Coding Practice Set – 8

1. **3SumCloset**

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.
Return *the sum of the three integers*.
You may assume that each input would have exactly one solution.

Example 1:
Input: nums = [-1,2,1,-4], target = 1
Output: 2
Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
Example 2:
Input: nums = [0,0,0], target = 1
Output: 0
Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

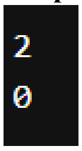
Constraints:

- 3 <= nums.length <= 500
- -1000 <= nums[i] <= 1000
- -10^4 <= target <= 10^4

Code:

```
import java.util.Arrays;

class Solution {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);
        int closestSum = Integer.MAX_VALUE;
        for (int i = 0; i < nums.length - 2; i++) {
            int left = i + 1, right = nums.length - 1;
            while (left < right) {
                int currentSum = nums[i] + nums[left] + nums[right];
                if (Math.abs(target - currentSum) < Math.abs(target - closestSum)) {
                    closestSum = currentSum;
                }
                if (currentSum < target) {
                    left++;
                } else if (currentSum > target) {
                    right--;
                } else {
                    return currentSum;
                }
            }
        }
        return closestSum;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        System.out.println(solution.threeSumClosest(new int[]{-1, 2, 1, -4}, 1));
        System.out.println(solution.threeSumClosest(new int[]{0, 0, 0}, 1));
    }
}
```

Output:



```
2
0
```

Time complexity: O(n^2)

2. **Jump Game II**

You are given a **0-indexed** array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

- 0 <= j <= nums[i] and
- i + j < n

Return *the minimum number of jumps to reach* nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Example 1:
Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.
Example 2:
Input: nums = [2,3,0,1,4]
Output: 2

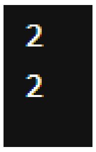
Constraints:

- 1 <= nums.length <= 10^4
- 0 <= nums[i] <= 1000
- It's guaranteed that you can reach nums[n - 1].

Code:

```
class Solution {
    public int jump(int[] nums) {
        int jumps = 0, currentEnd = 0, farthest = 0;
        for (int i = 0; i < nums.length - 1; i++) {
            farthest = Math.max(farthest, i + nums[i]);
            if (i == currentEnd) {
                jumps++;
                currentEnd = farthest;
            }
        }
        return jumps;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        System.out.println(solution.jump(new int[]{2, 3, 1, 1, 4}));
        System.out.println(solution.jump(new int[]{2, 3, 0, 1, 4}));
    }
}
```

Output:


```
2
2
```

Time complexity: O(n)

3. **Group Anagrams**
Given an array of strings strs, group the
anagrams
together. You can return the answer in **any order**.

Example 1:
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
Explanation:

- There is no string in strs that can be rearranged to form "bat".
- The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.
- The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

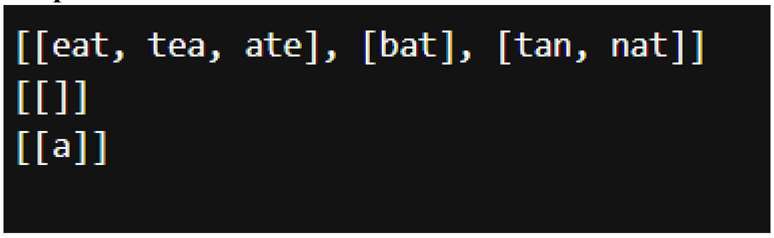
Example 2:
Input: strs = [""]
Output: [[""]]
Example 3:
Input: strs = ["a"]
Output: [["a"]]

Constraints:

- 1 <= strs.length <= 10^4
- 0 <= strs[i].length <= 100
- strs[i] consists of lowercase English letters.

Code:
```java
import java.util.*;

class Solution {
   public List<List<String>> groupAnagrams(String[] strs) {
      Map<String, List<String>> map = new HashMap<>();
      for (String s : strs) {
         char[] charArray = s.toCharArray();
         Arrays.sort(charArray);
         String sorted = new String(charArray);
         map.putIfAbsent(sorted, new ArrayList<>());
         map.get(sorted).add(s);
      }
      return new ArrayList<>(map.values());
   }

   public static void main(String[] args) {
      Solution solution = new Solution();
      System.out.println(solution.groupAnagrams(new String[]{"eat", "tea", "tan", "ate", "nat",
"bat"}));
      System.out.println(solution.groupAnagrams(new String[]{""}));
      System.out.println(solution.groupAnagrams(new String[]{"a"}));
   }
}
```

Output:



```
[[eat, tea, ate], [bat], [tan, nat]]
[[]]
[[a]]
```

Time Complexity: O(n·k logk)

4. **Decode Ways**
   You have intercepted a secret message encoded as a string of numbers. The message is **decoded** via the following mapping:
   "1" -> 'A'
   "2" -> 'B'
   ...
   "25" -> 'Y'
   "26" -> 'Z'
   However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").
   For example, "11106" can be decoded into:

- "AAJF" with the grouping (1, 1, 10, 6)
- "KJF" with the grouping (11, 10, 6)
- The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.

Given a string s containing only digits, return the **number of ways** to **decode** it. If the entire string cannot be decoded in any valid way, return 0.
The test cases are generated so that the answer fits in a **32-bit** integer.

Example 1:
Input: s = "12"
Output: 2
Explanation:
"12" could be decoded as "AB" (1 2) or "L" (12).
Example 2:
Input: s = "226"
Output: 3
Explanation:
"226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
Example 3:
Input: s = "06"
Output: 0
Explanation:
"06" cannot be mapped to "F" because of the leading zero ("6" is different from "06"). In this case, the string is not a valid encoding, so return 0.

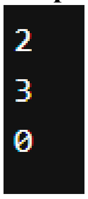
Constraints:

- 1 <= s.length <= 100
- s contains only digits and may contain leading zero(s).

Code:

```
class Solution {
    public int numDecodings(String s) {
        if (s == null || s.length() == 0 || s.charAt(0) == '0') return 0;
        int n = s.length();
        int[] dp = new int[n + 1];
        dp[0] = 1;
        dp[1] = s.charAt(0) != '0' ? 1 : 0;
        for (int i = 2; i <= n; i++) {
            int oneDigit = Integer.parseInt(s.substring(i - 1, i));
            int twoDigits = Integer.parseInt(s.substring(i - 2, i));
            if (oneDigit >= 1 && oneDigit <= 9) dp[i] += dp[i - 1];
            if (twoDigits >= 10 && twoDigits <= 26) dp[i] += dp[i - 2];
        }
        return dp[n];
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        System.out.println(solution.numDecodings("12"));
        System.out.println(solution.numDecodings("226"));
        System.out.println(solution.numDecodings("06"));
    }
}
```

Output:



```
2
3
0
```

Time Complexity: O(n)

5. **Best time to buy and sell stock II**
   You are given an integer array prices where prices[i] is the price of a given stock on the i^{th} day.
   On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.
   Find and return *the **maximum** profit you can achieve*.

   Example 1:
   Input: prices = [7,1,5,3,6,4]
   Output: 7
   Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
   Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
   Total profit is 4 + 3 = 7.
   Example 2:
   Input: prices = [1,2,3,4,5]
   Output: 4
   Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
   Total profit is 4.
   Example 3:
   Input: prices = [7,6,4,3,1]
   Output: 0
   Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

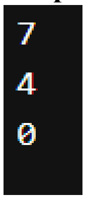
   Constraints:

- $1 <= prices.length <= 3 * 10^4$
- $0 <= prices[i] <= 10^4$

Code:

```
class Solution {
    public int maxProfit(int[] prices) {
        int maxProfit = 0;
        for (int i = 1; i < prices.length; i++) {
            if (prices[i] > prices[i - 1]) {
                maxProfit += prices[i] - prices[i - 1];
            }
        }
        return maxProfit;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        System.out.println(solution.maxProfit(new int[]{7, 1, 5, 3, 6, 4}));
        System.out.println(solution.maxProfit(new int[]{1, 2, 3, 4, 5}));
        System.out.println(solution.maxProfit(new int[]{7, 6, 4, 3, 1}));
    }
}
```

Output:



```
7
4
0
```

Time Complexity: O(n)

6. **Number of Islands**

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]

Output: 1

Example 2:

Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]

Output: 3

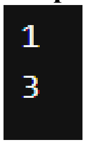
Constraints:

- m == grid.length
- n == grid[i].length
- 1 <= m, n <= 300
- grid[i][j] is '0' or '1'.

Code:

```
class Solution {
    public int numIslands(char[][] grid) {
        int count = 0;
        int m = grid.length, n = grid[0].length;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == '1') {
                    count++;
                    dfs(grid, i, j, m, n);
                }
            }
        }
        return count;
    }

    private void dfs(char[][] grid, int i, int j, int m, int n) {
        if (i < 0 || j < 0 || i >= m || j >= n || grid[i][j] == '0') {
            return;
        }
        grid[i][j] = '0';
        dfs(grid, i + 1, j, m, n);
        dfs(grid, i - 1, j, m, n);
        dfs(grid, i, j + 1, m, n);
        dfs(grid, i, j - 1, m, n);
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        System.out.println(solution.numIslands(new char[][]{
            {'1', '1', '1', '1', '0'},
            {'1', '1', '0', '1', '0'},
            {'1', '1', '0', '0', '0'},
            {'0', '0', '0', '0', '0'}
        }));

        System.out.println(solution.numIslands(new char[][]{
            {'1', '1', '0', '0', '0'},
            {'1', '1', '0', '0', '0'},
            {'0', '0', '1', '0', '0'},
            {'0', '0', '0', '1', '1'}
        }));
    }
}
```

Output:



```
1
3
```

Time Complexity: O(m*n)

7. Quick Sort

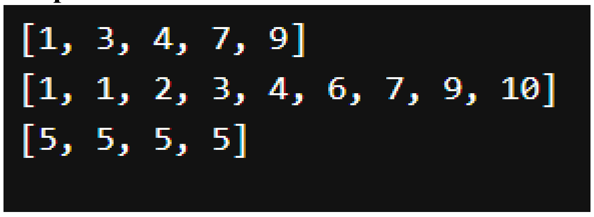
Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, **arr**[] in ascending order. Given an array, **arr**[], with starting index **low** and ending index **high**, complete the functions **partition()** and **quickSort()**. Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.

Examples:
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.
Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]
Output: **[**1, 1, 2, 3, 4, 6, 7, 9, 10]
Explanation: Duplicate elements (1) are retained in sorted order.
Input: arr[] = [5, 5, 5, 5]
Output: [5, 5, 5, 5]
Explanation: All elements are identical, so the array remains unchanged.
Constraints:
1 <= arr.size() <= 10^3
1 <= arr[i] <= 10^4

Code:
```
class Solution {
   public void quickSort(int[] arr, int low, int high) {
      if (low < high) {
         int pi = partition(arr, low, high);
         quickSort(arr, low, pi - 1);
         quickSort(arr, pi + 1, high);
      }
   }

   private int partition(int[] arr, int low, int high) {
      int pivot = arr[high];
      int i = (low - 1);
      for (int j = low; j <= high - 1; j++) {
         if (arr[j] <= pivot) {
            i++;
            swap(arr, i, j);
         }
      }
      swap(arr, i + 1, high);
      return i + 1;
   }

   private void swap(int[] arr, int i, int j) {
      int temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }

   public static void main(String[] args) {
      Solution solution = new Solution();
```

```
        int[] arr1 = {4, 1, 3, 9, 7};
        solution.quickSort(arr1, 0, arr1.length - 1);
        System.out.println(java.util.Arrays.toString(arr1));

        int[] arr2 = {2, 1, 6, 10, 4, 1, 3, 9, 7};
        solution.quickSort(arr2, 0, arr2.length - 1);
        System.out.println(java.util.Arrays.toString(arr2));

        int[] arr3 = {5, 5, 5, 5};
        solution.quickSort(arr3, 0, arr3.length - 1);
        System.out.println(java.util.Arrays.toString(arr3));
    }
}
```

Output:



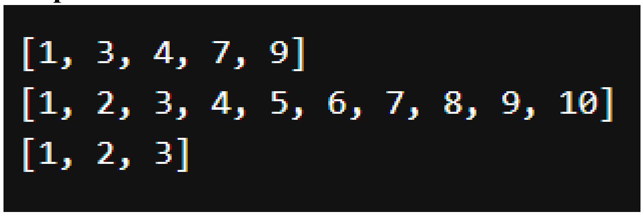
```
[1, 3, 4, 7, 9]
[1, 1, 2, 3, 4, 6, 7, 9, 10]
[5, 5, 5, 5]
```

Time Complexity: O(n log n)

8. **Merge Sort**
Given an array arr[], its starting position l and its ending position r. Sort the array using the merge sort algorithm.
Examples:
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Input: arr[] = [1, 3 , 2]
Output: [1, 2, 3]
Constraints:
1 <= arr.size() <= 10^5
1 <= arr[i] <= 10^5

Code:
```
class Solution {
    public void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    private void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1;
        int n2 = r - m;

        int[] left = new int[n1];
        int[] right = new int[n2];

        System.arraycopy(arr, l, left, 0, n1);
        System.arraycopy(arr, m + 1, right, 0, n2);

        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (left[i] <= right[j]) {
                arr[k] = left[i];
                i++;
            } else {
                arr[k] = right[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = left[i];
            i++;
            k++;
        }

        while (j < n2) {
            arr[k] = right[j];
            j++;
            k++;
        }
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] arr1 = {4, 1, 3, 9, 7};
        solution.mergeSort(arr1, 0, arr1.length - 1);
        System.out.println(java.util.Arrays.toString(arr1));

        int[] arr2 = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
        solution.mergeSort(arr2, 0, arr2.length - 1);
        System.out.println(java.util.Arrays.toString(arr2));

        int[] arr3 = {1, 3, 2};
        solution.mergeSort(arr3, 0, arr3.length - 1);
        System.out.println(java.util.Arrays.toString(arr3));
    }
}
```

Output:



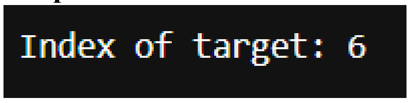
```
[1, 3, 4, 7, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3]
```

Time Complexity: O(n log n)

9. **Ternary Search**
Code:

```
public class TernarySearch {
    public static int ternarySearch(int[] arr, int left, int right, int target) {
        if (right >= left) {
            int mid1 = left + (right - left) / 3;
            int mid2 = right - (right - left) / 3;
            if (arr[mid1] == target) return mid1;
            if (arr[mid2] == target) return mid2;
            if (target < arr[mid1]) return ternarySearch(arr, left, mid1 - 1, target);
            else if (target > arr[mid2]) return ternarySearch(arr, mid2 + 1, right, target);
            else return ternarySearch(arr, mid1 + 1, mid2 - 1, target);
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int target = 7;
        int result = ternarySearch(arr, 0, arr.length - 1, target);
        System.out.println("Index of target: " + result);
    }
}
```

Output:



```
Index of target: 6
```

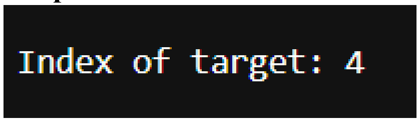
Time Complexity: O(1)

10. **Interpolation search**

Code:

```
public class InterpolationSearch {
    public static int interpolationSearch(int[] arr, int low, int high, int target) {
        while (low <= high && target >= arr[low] && target <= arr[high]) {
            int pos = low + (target - arr[low]) * (high - low) / (arr[high] - arr[low]);
            if (arr[pos] == target) return pos;
            if (arr[pos] < target) low = pos + 1;
            else high = pos - 1;
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50, 60, 70, 80, 90};
        int target = 50;
        int result = interpolationSearch(arr, 0, arr.length - 1, target);
        System.out.println("Index of target: " + result);
    }
}
```

Output:



```
Index of target: 4
```

Time Complexity: O(1)