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;

    }
}
```

Problem 11: 2112-minimum-difference-between-highest-and-lowest-of-k-scores

Problem Statement

Minimum Difference Between Highest and Lowest of K Scores

You are given a 0-indexed integer array nums, where nums[i] represents the score of the ith student. You are also given an integer k. Pick the

scores of any k students from the array so that the difference between the highest and the lowest of the k scores is minimized. Return the minimum possible difference.

Example 1:
Input: nums = [90], k = 1
Output: 0
Explanation: There is one way to pick score(s) of one student: [90]. The difference between the highest and lowest score is 90 - 90 = 0. The minimum possible difference is 0.

Example 2:
Input: nums = [9,4,1,7], k = 2
Output: 2
Explanation: There are six ways to pick score(s) of two students: [9,4,1,7]. The difference between the highest and lowest score is 9 - 4 = 5. [9,4,1,7]. The difference between the highest and lowest score is 9 - 1 = 8. [9,4,1,7]. The difference between the highest and lowest score is 9 - 7 = 2. [9,4,1,7]. The difference between the highest and lowest score is 4 - 1 = 3. [9,4,1,7]. The difference between the highest and lowest score is 7 - 4 = 3. [9,4,1,7]. The difference between the highest and lowest score is 7 - 1 = 6. The minimum possible difference is 2.

Constraints:
1 <= k <= nums.length <= 1000
0 <= nums[i] <= 10^5

Java Code

```
--- File: minimum-difference-between-highest-and-lowest-of-k-scores.java ---
class Solution {
    public int minimumDifference(int[] nums, int k) {
        int mindiff = Integer.MAX_VALUE;

        Arrays.sort(nums);
        for(int i = 0; i<=nums.length -k; i++){
            int diff = nums[i+k-1] - nums[i];
            mindiff = Math.min(mindiff, diff);
        }

        return mindiff;

    }
}
```

Problem 12: 217-contains-duplicate

Problem Statement

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Example 1:
Input: nums = [1,2,3,1]
Output: true
Explanation: The element 1 occurs at the indices 0 and 3.

Example 2:
Input: nums = [1,2,3,4]
Output: false
Explanation: All elements are distinct.

Example 3:
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true

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

Java Code

```
--- File: contains-duplicate.java ---
class Solution {
    public boolean containsDuplicate(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for(int num : nums){
            if(!set.add(num)){
                return true;
            }
        }
        return false;
    }
}
```

Problem 13: 26-remove-duplicates-from-sorted-array

Problem Statement

Remove Duplicates from Sorted Array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums. Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things: Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums. Return k.

Custom Judge: The judge will test your solution with the following code: int[] nums = [...]; // Input array int[] expectedNums = [...]; // The expected answer with correct length int k = removeDuplicates(nums); // Calls your implementation assert k == expectedNums.length; for (int i = 0; i < k; i++) { assert nums[i] == expectedNums[i]; }

Example 1:
Input: nums = [1,1,2]
Output: 2, nums = [1,2,]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively. It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:
Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,,,,,]
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively. It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:
1 <= nums.length <= 3 * 10^4
-100 <= nums[i] <= 100
nums is sorted in non-decreasing order.

Java Code

```
--- File: remove-duplicates-from-sorted-array.java ---
public class Solution {
    public int removeDuplicates(int[] nums) {
        if (nums.length == 0) return 0;

        int i = 0; // First element is always unique

        for (int j = 1; j < nums.length; j++) {
            if(nums[j] != nums[i]){
                i++;
                nums[i] = nums[j];
            }
        }

        return i+1;
    }
}
```

Problem 14: 287-find-the-duplicate-number

Problem Statement

Find the Duplicate Number

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive. There is only one repeated number in nums, return this repeated number. You must solve the problem without modifying the array nums and using only constant extra space.

Example 1:
Input: nums = [1,3,4,2,2]
Output: 2

Example 2:
Input: nums = [3,1,3,4,2]
Output: 3

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

Constraints:
1 <= n <= 10^5
nums.length == n + 1
1 <= nums[i] <= n
All the integers in nums appear only once except for precisely one integer which appears two or more times.

Follow up:
How can we prove that at least one duplicate number must exist in nums?
Can you solve the problem in linear runtime complexity?

Java Code

```
--- File: find-the-duplicate-number.java ---
class Solution {
    public int findDuplicate(int[] nums) {
        // Phase 1: Detect cycle
        int slow = nums[0];
        int fast = nums[0];
        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

        // Phase 2: Find entry point of cycle (the duplicate number)
        return findEntry(nums, nums[0], slow);
    }

    // Recursive function to find entry point
    private int findEntry(int[] nums, int ptr1, int ptr2) {
        if (ptr1 == ptr2) {
            return ptr1;  // duplicate found
        }
        return findEntry(nums, nums[ptr1], nums[ptr2]);  // move both pointers one step
    }
}
```

Problem 15: 289-game-of-life

Problem Statement

The Game of Life

The Game of Life, also known as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970. The board is made up of an m x n grid of cells, where each cell has an initial state: live (represented by a 1) or dead (represented by a 0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules:

1. Any live cell with fewer than two live neighbors dies as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The next state of the board is determined by applying the above rules simultaneously to every cell in the current state of the m x n grid board. In this process, births and deaths occur simultaneously.

Given the current state of the board, update the board to reflect its next state. Note that you do not need to return anything.

Example 1:
Input: board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]

Example 2:
Input: board = [[1,1],[1,0]]
Output: [[1,1],[1,1]]

Constraints:
- m == board.length
- n == board[i].length
- 1 <= m, n <= 25
- board[i][j] is 0 or 1.

Follow-up:
Could you solve it in-place? Remember that the board needs to be updated simultaneously: You cannot update some cells first and then use their updated values to update other cells. In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches upon the border of the array (i.e., live cells reach the border). How would you address these problems?

Java Code

```
--- File: game-of-life.java ---
class Solution {
    public void gameOfLife(int[][] a) {
        int m = a.length;
        int n = a[0].length;

        // Make a copy of the board so we don't overwrite while counting
        int[][] copy = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                copy[i][j] = a[i][j];
            }
        }

        // For each cell, count live neighbors
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                int count = 0;

                // check 8 neighbors
                for (int x = i - 1; x <= i + 1; x++) {
                    for (int y = j - 1; y <= j + 1; y++) {
                        if (x == i && y == j) continue; // skip itself
                        if (x >= 0 && x < m && y >= 0 && y < n) {
                            if (copy[x][y] == 1) {
                                count++;
                            }
                        }
                    }
                }
```

```
                // Apply rules
                if (copy[i][j] == 1 && count < 2) {
                    a[i][j] = 0; // Rule 1: underpopulation
                } else if (copy[i][j] == 1 && (count == 2 || count == 3)) {
                    a[i][j] = 1; // Rule 2: lives on
                } else if (copy[i][j] == 1 && count > 3) {
                    a[i][j] = 0; // Rule 3: overpopulation
                } else if (copy[i][j] == 0 && count == 3) {
                    a[i][j] = 1; // Rule 4: reproduction
                } else {
                    a[i][j] = 0; // stays dead
                }
            }
        }
    }
}
```

Problem 16: 31-next-permutation

Problem Statement

A permutation of an array of integers is an arrangement of its members into a sequence or linear order. For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1]. The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order). For example, the next permutation of arr = [1,2,3] is [1,3,2]. Similarly, the next permutation of arr = [2,3,1] is [3,1,2]. While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement.

Given an array of integers nums, find the next permutation of nums. The replacement must be in place and use only constant extra memory.

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

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

Example 3:
Input: nums = [1,1,5]
Output: [1,5,1]

Constraints:
1 < nums.length <= 100
0 <= nums[i] <= 100

Java Code

```
--- File: next-permutation.java ---
import java.util.Arrays;

class Solution {
    public void nextPermutation(int[] nums) {
        int n = nums.length;
        int i = n - 2;

        // Step 1: Find the first decreasing number from the right
        while (i >= 0 && nums[i] >= nums[i + 1]) {
            i--;
        }

        // Step 2: If we found a number, find the next larger number on the right
        if (i >= 0) {
            int j = n - 1;
```

```
            // Find a number larger than nums[i]
            while (nums[j] <= nums[i]) {
                j--;
            }

            // Swap nums[i] and nums[j]
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }

        // Step 3: Reverse the numbers to the right of i to get the next permutation
        int start = i + 1;
        int end = n - 1;
        while (start < end) {
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }
}
```

Problem 17: 414-third-maximum-number

Problem Statement

Third Maximum Number

Given an integer array nums, return the third distinct maximum number in this array. If the third maximum does not exist, return the maximum number.

Example 1:
Input: nums = [3,2,1]
Output: 1
Explanation: The first distinct maximum is 3. The second distinct maximum is 2. The third distinct maximum is 1.

Example 2:
Input: nums = [1,2]
Output: 2
Explanation: The first distinct maximum is 2. The second distinct maximum is 1. The third distinct maximum does not exist, so the maximum (2) is returned instead.

Example 3:
Input: nums = [2,2,3,1]
Output: 1
Explanation: The first distinct maximum is 3. The second distinct maximum is 2 (both 2's are counted together since they have the same value). The third distinct maximum is 1.

Constraints:
1 <= nums.length <= 10^4
-2^31 <= nums[i] <= 2^31 - 1

Follow up: Can you find an O(n) solution?

Java Code

```
--- File: third-maximum-number.java ---
class Solution {
    public int thirdMax(int[] arr) {

        Long first = Long.MIN_VALUE;
        Long second = Long.MIN_VALUE;
        Long third = Long.MIN_VALUE;
```

```
        for (int val : arr) {
            // Skip duplicates
            if (val == first || val == second || val == third) continue;

            if (val > first) {
                third = second;
                second = first;
                first = (long) val;
            } else if (val > second) {
                third = second;
                second = (long) val;
            } else if (val > third) {
                third = (long) val;
            }
        }

        if (third == Long.MIN_VALUE) {
            return first.intValue();
        } else {
            return third.intValue();
        }

    }
}
```

Problem 18: 42-trapping-rain-water

Problem Statement

Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:
Input: height = [4,2,0,3,2,5]
Output: 9

Constraints:
- n == height.length
- 1 <= n <= 2 * 10^4
- 0 <= height[i] <= 10^5

Java Code

```
--- File: trapping-rain-water.java ---
class Solution {
    public int trap(int[] height) {

        if (height == null || height.length < 3) return 0;

        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0;
        int trappedWater = 0;

        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left]; // Update left max
                } else {
```

```
                    trappedWater += leftMax - height[left]; // Water trapped at left
                }
                left++;
            } else {
                if (height[right] >= rightMax) {
                    rightMax = height[right]; // Update right max
                } else {
                    trappedWater += rightMax - height[right]; // Water trapped at right
                }
                right--;
            }
        }
        return trappedWater;

    }
}
```

Problem 19: 53-maximum-subarray

Problem Statement

Maximum Subarray

Given an integer array nums, find the subarray with the largest sum, and return its sum.

Example 1:
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.

Example 2:
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.

Example 3:
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

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

Follow up:
If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle.

Java Code

```
--- File: maximum-subarray.java ---
class Solution {
    public int maxSubArray(int[] nums) {

        int maxSum = nums[0]; int Curr = nums[0];
        for(int i =1; i<nums.length; i++){
            Curr = Math.max(nums[i], Curr + nums[i]);
            maxSum = Math.max(maxSum, Curr);
        }
        return maxSum;
    }
}
```

Problem 20: 628-maximum-product-of-three-numbers

Problem Statement
Maximum Product of Three Numbers

Given an integer array nums, find three numbers whose product is maximum and return the maximum product.

Example 1:
Input: nums = [1,2,3]
Output: 6

Example 2:
Input: nums = [1,2,3,4]
Output: 24

Example 3:
Input: nums = [-1,-2,-3]
Output: -6

Constraints:
- 3 <= nums.length <= 104
- -1000 <= nums[i] <= 1000

Java Code
```
--- File: maximum-product-of-three-numbers.java ---
class Solution {
    public int maximumProduct(int[] nums) {
        Arrays.sort(nums);
        int n = nums.length;
        int result1 = nums[n-1] * nums[n-2] * nums[n-3];
        int result2 = nums[0] * nums[1] * nums[n-1];

        return Math.max(result1, result2);

    }
}
```

Problem 21: 69-sqrtx

Problem Statement
Given a non-negative integer x, return the square root of x rounded down to the nearest integer. The returned integer should be non-negative as well. You must not use any built-in exponent function or operator. For example, do not use pow(x, 0.5) in c++ or x ** 0.5 in python.

Example 1:
Input: x = 4
Output: 2
Explanation: The square root of 4 is 2, so we return 2.

Example 2:
Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

Constraints:
0 < x <= 2^31 - 1

Java Code
```
--- File: sqrtx.java ---
class Solution {
    public int mySqrt(int x) {

        int i=0;
```

```
        int j = x;

        while(i<=j){
            int mid = (j-i)/2 +i;
            if((long)((long)mid*(long)mid)==x)return mid;
            else if((long)((long)mid*(long)mid)>x) j= mid-1;
            else i=mid+1;
        }
        return j;

    }
}
```

Problem 22: 88-merge-sorted-array

Problem Statement

Merge Sorted Array

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 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [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.

Example 2:
Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]
Explanation: The arrays we are merging are [1] and []. The result of the merge is [1].

Example 3:
Input: nums1 = [0], m = 0, nums2 = [1], n = 1
Output: [1]
Explanation: The arrays we are merging are [] and [1]. The result of the merge is [1]. Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

Constraints:
- nums1.length == m + n
- nums2.length == n
- 0 <= m, n <= 200
- 1 <= m + n <= 200
- -10^9 <= nums1[i], nums2[j] <= 10^9

Follow up: Can you come up with an algorithm that runs in O(m + n) time?

Java Code

```
--- File: merge-sorted-array.java ---
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {

        int i = m-1; int j = n-1; int k = i+j+1;
        while(i>=0 && j>=0){
            nums1[k--] = nums1[i] > nums2[j] ? nums1[i--] : nums2[j--];
        }
        while(j>=0){
            nums1[k--] = nums2[j--];
        }
```

```
    }
}
```