add(root);

  bool leftToRight = true;

  while (queue.isNotEmpty) {

    int levelSize = queue.length;

    List<int> levelNodes = [];

    for (int i = 0; i < levelSize; i++) {

      TreeNode node = queue.removeFirst();

      if (leftToRight) {

        levelNodes.add(node.val);

      } else {

        levelNodes.insert(0, node.val);  // Insert at the beginning for reverse order

      }

      if (node.left != null) queue.add(node.left!);

      if (node.right != null) queue.add(node.right!);

    }

    result.add(levelNodes);

    leftToRight = !leftToRight;

  }

  return result;

}

// Example usage

void main() {

  // Constructing the binary tree

  TreeNode root = TreeNode(3,

    left: TreeNode(9),

    right: TreeNode(20,

      left: TreeNode(15),

      right: TreeNode(7),

    ),

  );

  List<List<int>> result = zigzagLevelOrder(root);

  print(result);  // Output: [[3], [20, 9], [15, 7]]

}

**4. Sorting Algorithms: Quick Sort**

**Question:** Implement the quick sort algorithm to sort an array of integers in ascending order.

// Quick Sort function

void quickSort(List<int> arr, int low, int high) {

  if (low < high) {

    int pivotIndex = partition(arr, low, high);

    // Recursively apply the same logic to the left and right subarrays

    quickSort(arr, low, pivotIndex - 1);

    quickSort(arr, pivotIndex + 1, high);

  }

}

// Partition function to place the pivot element in its correct position

int partition(List<int> arr, int low, int high) {

  int pivot = arr[high];

  int i = low - 1; // Index of the smaller element

  for (int j = low; j < high; j++) {

    if (arr[j] <= pivot) {

      i++;

      // Swap arr[i] and arr[j]

      int temp = arr[i];

      arr[i] = arr[j];

      arr[j] = temp;

    }

  }

  // Swap arr[i+1] and arr[high] (or pivot)

  int temp = arr[i + 1];

  arr[i + 1] = arr[high];

  arr[high] = temp;

  return i + 1;

}

// Example usage

void main() {

  List<int> arr = [10, 7, 8, 9, 1, 5];

  int n = arr.length;

  print("Unsorted array: $arr");

  quickSort(arr, 0, n - 1);

  print("Sorted array: $arr");

}

## 5. Dynamic Programming: Unique Paths

**Question: A robot is located at the top-left corner of a m x n grid. The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid. How many possible unique paths are there?**

**Example: Input: m = 3, n = 7. Output: 28.**

int uniquePaths(int m, int n) {

  // Create a 2D table to store the number of unique paths

  List<List<int>> dp = List.generate(m, (\_) => List.filled(n, 0));

  // Initialize the first row and the first column

  for (int i = 0; i < m; i++) {

    dp[i][0] = 1;

  }

  for (int j = 0; j < n; j++) {

    dp[0][j] = 1;

  }

  // Fill the rest of the table

  for (int i = 1; i < m; i++) {

    for (int j = 1; j < n; j++) {

      dp[i][j] = dp[i - 1][j] + dp[i][j - 1];

    }

  }

  // The number of unique paths to reach the bottom-right corner

  return dp[m - 1][n - 1];

}

void main() {

  int m = 3;

  int n = 7;

  print("Number of unique paths for a $m x $n grid: ${uniquePaths(m, n)}");

}