Problem 1: 1055-pairs-of-songs-with-total-durations-divisible-by-60

Problem Statement

Pairs of Songs With Total Durations Divisible by 60

You are given a list of songs where the ith song has a duration of time[i] seconds. Return the number of pairs of songs for which their total duration in seconds is divisible by 60. Formally, we want the number of indices i, j such that i < j with (time[i] + time[j]) % 60 == 0.

Example 1:
Input: time = [30,20,150,100,40]
Output: 3
Explanation: Three pairs have a total duration divisible by 60: (time[0] = 30, time[2] = 150): total duration 180 (time[1] = 20, time[3] = 100): total duration 120 (time[1] = 20, time[4] = 40): total duration 60

Example 2:
Input: time = [60,60,60]
Output: 3
Explanation: All three pairs have a total duration of 120, which is divisible by 60.

Constraints:
1 <= time.length <= 6 * 10^4
1 <= time[i] <= 500

Java Code

```
--- File: pairs-of-songs-with-total-durations-divisible-by-60.java ---
class Solution {
    public int numPairsDivisibleBy60(int[] time) {
        int[] count = new int[60];
        int res = 0;

        for (int t : time) {
            int rem = t % 60;
            int complement = (60 - rem) % 60;
            res = res + count[complement];
            count[rem]++;
        }

        return res;
    }
}
```

Problem 2: 11-container-with-most-water

Problem Statement

Container With Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store. Notice that you may not slant the container.

Example 1:
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:
Input: height = [1,1]
Output: 1

Constraints:

- n == height.length
- 2 <= n <= 105
- 0 <= height[i] <= 104

Java Code

```
--- File: container-with-most-water.java ---
public class Solution {
    public int maxArea(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int maxArea = 0;

        while (left < right) {
            int width = right - left;
            int minHeight = Math.min(height[left], height[right]);
            int area = width * minHeight;

            maxArea = Math.max(maxArea, area);

            // Move the pointer pointing to the shorter line
            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return maxArea;
    }
}
```

Problem 3: 14-longest-common-prefix

Problem Statement

Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".

Example 1:
Input: strs = ["flower", "flow", "flight"]
Output: "fl"

Example 2:
Input: strs = ["dog", "racecar", "car"]
Output: ""

Explanation: There is no common prefix among the input strings.

Constraints:
1 <= strs.length <= 200
0 <= strs[i].length <= 200
strs[i] consists of only lowercase English letters if it is non-empty.

Java Code

```
--- File: longest-common-prefix.java ---
class Solution {
    public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) return "";
        String cm = strs[0];
        for (int i = 1; i < strs.length && !cm.isEmpty(); i++) {
            String s = strs[i];
            int min = Math.min(s.length(), cm.length());
            int j = 0;
            while (j < min && s.charAt(j) == cm.charAt(j)) j++;
```

```
            cm = cm.substring(0, j);
        }
        return cm;
        }

}
```

Problem 4: 15-3sum

Problem Statement

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets.

Example 1:
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation: nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0. nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0. nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0. The distinct triplets are [-1,0,1] and [-1,-1,2]. Notice that the order of the output and the order of the triplets does not matter.

Example 2:
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.

Example 3:
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.

Constraints:
- 3 <= nums.length <= 3000
- -105 <= nums[i] <= 105

Java Code

```
--- File: 3sum.java ---
import java.util.*;

class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();

        Arrays.sort(nums);

        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            int left = i + 1;
            int right = nums.length - 1;

            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];

                if (sum == 0) {
                    result.add(Arrays.asList(nums[i], nums[left], nums[right]));
                    while (left < right && nums[left] == nums[left + 1]) left++;
                    while (left < right && nums[right] == nums[right - 1]) right--;

                    left++;
                    right--;
                } else if (sum < 0) {
                    left++;
                } else {
                    right--;
                }
```

```
            }
        }

        return result;
    }
}
```

Problem 5: 152-maximum-product-subarray

Problem Statement

Maximum Product Subarray

Given an integer array nums, find a subarray that has the largest product, and return the product. The test cases are generated so that the answer will fit in a 32-bit integer.

Example 1:
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.

Example 2:
Input: nums = [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

Constraints:
1 <= nums.length <= 2 * 10^4
-10 <= nums[i] <= 10
The product of any subarray of nums is guaranteed to fit in a 32-bit integer.

Java Code

```
--- File: maximum-product-subarray.java ---
class Solution {
    public int maxProduct(int[] nums) {
        int max = nums[0]; int cmax = nums[0]; int cmin = nums[0];
        for(int i = 1; i<nums.length ; i++){
            if(nums[i]<0){
                int temp = cmax;
                cmax = cmin;
                cmin = temp;
            }
            cmax = Math.max(nums[i], nums[i] * cmax);
            cmin = Math.min(nums[i], nums[i] * cmin);
            max = Math.max(max, cmax);
        }
        return max;
    }
}
```

Problem 6: 16-3sum-closest

Problem Statement

Here is the extracted problem statement in clean, plain text format:

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
- -104 <= target <= 104

Java Code

```
--- File: 3sum-closest.java ---
class Solution {
    public int threeSumClosest(int[] a, int target) {
        int closestSum = 0;
        int diff = Integer.MAX_VALUE;

        for (int i = 0; i < a.length; i++) {
            for (int j = i + 1; j < a.length; j++) {
                for (int k = j + 1; k < a.length; k++) {
                    int sum = a[i] + a[j] + a[k];
                    int currDiff = Math.abs(target - sum);

                    if (currDiff < diff) {
                        diff = currDiff;
                        closestSum = sum;
                    }
                }
            }
        }
        return closestSum;
    }
}
```

Problem 7: 169-majority-element

Problem Statement

Here's the extracted problem statement in clean, plain text format:

Given an array nums of size n, return the majority element. The majority element is the element that appears more than floor(n / 2) times. You may assume that the majority element always exists in the array.

Example 1:
Input: nums = [3,2,3]
Output: 3

Example 2:
Input: nums = [2,2,1,1,1,2,2]
Output: 2

Constraints:
- n == nums.length
- 1 <= n <= 5 * 10^4
- -10^9 < nums[i] < 10^9

Follow-up: Could you solve the problem in linear time and in O(1) space?

Java Code

```
--- File: majority-element.java ---
class Solution {
    public int majorityElement(int[] nums) {
```

```
        int count = 0; int candidate = 0;
        for(int num : nums){
            if(count == 0) candidate = num;
            count += (num == candidate) ? 1 : -1;
        }
        return candidate;
    }
}
```

Problem 8: 18-4sum

Problem Statement

4Sum

Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that:
0 <= a, b, c, d < n
a, b, c, and d are distinct.
nums[a] + nums[b] + nums[c] + nums[d] == target
You may return the answer in any order.

Example 1:
Input: nums = [1,0,-1,0,-2,2], target = 0
Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

Example 2:
Input: nums = [2,2,2,2,2], target = 8
Output: [[2,2,2,2]]

Constraints:
1 <= nums.length <= 200
-10^9 <= nums[i] <= 10^9
-10^9 <= target <= 10^9

Java Code

```
--- File: 4sum.java ---
import java.util.*;

class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> res = new ArrayList<>();
        if (nums == null || nums.length < 4) return res;

        Arrays.sort(nums); // Step 1: Sort array

        int n = nums.length;

        // Step 2: First loop for first number
        for (int i = 0; i < n - 3; i++) {
            // Avoid duplicate for first number
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            // Step 3: Second loop for second number
            for (int j = i + 1; j < n - 2; j++) {
                // Avoid duplicate for second number
                if (j > i + 1 && nums[j] == nums[j - 1]) continue;

                int left = j + 1;
                int right = n - 1;

                // Step 4: Two-pointer search for remaining two numbers
                while (left < right) {
                    long sum = (long) nums[i] + nums[j] + nums[left] + nums[right];

                    if (sum == target) {
```

```
                        res.add(Arrays.asList(nums[i], nums[j], nums[left], nums[right]));

                        // Move pointers
                        left++;
                        right--;

                        // Skip duplicates
                        while (left < right && nums[left] == nums[left - 1]) left++;
                        while (left < right && nums[right] == nums[right + 1]) right--;

                    } else if (sum < target) {
                        left++; // need bigger sum
                    } else {
                        right--; // need smaller sum
                    }
                }
            }
        }
        return res;
    }
}
```

Problem 9: 189-rotate-array

Problem Statement

Rotate Array

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

Example 1:
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]

Example 2:
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]

Constraints:
1 <= nums.length <= 10^5
-2^31 <= nums[i] <= 2^31 - 1
0 <= k <= 10^5

Follow up:
Try to come up with as many solutions as you can. There are at least three different ways to solve this problem. Could you do it in-place with O(1) extra space?

Java Code

```
--- File: rotate-array.java ---
class Solution {
    public void rotate(int[] nums, int k) {
        k = k % nums.length;
        reverse(nums, 0, nums.length - 1);
        reverse(nums, 0, k - 1);
        reverse(nums, k, nums.length - 1);
    }
    private void reverse(int[] nums, int start, int end) {
```

```
        while (start < end) {
            int temp = nums[start];
            nums[start++] = nums[end];
            nums[end--] = temp;
        }
    }
}
```

Problem 10: 2021-remove-all-occurrences-of-a-substring

Problem Statement

Remove All Occurrences of a Substring

Given two strings s and part, perform the following operation on s until all occurrences of the substring part are removed: Find the leftmost occurrence of the substring part and remove it from s. Return s after removing all occurrences of part. A substring is a contiguous sequence of characters in a string.

Example 1:
Input: s = "daabcbaabcbc", part = "abc"
Output: "dab"
Explanation: The following operations are done:
- s = "daabcbaabcbc", remove "abc" starting at index 2, so s = "dabaabcbc".
- s = "dabaabcbc", remove "abc" starting at index 4, so s = "dababc".
- s = "dababc", remove "abc" starting at index 3, so s = "dab".
Now s has no occurrences of "abc".

Example 2:
Input: s = "axxxxyyyyb", part = "xy"
Output: "ab"
Explanation: The following operations are done:
- s = "axxxxyyyyb", remove "xy" starting at index 4 so s = "axxxyyyb".
- s = "axxxyyyb", remove "xy" starting at index 3 so s = "axxyyb".
- s = "axxyyb", remove "xy" starting at index 2 so s = "axyb".
- s = "axyb", remove "xy" starting at index 1 so s = "ab".
Now s has no occurrences of "xy".

Constraints:
1 <= s.length <= 1000
1 <= part.length <= 1000
s and part consists of lowercase English letters.

Java Code

```
--- File: remove-all-occurrences-of-a-substring.java ---
class Solution {
    public String removeOccurrences(String s, String part) {

        while (s.contains(part)) {   // keep removing until no "part" is left
            s = s.replaceFirst(part, "");
        }
        return s;

    }
}
```