15. 3Sum

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`
- `-10^5 <= nums[i] <= 10^5`

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums.sort()

        for i in range(len(nums)):
            if i > 0 and nums[i] == nums[i-1]:
                continue

            j = i + 1
            k = len(nums) - 1

            while j < k:
                total = nums[i] + nums[j] + nums[k]

                if total > 0:
                    k -= 1
                elif total < 0:
                    j += 1
                else:
                    res.append([nums[i], nums[j], nums[k]])
                    j += 1

                    while nums[j] == nums[j-1] and j < k:
                        j += 1

        return res
```

---

18. 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

```
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        nums.sort()
        n = len(nums)
        result = []
        for i in  range(n):
            if i>0 and nums[i] == nums[i-1]:
                continue
            for j in range(i+1,n):
                if j>i+1 and nums[j] == nums[j-1]:
                    continue
                k = j+1
                l = n-1
                while k < l :
                    total = nums[i]+nums[j]+nums[k]+nums[l]
                    if total < target:
                        k+=1
                    elif total > target:
                        l-=1
                    else:
                        result.append((nums[i],nums[j],nums[k],nums[l]))
                        print(i,j,k,l)
                        k+=1
                        while k< n and nums[k] == nums[k-1]:
                            k+=1
        return result
```

---

26. Remove Duplicates from Sorted Array

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** (https://en.wikipedia.org/wiki/In-place_algorithm) 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];
}
```

If all assertions pass, then your solution will be **accepted**.

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 b
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
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.

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        if not nums:
            return 0

        k = 1  # Start placing unique elements from index 1

        for i in range(1, len(nums)):
            if nums[i] != nums[i - 1]:  # Found a new unique element
                nums[k] = nums[i]  # Place it at the `k` index
                k += 1  # Move `k` forward

        return k  # Number of unique elements
```

31. Next Permutation

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 (http://en.wikipedia.org/wiki/In-place_algorithm)** 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`

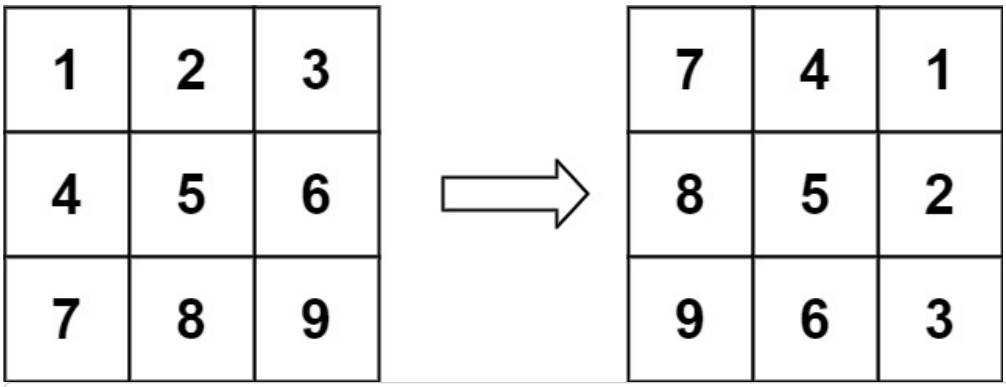
```
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)
        i = n - 2
        # find the index i until nums[i] >= nums[i+1]
        while i >= 0 and nums[i] >= nums[i+1]:
            i -= 1
        if i >= 0:
            # find the smallest element just larger than nums[i] from right
            j = n-1
            while nums[j] <= nums[i]:
                j -= 1
            nums[i], nums[j] = nums[j], nums[i]
        # Reverse subarray to the right of i
        nums[i+1:]  = nums[i+1:][::-1]
```

48. Rotate Image

You are given an `n x n` 2D `matrix` representing an image, rotate the image by **90** degrees (clockwise).

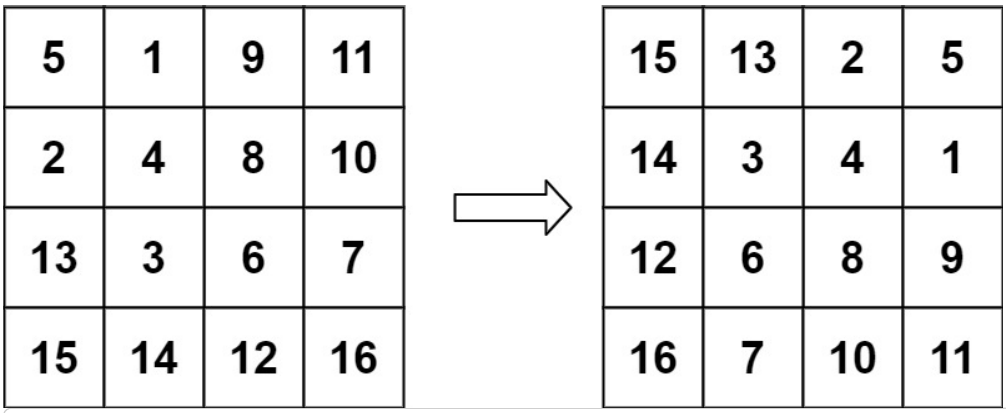
You have to rotate the image **in-place** (https://en.wikipedia.org/wiki/In-place_algorithm), which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:




```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Example 2:




```
Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
```

Constraints:

- `n == matrix.length == matrix[i].length`
- `1 <= n <= 20`
- `-1000 <= matrix[i][j] <= 1000`

```
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        n = len(matrix)
        for i in range(n):
            for j in range(i,n):
                matrix[i][j],matrix[j][i] = matrix[j][i],matrix[i][j]

        for i in range(n):
            matrix[i] = matrix[i][::-1]

        return matrix
```

53. 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.

Kadanes algo.

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        n = len(nums)
        max_current = max_global = nums[0]
        for i in range(1,n):
            max_current = max(nums[i],max_current+nums[i])
            if max_current > max_global:
                max_global = max_current
        return max_global
```

---

56. Merge Intervals

Given an array of `intervals` where `intervals[i] = [start_i, end_i]` , merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

Constraints:

- `1 <= intervals.length <= 10^4`
- `intervals[i].length == 2`
- `0 <= start_i <= end_i <= 10^4`

find better solution on chatgpt

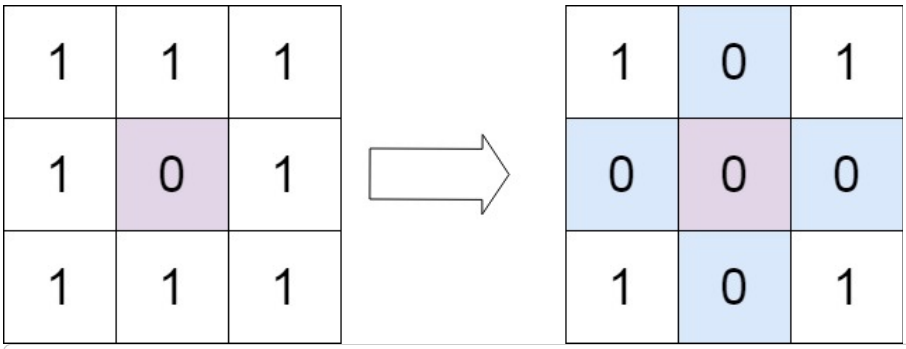
```
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        intervals.sort(key = lambda x: x[0])
        n = len(intervals)
        if n == 1:
            return intervals
        i = 1
        res = []
        while i <= n:
            start = intervals[i-1][0]
            end = intervals[i-1][1]
            while i<n and end >= intervals[i][0]:
                if end < intervals[i][1]:
                    end =  intervals[i][1]
                i+=1
            res.append((start,end))
            i+=1
        return res
```

73. Set Matrix Zeroes

Given an `m x n` integer matrix `matrix`, if an element is `0`, set its entire row and column to `0`'s.

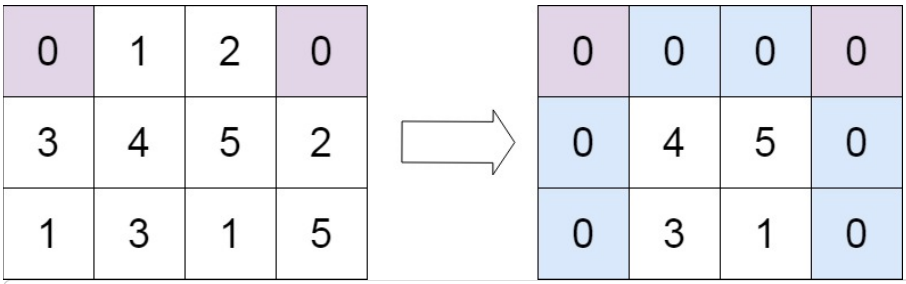
You must do it in place (https://en.wikipedia.org/wiki/In-place_algorithm).

Example 1:




```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

Example 2:




```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

Constraints:

- `m == matrix.length`
- `n == matrix[0].length`
- `1 <= m, n <= 200`
- `-2^31 <= matrix[i][j] <= 2^31 - 1`

Follow up:

- A straightforward solution using `O(mn)` space is probably a bad idea.
- A simple improvement uses `O(m + n)` space, but still not the best solution.
- Could you devise a constant space solution?

```
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        rows = len(matrix)
        cols = len(matrix[0])

        first_row_has_zero = False
        first_col_has_zero = False

        # check if the first row contains zero
        for c in range(cols):
            if matrix[0][c] == 0:
                first_row_has_zero = True
                break

        # check if the first column contains zero
        for r in range(rows):
            if matrix[r][0] == 0:
                first_col_has_zero = True
                break

        # use the first row and column as a note
        for r in range(1, rows):
            for c in range(1, cols):
                if matrix[r][c] == 0:
                    matrix[r][0] = 0
                    matrix[0][c] = 0

        # set the marked rows to zero
        for r in range(1, rows):
            if matrix[r][0] == 0:
                for c in range(1, cols):
                    matrix[r][c] = 0

        # set the marked columns to zero
        for c in range(1, cols):
            if matrix[0][c] == 0:
                for r in range(1, rows):
                    matrix[r][c] = 0

        # set the first row to zero if needed
        if first_row_has_zero:
            for c in range(cols):
                matrix[0][c] = 0

        # set the first column to zero if needed
        if first_col_has_zero:
            for r in range(rows):
```

```
                matrix[r][0] = 0

        return matrix
```

---

75. Sort Colors

Given an array `nums` with `n` objects colored red, white, or blue, sort them **in-place (https://en.wikipedia.org/wiki/In-place_algorithm)** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

Constraints:

- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` is either `0`, `1`, or `2`.

Follow up: Could you come up with a one-pass algorithm using only constant extra space?

Dutch National Flag algorithm

```
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        j = len(nums) -1
        i = 0
        k = 0
        while i<=j:
            if nums[i] == 0:
                nums[k],nums[i] = nums[i],nums[k]
                i+=1
                k+=1
            elif nums[i] == 1:
                i+=1
            else:
                nums[i],nums[j] = nums[j],nums[i]
                j-=1

        return nums
```

88. 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 nur
```

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 ensur
```

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?

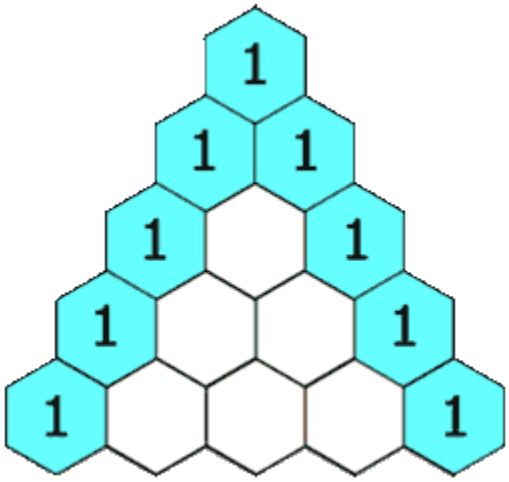
```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        i = m-1
        j = n-1
        k = m+n-1
        while j >= 0 and i>= 0:
            if nums1[i] <= nums2[j] :
                nums1[k] = nums2[j]
                j-=1
            else:
                nums1[k] = nums1[i]
                i-=1
            k-=1

        while 0 <= j:
            nums1[k] = nums2[j]
            j-=1
            k-=1
```

118. Pascal's Triangle

Given an integer `numRows`, return the first numRows of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



Example 1:

```
Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

Example 2:

```
Input: numRows = 1
Output: [[1]]
```

Constraints:

- `1 <= numRows <= 30`

```
from collections import deque
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        rows = [[1]]
        for i in range(numRows-1):
            arr = rows[-1]
            temp = [1]
            i = 0
            while i<=len(arr)-2:
                num = arr[i] + arr[i+1]
                temp.append(num)
                i+=1
            temp.append(1)
            rows.append(temp)
        return rows
```

---

121. Best Time to Buy and Sell Stock

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i`th day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5
Note that buying on day 2 and selling on day 1 is not allowed because you must buy be
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

Constraints:

- `1 <= prices.length <= 10^5`
- `0 <= prices[i] <= 10^4`

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        buy = prices[0]
        n =len(prices)
        max_profit = 0
        for i in range(1,n):
            if prices[i] - buy > max_profit:
                max_profit = prices[i] - buy
            elif prices[i] < buy:
                buy = prices[i]
        return max_profit
```

128. Longest Consecutive Sequence

Given an unsorted array of integers `nums`, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in `O(n)` time.

Example 1:

```
Input: nums = [100,4,200,1,3,2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore i
ts length is 4.
```

Example 2:

```
Input: nums = [0,3,7,2,5,8,4,6,0,1]
Output: 9
```

Example 3:

```
Input: nums = [1,0,1,2]
Output: 3
```

Constraints:

- `0 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`

```
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        num_set = set(nums)
        longest = 0

        for n in num_set:
            if n - 1 not in num_set:
                length = 1

                while n + length in num_set:
                    length += 1

                longest = max(longest, length)

        return longest
```

136. Single Number

Given a **non-empty** array of integers `nums` , every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

Input: nums = [2,2,1]

Output: 1

Example 2:

Input: nums = [4,1,2,1,2]

Output: 4

Example 3:

Input: nums = [1]

Output: 1

Constraints:

- `1 <= nums.length <= 3 * 10^4`
- `-3 * 10^4 <= nums[i] <= 3 * 10^4`
- Each element in the array appears twice except for one element which appears only once.

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        xor = 0
        for i in nums:
            xor ^= i
        return xor
```

152. 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.

---

maximum subarray for given xor

```
class Solution:
    def solve(self, A, B):
        freq = {0: 1}  # To handle cases where prefix_XOR itself is B
        prefix_XOR = 0
        count = 0
        for num in A:
            prefix_XOR ^= num  # Compute prefix XOR
            # Check if there exists a subarray ending at the current index with XOR
== B
            required_XOR = prefix_XOR ^ B
            if required_XOR in freq:
                count += freq[required_XOR]  # Add the count of such prefix_XOR occ
urrences
            # Store prefix XOR in the hashmap
            freq[prefix_XOR] = freq.get(prefix_XOR, 0) + 1
        return count
```

---

169. Majority Element

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than `⌊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?

```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        majority = cur = 0
        for i in nums:
            if majority == 0:
                cur  = i

            if cur == i:
                majority+=1
            else:
                majority-=1
        return cur
```

189. 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?

```
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)
        k= k%n
        if k!=0:
            nums[:k],nums[k:] = nums[-k:],nums[:-k]
```

---

229. Majority Element II

Given an integer array of size `n`, find all elements that appear more than `⌊ n/3 ⌋` times.

Example 1:

```
Input: nums = [3,2,3]
Output: [3]
```

Example 2:

```
Input: nums = [1]
Output: [1]
```

Example 3:

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

Constraints:

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

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

```
class Solution:
    def majorityElement(self, nums: list[int]) -> list[int]:
        # Counters for the potential majority elements
        count1 = count2 = 0
        # Potential majority element candidates
        candidate1 = candidate2 = 0

        # First pass to find potential majority elements.
        for num in nums:
            # If count1 is 0 and the current number is not equal to candidate2, upd
ate candidate1.
            if count1 == 0 and num != candidate2:
                count1 = 1
                candidate1 = num

            # If count2 is 0 and the current number is not equal to candidate1, upd
ate candidate2.
            elif count2 == 0 and num != candidate1:
                count2 = 1
                candidate2 = num

            # Update counts for candidate1 and candidate2.
            elif candidate1 == num:
                count1 += 1
            elif candidate2 == num:
                count2 += 1

            # If the current number is different from both candidates, decrement th
eir counts.
            else:
                count1 -= 1
                count2 -= 1

        result = []
        threshold = len(nums) // 3  # Threshold for majority element

        # Second pass to count occurrences of the potential majority elements.
        count1 = count2 = 0
        for num in nums:
            if candidate1 == num:
                count1 += 1
            elif candidate2 == num:
                count2 += 1

        # Check if the counts of potential majority elements are greater than n/3 a
nd add them to the result.
        if count1 > threshold:
```

```
            result.append(candidate1)
        if count2 > threshold:
            result.append(candidate2)

        return result
```

---

268. Missing Number

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

Example 1:

Input: nums = [3,0,1]

Output: 2

Explanation:

`n = 3` since there are 3 numbers, so all numbers are in the range `[0,3]`. 2 is the missing number in the range since it does not appear in `nums`.

Example 2:

Input: nums = [0,1]

Output: 2

Explanation:

`n = 2` since there are 2 numbers, so all numbers are in the range `[0,2]`. 2 is the missing number in the range since it does not appear in `nums`.

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1]

Output: 8

Explanation:

`n = 9` since there are 9 numbers, so all numbers are in the range `[0,9]`. 8 is the missing number in the range since it does not appear in `nums`.

Constraints:

- `n == nums.length`
- `1 <= n <= 10^4`
- `0 <= nums[i] <= n`
- All the numbers of `nums` are **unique**.

Follow up: Could you implement a solution using only `O(1)` extra space complexity and `O(n)` runtime complexity?

```
from typing import List

class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        n = len(nums)
        xor_all = 0
        xor_nums = 0

        # XOR all numbers from 0 to n
        for i in range(n + 1):
            xor_all ^= i

        # XOR all elements in the array
        for num in nums:
            xor_nums ^= num

        # The missing number is the remaining XOR value
        return xor_all ^ xor_nums
```

283. Move Zeroes

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

Constraints:

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

Follow up: Could you minimize the total number of operations done?

```
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)
        cnt = 0
        k=0
        for i in range(n):
            if nums[i]!=0:
                nums[k] = nums[i]
                k+=1
            else:cnt+=1

        for i in range(n-1,n-cnt-1,-1):
            nums[i] = 0
```

---

485. Max Consecutive Ones

Given a binary array `nums`, return *the maximum number of consecutive* `1`*'s in the array*.

Example 1:

```
Input: nums = [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s. The ma
```

Example 2:

```
Input: nums = [1,0,1,1,0,1]
Output: 2
```

Constraints:

- `1 <= nums.length <= 10^5`
- `nums[i]` is either `0` or `1`.

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        n = len(nums)
        max_ = 0
        s = 0
        for i in range(n):
            if nums[i] == 1:
                s+=1
            else:
                max_ = max(max_,s)
                s = 0
        return max(max_,s)
```

560. Subarray Sum Equals K

Given an array of integers `nums` and an integer `k`, return *the total number of subarrays whose sum equals to* `k`.

A subarray is a contiguous **non-empty** sequence of elements within an array.

Example 1:

```
Input: nums = [1,1,1], k = 2
Output: 2
```

Example 2:

```
Input: nums = [1,2,3], k = 3
Output: 2
```

Constraints:

- 1 <= nums.length <= $2 * 10^4$
- -1000 <= nums[i] <= 1000
- -10^7 <= k <= 10^7

Also check question that are similar to this

```
from collections import defaultdict
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        mapp = defaultdict(int)
        cur_sum = 0
        count = 0
        for num in nums:
            cur_sum+=num
            if cur_sum == k:
                count+=1
            if cur_sum - k in mapp:
                count += mapp[cur_sum - k]
            mapp[cur_sum]+=1
        return count
```

1752. Check if Array Is Sorted and Rotated

Given an array `nums`, return `true` *if the array was originally sorted in non-decreasing order, then rotated*

some *number of positions (including zero)*. Otherwise, return `false` .

There may be **duplicates** in the original array.

Note: An array `A` rotated by `x` positions results in an array `B` of the same length such that `B[i] == A[(i+x) % A.length]` for every valid index `i` .

Example 1:

```
Input: nums = [3,4,5,1,2]
Output: true
Explanation: [1,2,3,4,5] is the original sorted array.
You can rotate the array by x = 3 positions to begin on the element of value 3: [3,4,
```

Example 2:

```
Input: nums = [2,1,3,4]
Output: false
Explanation: There is no sorted array once rotated that can make nums.
```

Example 3:

```
Input: nums = [1,2,3]
Output: true
Explanation: [1,2,3] is the original sorted array.
You can rotate the array by x = 0 positions (i.e. no rotation) to make nums.
```

Constraints:

- `1 <= nums.length <= 100`
- `1 <= nums[i] <= 100`

```
class Solution:
    def check(self, nums: List[int]) -> bool:
        count = 0
        n = len(nums)

        for i in range(n):
            if nums[i] > nums[(i + 1) % n]:  # Check decreasing pair
                count += 1
                if count > 1:  # More than one break means not rotated sorted
                    return False

        return True
```

2149. Rearrange Array Elements by Sign

You are given a **0-indexed** integer array `nums` of **even** length consisting of an **equal** number of positive and negative integers.

You should return the array of nums such that the the array follows the given conditions:

1. Every **consecutive pair** of integers have **opposite signs**.
2. For all integers with the same sign, the **order** in which they were present in `nums` is **preserved**.
3. The rearranged array begins with a positive integer.

Return *the modified array after rearranging the elements to satisfy the aforementioned conditions*.

Example 1:

```
Input: nums = [3,1,-2,-5,2,-4]
Output: [3,-2,1,-5,2,-4]
Explanation:
The positive integers in nums are [3,1,2]. The negative integers are [-2,-5,-4].
The only possible way to rearrange them such that they satisfy all conditions is [3,-
Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect
```

Example 2:

```
Input: nums = [-1,1]
Output: [1,-1]
Explanation:
1 is the only positive integer and -1 the only negative integer in nums.
So nums is rearranged to [1,-1].
```

Constraints:

- `2 <= nums.length <= 2 * 10^5`
- `nums.length` is **even**
- `1 <= |nums[i]| <= 10^5`
- `nums` consists of **equal** number of positive and negative integers.

It is not required to do the modifications in-place.

```
class Solution:
    def rearrangeArray(self, nums: List[int]) -> List[int]:
        n = len(nums)
        k =i= 0
        j =1
        ans = [0]*n
        while i < n :
            if nums[i]>0:
                ans[k] = nums[i]
                k+=2
            else:
                ans[j] = nums[i]
                j+=2
            i+=1
        return ans
```
