Class Notes 2

Python

💡 Question 1

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.

TC: O(n)

SC: O (n)

```
class Solution:
    def missingNumber(self, nums):
        numSet = set(nums)

        expectedNumCount = len(nums) + 1
        for number in range(expectedNumCount):
            if number not in numSet:
                return number

        return -1
```

💡 Question 2

Given an array of intervals where intervals[i] = [starti, endi], 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].

Solution:

TC : O(nlogn)

SC : O (log n)

```
class Solution:
  def merge(self, intervals):
    intervals.sort(key=lambda x: x[0])
    merged = []
    for interval in intervals:
      if not merged or merged[-1][1] < interval[0]:
        merged.append(interval)
      else:
        merged[-1][1] = max(merged[-1][1], interval[1])
    return merged
```

💡 Question 3

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.

Solution:

TC : O(nlogn)

SC : O(log n)

```
class Solution:
  def merge(self, nums1, m, nums2, n):
    nums1Copy = nums1[:m]
    p1, p2 = 0, 0
    p = 0

    while p < m + n:
      if p2 >= n or (p1 < m and nums1Copy[p1] < nums2[p2]):
        nums1[p] = nums1Copy[p1]
        p1 += 1
      else:
        nums1[p] = nums2[p2]
        p2 += 1
      p += 1
```

💡 Question 4

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

Solution:

TC: O(nlogn)

SC : O(logn)

```
class Solution:
    def majorityElement(self, nums):
        nums.sort()
        return nums[len(nums) // 2]
```

💡 Question 5

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 uses only constant extra space.

Example 1:

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

Output: 2

TC : O(n)

SC : O(n)

```
class Solution:
    def findDuplicate(self, nums):
        seen = set()
        for num in nums:
            if num in seen:
                return num
            seen.add(num)
        return -1
```

💡 Question 6

There are many situations where we use integer values as index in array to see presence or absence. We can use bit manipulations to optimize space in such problems.

Let us consider the below problem as an example.

Given two numbers say a and b, mark the multiples of 2 and 5 between a and b and output each of the multiples.

Note : We have to mark the multiples i.e save (key, value) pairs in memory such that each key either has a value as 1 or 0 representing a multiple of 2 or 5 or not respectively.

Examples :

Input : 2 10

Output : 2 4 5 6 8 10

Input: 60 95

Output: 60 62 64 65 66 68 70 72 74 75 76 78

80 82 84 85 86 88 90 92 94 95

Solution:

TC : O (n)

SC : O(n)

```
def main():
    a = 60
    b = 70
    size = abs(b - a) + 1
    array = [0] * size

    for i in range(a, b + 1):
        if i % 2 == 0 or i % 5 == 0:
            array[i - a] = 1

    for i in range(a, b + 1):
        if array[i - a] == 1:
            print(i, end=' ')

if __name__ == '__main__':
    main()
```

💡 **Question 7**

Given an array of positive integers. We need to make the given array a 'Palindrome'. The only allowed operation is"merging" (of two adjacent elements). Merging two adjacent elements means replacing them with their sum. The task is to find the minimum number of merge operations required to make the given array a 'Palindrome'.

To make any array a palindrome, we can simply apply merge operation n-1 times where n is the size of the array (because a single-element array is always palindromic, similar to a single-character string). In that case, the size of array will be reduced to 1. But in this problem, we are asked to do it in the minimum number of operations.

Example :

Input : arr[] = {15, 4, 15}

Output : 0

Array is already a palindrome. So we do not need any merge operation.

Input : arr[] = {1, 4, 5, 1}

Output : 1

We can make given array palindrome with minimum one merging (merging 4 and 5 to make 9)

Input : arr[] = {11, 14, 15, 99}

Output : 3

We need to merge all elements to make a palindrome.

TC : O(n)

SC : O(1)

```
def findMinOps(arr, n):
  ans = 0  # Initialize result

  # Start from two corners
  i, j = 0, n - 1
  while i <= j:
     # If corner elements are the same,
     # problem reduces arr[i+1..j-1]
     if arr[i] == arr[j]:
```

```
            i += 1
            j -= 1
        # If the left element is greater, then
        # we merge the right two elements
        elif arr[i] > arr[j]:
            # Need to merge from the tail.
            j -= 1
            arr[j] += arr[j + 1]
            ans += 1
        # Else we merge the left two elements
        else:
            i += 1
            arr[i] += arr[i - 1]
            ans += 1

    return ans

# Driver method to test the above function
arr = [1, 4, 5, 9, 1]
print("Count of minimum operations is", findMinOps(arr, len(arr)))
```

-

JavaScript

💡 Question 1

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.

TC: O(n)

SC: O (n)

```
function missingNumber(nums) {
```

```
  let numSet = new Set(nums);

  let expectedNumCount = nums.length + 1;
  for (let number = 0; number < expectedNumCount; number++) {
    if (!numSet.has(number)) {
      return number;
    }
  }
  return -1;
}
```

💡 Question 2

Given an array of intervals where intervals[i] = [starti, endi], 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].

Solution:

TC : O(nlogn)

SC : O (log n)

```
function merge(intervals) {
 intervals.sort((a, b) => a[0] - b[0]);
 const merged = [];

 for (const interval of intervals) {
   if (!merged.length || merged[merged.length - 1][1] < interval[0]) {
     merged.push(interval);
   } else {
     merged[merged.length - 1][1] = Math.max(
       merged[merged.length - 1][1],
       interval[1]
     );
```

```
    }
  }
  return merged;
}
```

💡 Question 3

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.

Solution:

TC : O(nlogn)

SC : O(log n)

```
class Solution {
  merge(nums1, m, nums2, n) {
    let nums1Copy = nums1.slice(0, m);
    let p1 = 0;
    let p2 = 0;
    let p = 0;

    while (p < m + n) {
      if (p2 >= n || (p1 < m && nums1Copy[p1] < nums2[p2])) {
```

```
            nums1[p] = nums1Copy[p1];
            p1++;
        } else {
            nums1[p] = nums2[p2];
            p2++;
        }
        p++;
    }
  }
}
```

💡 Question 4

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

Solution:

TC: O(nlogn)

SC : O(logn)

```
class Solution {
  majorityElement(nums) {
    nums.sort((a, b) => a - b);
    return nums[Math.floor(nums.length / 2)];
  }
}
```

💡 **Question 5**

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 uses only constant extra space.

Example 1:

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

Output: 2

TC : O(n)

SC : O(n)

```
class Solution {
  findDuplicate(nums) {
      let seen = new Set();
      for (let num of nums) {
          if (seen.has(num)) {
              return num;
          }
          seen.add(num);
      }
      return -1;
  }
}
```

💡 Question 6

There are many situations where we use integer values as index in array to see presence or absence. We can use bit manipulations to optimize space in such problems.

Let us consider the below problem as an example.

Given two numbers say a and b, mark the multiples of 2 and 5 between a and b and output each of the multiples.

Note : We have to mark the multiples i.e save (key, value) pairs in memory such that each key either has a value as 1 or 0 representing a multiple of 2 or 5 or not respectively.

Examples :

Input : 2 10

Output : 2 4 5 6 8 10

Input: 60 95

Output: 60 62 64 65 66 68 70 72 74 75 76 78

80 82 84 85 86 88 90 92 94 95

Solution:

TC : O (n)

SC : O(n)

```
function main() {
    let a = 60;
    let b = 70;
    let size = Math.abs(b - a) + 1;
    let array = new Array(size).fill(0);

    for (let i = a; i <= b; i++) {
        if (i % 2 === 0 || i % 5 === 0) {
            array[i - a] = 1;
        }
    }

    for (let i = a; i <= b; i++) {
        if (array[i - a] === 1) {
            process.stdout.write(i + ' ');
        }
    }
}

main();
```

💡 Question 7

Given an array of positive integers. We need to make the given array a ‘Palindrome’. The only allowed operation is”merging” (of two adjacent elements). Merging two adjacent elements means replacing them with their sum. The task is to find the minimum number of merge operations required to make the given array a ‘Palindrome’.

To make any array a palindrome, we can simply apply merge operation n-1 times where n is the size of the array (because a single-element array is always palindromic, similar to a single-character string). In that case, the size of array will be reduced to 1. But in this problem, we are asked to do it in the minimum number of operations.

Example :

Input : arr[] = {15, 4, 15}

Output : 0

Array is already a palindrome. So we

do not need any merge operation.

Input : arr[] = {1, 4, 5, 1}

Output : 1

We can make given array palindrome with

minimum one merging (merging 4 and 5 to

make 9)

Input : arr[] = {11, 14, 15, 99}

Output : 3

We need to merge all elements to make

a palindrome.

TC : O(n)

SC : O(1)

```
function findMinOps(arr, n) {
  let ans = 0; // Initialize result

  // Start from two corners
  let i = 0, j = n - 1;
  while (i <= j) {
```

```
        // If corner elements are the same,
        // problem reduces arr[i+1..j-1]
        if (arr[i] === arr[j]) {
            i++;
            j--;
        }
        // If the left element is greater, then
        // we merge the right two elements
        else if (arr[i] > arr[j]) {
            // Need to merge from the tail.
            j--;
            arr[j] += arr[j + 1];
            ans++;
        }
        // Else we merge the left two elements
        else {
            i++;
            arr[i] += arr[i - 1];
            ans++;
        }
    }

    return ans;
}

// Driver method to test the above function
let arr = [1, 4, 5, 9, 1];
console.log("Count of minimum operations is", findMinOps(arr, arr.length));
```

-

Java

💡 Question 1

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.

TC: O(n)

SC: O (n)

```
class Solution {
    public int missingNumber(int[] nums) {
        Set<Integer> numSet = new HashSet<Integer>();
        for (int num : nums) numSet.add(num);

        int expectedNumCount = nums.length + 1;
        for (int number = 0; number < expectedNumCount; number++) {
            if (!numSet.contains(number)) {
                return number;
            }
        }
        return -1;
    }
}
```

💡 Question 2

Given an array of intervals where intervals[i] = [starti, endi], 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].

Solution:

TC : O(nlogn)

SC : O (log n)

```
class Solution {
```

```
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        LinkedList<int[]> merged = new LinkedList<>();
        for (int[] interval : intervals) {
            // if the list of merged intervals is empty or if the current
            // interval does not overlap with the previous, simply append it.
            if (merged.isEmpty() || merged.getLast()[1] < interval[0]) {
                merged.add(interval);
            }
        // otherwise, there is overlap, so we merge the current and previous
            // intervals.
            else {
                merged.getLast()[1] = Math.max(merged.getLast()[1], interval[1]);
            }
        }
        return merged.toArray(new int[merged.size()][]);
    }
}
```

💡 Question 3

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.

Solution:

TC : O(nlogn)

SC : O(log n)

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        // Make a copy of the first m elements of nums1.
        int[] nums1Copy = new int[m];
        for (int i = 0; i < m; i++) {
            nums1Copy[i] = nums1[i];
        }

        // Read pointers for nums1Copy and nums2 respectively.
        int p1 = 0;
        int p2 = 0;

        // Compare elements from nums1Copy and nums2 and write the smallest to nums1.
        for (int p = 0; p < m + n; p++) {
            // We also need to ensure that p1 and p2 aren't over the boundaries
            // of their respective arrays.
            if (p2 >= n || (p1 < m && nums1Copy[p1] < nums2[p2])) {
                nums1[p] = nums1Copy[p1++];
            } else {
                nums1[p] = nums2[p2++];
            }
        }
    }
}
```

💡 Question 4

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

Solution:

TC: O(nlogn)

SC : O(logn)

```
class Solution {
    public int majorityElement(int[] nums) {
        Arrays.sort(nums);
        return nums[nums.length/2];
    }
}
```

💡 **Question 5**

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 uses only constant extra space.

Example 1:

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

Output: 2

TC : O(n)

SC : O(n)

```
class Solution {
    public int findDuplicate(int[] nums) {
        Set<Integer> seen = new HashSet<Integer>();
        for (int num : nums) {
            if (seen.contains(num))
                return num;
            seen.add(num);
        }
        return -1;
    }
```

}

💡 Question 6

There are many situations where we use integer values as index in array to see presence or absence. We can use bit manipulations to optimize space in such problems.

Let us consider the below problem as an example.

Given two numbers say a and b, mark the multiples of 2 and 5 between a and b and output each of the multiples.

Note : We have to mark the multiples i.e save (key, value) pairs in memory such that each key either has a value as 1 or 0 representing a multiple of 2 or 5 or not respectively.

Examples :

Input : 2 10

Output : 2 4 5 6 8 10

Input: 60 95

Output: 60 62 64 65 66 68 70 72 74 75 76 78

80 82 84 85 86 88 90 92 94 95

Solution:

TC : O (n)

SC : O(n)

```
import java.lang.*;

class Main {

    // Driver code
    public static void main(String[] args)
    {
        int a = 60, b = 70;
        int size = Math.abs(b - a) + 1;
        int array[] = new int[size];
```

```
        // Iterate through a to b, If
        // it is a multiple of 2 or 5
        // Mark index in array as 1
        for (int i = a; i <= b; i++)
            if (i % 2 == 0 || i % 5 == 0)
                array[i - a] = 1;


        for (int i = a; i <= b; i++)
            if (array[i - a] == 1)
                System.out.printf(i + " ");
    }
}
```

💡 Question 7

Given an array of positive integers. We need to make the given array a 'Palindrome'. The only allowed operation is"merging" (of two adjacent elements). Merging two adjacent elements means replacing them with their sum. The task is to find the minimum number of merge operations required to make the given array a 'Palindrome'.

To make any array a palindrome, we can simply apply merge operation n-1 times where n is the size of the array (because a single-element array is always palindromic, similar to a single-character string). In that case, the size of array will be reduced to 1. But in this problem, we are asked to do it in the minimum number of operations.

Example :

Input : arr[] = {15, 4, 15}

Output : 0

Array is already a palindrome. So we

do not need any merge operation.

Input : arr[] = {1, 4, 5, 1}

Output : 1

We can make given array palindrome with

minimum one merging (merging 4 and 5 to

make 9)

Input : arr[] = {11, 14, 15, 99}

Output : 3

We need to merge all elements to make

a palindrome.

TC : O(n)

SC : O(1)

```
class Main {
  // Returns minimum number of count operations
  // required to make arr[] palindrome
  static int findMinOps(int[] arr, int n) {
     int ans = 0; // Initialize result

     // Start from two corners
     for (int i = 0, j = n - 1; i <= j;) {
        // If corner elements are the same,
        // problem reduces arr[i+1..j-1]
        if (arr[i] == arr[j]) {
           i++;
           j--;
        }
        // If the left element is greater, then
        // we merge the right two elements
        else if (arr[i] > arr[j]) {
           // Need to merge from the tail.
           j--;
           arr[j] += arr[j + 1];
           ans++;
        }
        // Else we merge the left two elements
        else {
           i++;
           arr[i] += arr[i - 1];
           ans++;
        }
```

```
        }

        return ans;
    }

    // Driver method to test the above function
    public static void main(String[] args) {
        int arr[] = new int[]{1, 4, 5, 9, 1};
        System.out.println("Count of minimum operations is " + findMinOps(arr, arr.length));

    }
}
```

-

Class Notes 3

Python

💡 **Question 1**

Implement pow(x, n), which calculates x raised to the power n (i.e., xn).

Example 1:

Input: x = 2.00000, n = 10

Output: 1024.00000

TC : O(log n)

SC : O(1)

```
class Solution:
    def pow(self, x, n):
        if n == 0:
            return 1
        if n < 0:
            n = -n
            x = 1 / x
```

```
    return x * self.pow(x * x, n // 2) if n % 2 else self.pow(x * x, n // 2)
```

-

  💡 **Question 2**

  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]

Time complexity: *O*(*n*). In worst case, only two scans of the whole array are needed.

Space complexity: *O*(1). No extra space is used. In place replacements are done.

```
def next_permutation(nums):
  i = len(nums) - 2
  while i >= 0 and nums[i + 1] <= nums[i]:
    i -= 1
  if i >= 0:
    j = len(nums) - 1
    while nums[j] <= nums[i]:
      j -= 1
```

```
        nums[i], nums[j] = nums[j], nums[i]
    reverse(nums, i + 1)

def reverse(nums, start):
    i, j = start, len(nums) - 1
    while i < j:
        nums[i], nums[j] = nums[j], nums[i]
        i += 1
        j -= 1
```

💡 Question 3

Given an array arr[] of distinct elements size N that is sorted and then around an unknown point, the task is to check if the array has a pair with a given sum X.

Examples :

Input: arr[] = {11, 15, 6, 8, 9, 10}, X = 16

Output: true

Explanation: There is a pair (6, 10) with sum 16

Time Complexity: O(n), where n is the length of the input array.

Space Complexity: O(1).

```
def find_pair(arr, x):
    n = len(arr)
    # find pivot element
    pivot = 0
    for i in range(n - 1):
        if arr[i] > arr[i + 1]:
            pivot = i + 1
            break
    left = pivot
    right = pivot - 1
    while left != right:
        if arr[left] + arr[right] == x:
            return True
        elif arr[left] + arr[right] < x:
            left = (left + 1) % n
        else:
```

```
            right = (right - 1 + n) % n
    return False

arr = [11, 15, 6, 8, 9, 10]
x = 16
print(find_pair(arr, x))
```

💡 Question 4

Given an array nums with n objects colored red, white, or blue, sort them in-place 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]

Solution:

TC : O(n)

SC : O(1)

```
def sort_colors(nums):
    p0 = 0
    curr = 0
    p2 = len(nums) - 1

    while curr <= p2:
        if nums[curr] == 0:
            nums[p0], nums[curr] = nums[curr], nums[p0]
            p0 += 1
            curr += 1
        elif nums[curr] == 2:
            nums[curr], nums[p2] = nums[p2], nums[curr]
            p2 -= 1
        else:
```

```
        curr += 1
```

💡 Question 5

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]

Solution:

TC: O(n)

SC: O(1)

```
class Solution {
 public void rotate(int[] nums, int k) {
   k %= nums.length;
   reverse(nums, 0, nums.length - 1);
   reverse(nums, 0, k - 1);
   reverse(nums, k, nums.length - 1);
 }
 public void reverse(int[] nums, int start, int end) {
   while (start < end) {
     int temp = nums[start];
     nums[start] = nums[end];
     nums[end] = temp;
     start++;
     end--;
   }
 }
```

}

💡 Question 6

*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 maximum number of consecutive 1s is 3.

Solution:

TC : O(n)

SC : O(1)

```
class Solution {
 public int findMaxConsecutiveOnes(int[] nums) {
   int count = 0;
   int maxCount = 0;
   for(int i = 0; i < nums.length; i++) {
     if(nums[i] == 1) {
       // Increment the count of 1's by one.
       count += 1;
     } else {
       // Find the maximum till now.
       maxCount = Math.max(maxCount, count);
       // Reset count of 1.
       count = 0;
     }
   }
   return Math.max(maxCount, count);
 }
}
```

-

JavaScript

💡 Question 1

Implement pow(x, n), which calculates x raised to the power n (i.e., xn).

Example 1:

Input: x = 2.00000, n = 10

Output: 1024.00000

TC : O(log n)

SC : O(1)

```
function pow(x, n) {
 if (n === 0) {
   return 1;
 }
 if (n < 0) {
   n = -n;
   x = 1 / x;
 }
 return n % 2
   ? x * pow(x * x, Math.floor(n / 2))
   : pow(x * x, Math.floor(n / 2));
}
```

-

💡 Question 2

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]

Time complexity: *O*(*n*). In worst case, only two scans of the whole array are needed.

Space complexity: *O*(1). No extra space is used. In place replacements are done.

```
function nextPermutation(nums) {
    let i = nums.length - 2;
    while (i >= 0 && nums[i + 1] <= nums[i]) {
        i--;
    }
    if (i >= 0) {
        let j = nums.length - 1;
        while (nums[j] <= nums[i]) {
            j--;
        }
        swap(nums, i, j);
    }
    reverse(nums, i + 1);
}

function reverse(nums, start) {
    let i = start, j = nums.length - 1;
    while (i < j) {
        swap(nums, i, j);
        i++;
        j--;
    }
}
```

```
function swap(nums, i, j) {
    let temp = nums[i];
    nums[i] = nums[j];
    nums[j] = temp;
}
```

💡 Question 3

Given an array arr[] of distinct elements size N that is sorted and then around an unknown point, the task is to check if the array has a pair with a given sum X.

Examples :

Input: arr[] = {11, 15, 6, 8, 9, 10}, X = 16

Output: true

Explanation: There is a pair (6, 10) with sum 16

Time Complexity: O(n), where n is the length of the input array.

Space Complexity: O(1).

```
function findPair(arr, x) {
    const n = arr.length;
    // find pivot element
    let pivot = 0;
    for (let i = 0; i < n - 1; i++) {
        if (arr[i] > arr[i + 1]) {
            pivot = i + 1;
            break;
        }
    }
    let left = pivot;
    let right = pivot - 1;
    while (left !== right) {
        if (arr[left] + arr[right] === x) {
            return true;
        } else if (arr[left] + arr[right] < x) {
            left = (left + 1) % n;
        } else {
            right = (right - 1 + n) % n;
```

```
      }
    }
    return false;
}

const arr = [11, 15, 6, 8, 9, 10];
const x = 16;
console.log(findPair(arr, x));
```

💡 Question 4

Given an array nums with n objects colored red, white, or blue, sort them in-place 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]

Solution:

TC : O(n)

SC : O(1)

```
function sortColors(nums) {
    let p0 = 0;
    let curr = 0;
    let p2 = nums.length - 1;

    while (curr <= p2) {
        if (nums[curr] === 0) {
            [nums[p0], nums[curr]] = [nums[curr], nums[p0]];
            p0++;
            curr++;
        } else if (nums[curr] === 2) {
            [nums[curr], nums[p2]] = [nums[p2], nums[curr]];
```

```
            p2--;
        } else {
            curr++;
        }
    }
}
```

💡 Question 5

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]

Solution:

TC: O(n)

SC: O(1)

```
function rotate(nums, k) {
    k %= nums.length;
    reverse(nums, 0, nums.length - 1);
    reverse(nums, 0, k - 1);
    reverse(nums, k, nums.length - 1);
}

function reverse(nums, start, end) {
    while (start < end) {
        const temp = nums[start];
        nums[start] = nums[end];
```

```
        nums[end] = temp;
        start++;
        end--;
    }
}
```

💡 Question 6

*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 maximum number of consecutive 1s is 3.

Solution:

TC : O(n)

SC : O(1)

```
function findMaxConsecutiveOnes(nums) {
    let count = 0;
    let maxCount = 0;
    for (let i = 0; i < nums.length; i++) {
        if (nums[i] === 1) {
            count += 1;
        } else {
            maxCount = Math.max(maxCount, count);
            count = 0;
        }
    }
    return Math.max(maxCount, count);
}
```

-

Java

💡 Question 1

Implement pow(x, n), which calculates x raised to the power n (i.e., xn).

Example 1:

Input: x = 2.00000, n = 10

Output: 1024.00000

TC : O(log n)

SC : O(1)

```
public class Solution {
   public double pow(double x, int n) {
      if(n == 0)
         return 1;
      if(n<0){
         n = -n;
         x = 1/x;
      }
      return (n%2 == 0) ? pow(x*x, n/2) : x*pow(x*x, n/2);
   }
}
```

-

💡 Question 2

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]

Time complexity: *O*(*n*). In worst case, only two scans of the whole array are needed.

Space complexity: *O*(1). No extra space is used. In place replacements are done.

```
public class Solution {
  public void nextPermutation(int[] nums) {
    int i = nums.length - 2;
    while (i >= 0 && nums[i + 1] <= nums[i]) {
      i--;
    }
    if (i >= 0) {
      int j = nums.length - 1;
      while (nums[j] <= nums[i]) {
        j--;
      }
      swap(nums, i, j);
    }
    reverse(nums, i + 1);
  }

  private void reverse(int[] nums, int start) {
    int i = start, j = nums.length - 1;
    while (i < j) {
      swap(nums, i, j);
      i++;
      j--;
    }
  }

  private void swap(int[] nums, int i, int j) {
    int temp = nums[i];
```

```
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

💡 Question 3

Given an array arr[] of distinct elements size N that is sorted and then around an unknown point, the task is to check if the array has a pair with a given sum X.

Examples :

Input: arr[] = {11, 15, 6, 8, 9, 10}, X = 16

Output: true

Explanation: There is a pair (6, 10) with sum 16

Time Complexity: O(n), where n is the length of the input array.

Space Complexity: O(1).

```
public class FindPairInRotatedArray {
    public static boolean findPair(int[] arr, int x) {
        int n = arr.length;
        // find pivot element
        int pivot = 0;
        for (int i = 0; i < n-1; i++) {
            if (arr[i] > arr[i+1]) {
                pivot = i+1;
                break;
            }
        }
        int left = pivot;
        int right = pivot-1;
        while (left != right) {
            if (arr[left] + arr[right] == x) {
                return true;
            }
            else if (arr[left] + arr[right] < x) {
                left = (left+1) % n;
            }
            else {
```

```
                right = (right-1+n) % n;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        int[] arr = {11, 15, 6, 8, 9, 10};
        int x = 16;
        System.out.println(findPair(arr, x));
    }
}
```

💡 Question 4

Given an array nums with n objects colored red, white, or blue, sort them in-place 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]

Solution:

TC : O(n)

SC : O(1)

```
class Solution {
 /*
 Dutch National Flag problem solution.
 */
 public void sortColors(int[] nums) {
   // for all idx < i : nums[idx < i] = 0
   // j is an index of element under consideration
   int p0 = 0, curr = 0;
```

```
    // for all idx > k : nums[idx > k] = 2
    int p2 = nums.length - 1;

    int tmp;
    while (curr <= p2) {
      if (nums[curr] == 0) {
        // swap p0-th and curr-th elements
        // i++ and j++
        tmp = nums[p0];
        nums[p0++] = nums[curr];
        nums[curr++] = tmp;
      }
      else if (nums[curr] == 2) {
        // swap k-th and curr-th elements
        // p2--
        tmp = nums[curr];
        nums[curr] = nums[p2];
        nums[p2--] = tmp;
      }
      else curr++;
    }
  }
}
```

💡 Question 5

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]

Solution:

TC: O(n)

SC: O(1)

```
class Solution {
 public void rotate(int[] nums, int k) {
   k %= nums.length;
   reverse(nums, 0, nums.length - 1);
   reverse(nums, 0, k - 1);
   reverse(nums, k, nums.length - 1);
 }
 public void reverse(int[] nums, int start, int end) {
   while (start < end) {
     int temp = nums[start];
     nums[start] = nums[end];
     nums[end] = temp;
     start++;
     end--;
   }
 }
}
```

💡 Question 6

*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 maximum number of consecutive 1s is 3.

Solution:

TC : O(n)

SC : O(1)

```
class Solution {
 public int findMaxConsecutiveOnes(int[] nums) {
   int count = 0;
   int maxCount = 0;
   for(int i = 0; i < nums.length; i++) {
     if(nums[i] == 1) {
       // Increment the count of 1's by one.
       count += 1;
     } else {
       // Find the maximum till now.
       maxCount = Math.max(maxCount, count);
       // Reset count of 1.
       count = 0;
     }
   }
   return Math.max(maxCount, count);
 }
}
```

-

Cpp

💡 Question 1 Implement pow(x, n), which calculates x raised to the power n (i.e., xn).

Example 1:

Input: x = 2.00000, n = 10

Output: 1024.00000

TC : O(log n)

SC : O(1)

```
class Solution {
public:
   double myPow(double x, int n) {
       if (n == 0)
           return 1;
       double half = myPow(x, n / 2);
       if (n % 2 == 0)
```

```
            return half * half;
        else if (n > 0)
            return half * half * x;
        else
            return half * half / x;
    }
};
```

- 💡 Question 2

  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).

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]

Time complexity: O(n). In worst case, only two scans of the whole array are needed.

Space complexity: O(1). No extra space is used. In place replacements are done.

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int n = nums.size(), i, j;
        for (i = n - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) {
                for (j = n - 1; j > i; j--) {
```

```
                if (nums[j] > nums[i]) {
                    break;
                }
            }
            swap(nums[i], nums[j]);
            reverse(nums.begin() + i + 1, nums.end());
            return;
        }
    }
    reverse(nums.begin(), nums.end());
  }
};
```

💡 Question 3

Given an array arr[] of distinct elements size N that is sorted and then around an unknown point, the task is to check if the array has a pair with a given sum X.

Examples :

Input: arr[] = {11, 15, 6, 8, 9, 10}, X = 16

Output: true

Explanation: There is a pair (6, 10) with sum 16

Time Complexity: O(n), where n is the length of the input array.

Space Complexity: O(1).

```
#include <bits/stdc++.h>
using namespace std;

bool findPair(int arr[], int n, int x) {
  int i;
  for (i = 0; i < n - 1; i++)
    if (arr[i] > arr[i + 1])
      break;
  int l = (i + 1) % n, r = i;

  while (l != r) {
    if (arr[l] + arr[r] == x)
```

return

Question 3

Given an array arr[] of distinct elements size N that is sorted and then around an unknown point, the task is to check if the array has a pair with a given sum X.

Examples :

Input: arr[] = {11, 15, 6, 8, 9, 10}, X = 16

Output: true

Explanation: There is a pair (6, 10) with sum 16

Time Complexity: O(n), where n is the length of the input array.

Space Complexity: O(1).

```cpp
#include <bits/stdc++.h>
using namespace std;

bool findPair(int arr[], int n, int x) {
    int i;
    for (i = 0; i < n - 1; i++)
        if (arr[i] > arr[i + 1])
            break;
    int l = (i + 1) % n, r = i;

    while (l != r) {
        if (arr[l] + arr[r] == x)
            return true;
        if (arr[l] + arr[r] < x)
            l = (l + 1) % n;
        else
            r = (n + r - 1) % n;
    }
    return false;
}
```

💡 Question 4

Given an array nums with n objects colored red, white, or blue, sort them **in-place** 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]

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size()-1;
        while(mid <= high){
            if(nums[mid] == 0){
                swap(nums[low], nums[mid]);
                low++;
                mid++;
            }
            else if(nums[mid] == 1){
                mid++;
            }
            else{
                swap(nums[mid], nums[high]);
                high--;
            }
        }
    }
};
```

💡 Question 5

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]

Solution:

TC: O(n)

SC: O(1)

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        k %= n;
        reverse(nums.begin(), nums.end());
        reverse(nums.begin(), nums.begin() + k);
        reverse(nums.begin() + k, nums.end());
    }
};
```

💡 Question 6

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 maximum number of consecutive 1s is 3.

Solution:

TC : O(n)

SC : O(1)

```
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int count = 0;
        int maxCount = 0;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] == 1) {
                // Increment the count of 1's by one.
                count += 1;
            } else {
                // Find the maximum till now.
                maxCount = max(maxCount, count);
                // Reset count of 1.
                count = 0;
            }
        }
        return max(maxCount, count);
    }
};
```

-

Class Notes 4

Python

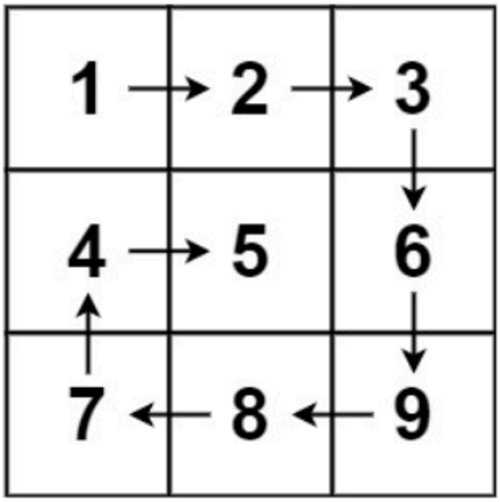
💡 **Question 1**

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

Example 1:

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

Output: [1,2,3,6,9,8,7,4,5]



Solution:

TC: O(n)

SC: O(1)

```
def spiralOrder(matrix):
  result = []
  rows = len(matrix)
  columns = len(matrix[0])
  up = 0
  left = 0
  right = columns - 1
  down = rows - 1

  while len(result) < rows * columns:
    # Traverse from left to right.
    for col in range(left, right + 1):
      result.append(matrix[up][col])
    # Traverse downwards.
```

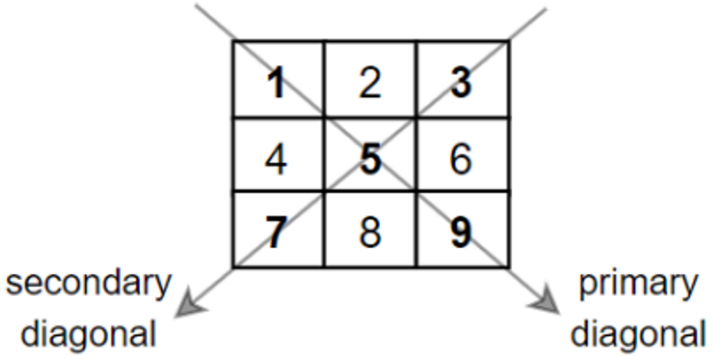
```
        for row in range(up + 1, down + 1):
            result.append(matrix[row][right])
        # Make sure we are now on a different row.
        if up != down:
            # Traverse from right to left.
            for col in range(right - 1, left - 1, -1):
                result.append(matrix[down][col])
        # Make sure we are now on a different column.
        if left != right:
            # Traverse upwards.
            for row in range(down - 1, up, -1):
                result.append(matrix[row][left])
        left += 1
        right -= 1
        up += 1
        down -= 1

    return result
```

💡 Question 2

Given a square matrix mat, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:



Input: mat = [[1,2,3],

[4,5,6],

[7,8,9]]

Output: 25

Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25

Notice that element mat[1][1] = 5 is counted only once.

Solution:

TC: O(m+n)

SC: O(1)

```
def diagonalSum(mat):
    n = len(mat)
    ans = 0

    for i in range(n):
        # Add elements from primary diagonal.
        ans += mat[i][i]
        # Add elements from secondary diagonal.
        ans += mat[n - 1 - i][i]

    # If n is odd, subtract the middle element as it's added twice.
    if n % 2 != 0:
        ans -= mat[n // 2][n // 2]

    return ans
```

💡 Question 3

Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return *the number of negative numbers in* grid.

Example 1:

Input: grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

Output: 8

Explanation: There are 8 negatives number in the matrix.

TC: O(m*n)

SC : O(1)

```
class Solution:
  def countNegatives(self, grid):
      count = 0
      n = len(grid[0])
      currRowNegativeIndex = n - 1

      for row in grid:
          while currRowNegativeIndex >= 0 and row[currRowNegativeIndex] < 0:
              currRowNegativeIndex -= 1
          count += (n - (currRowNegativeIndex + 1))

      return count
```

💡 Question 4

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return *the wealth that the richest customer has.*

A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Example 1:

Input: accounts = [[1,2,3],[3,2,1]]

Output: 6

Explanation:

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

Solution:

TC: O(m*n)

SC : O(1)

```
class Solution:
  def maximumWealth(self, accounts):
    maxWealthSoFar = 0

    for account in accounts:
      currCustomerWealth = sum(account)
      maxWealthSoFar = max(maxWealthSoFar, currCustomerWealth)

    return maxWealthSoFar
```

-

JavaScript

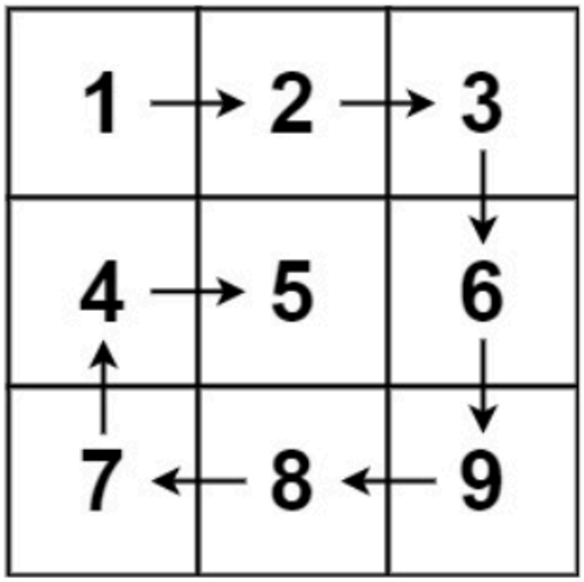
💡 **Question 1**

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

Example 1:

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

Output: [1,2,3,6,9,8,7,4,5]



Solution:

TC: O(n)

SC: O(1)

```
function spiralOrder(matrix) {
  const result = [];
  const rows = matrix.length;
  const columns = matrix[0].length;
  let up = 0;
  let left = 0;
  let right = columns - 1;
  let down = rows - 1;

  while (result.length < rows * columns) {
    // Traverse from left to right.
    for (let col = left; col <= right; col++) {
      result.push(matrix[up][col]);
    }
    // Traverse downwards.
    for (let row = up + 1; row <= down; row++) {
      result.push(matrix[row][right]);
    }
    // Make sure we are now on a different row.
    if (up !== down) {
      // Traverse from right to left.
```

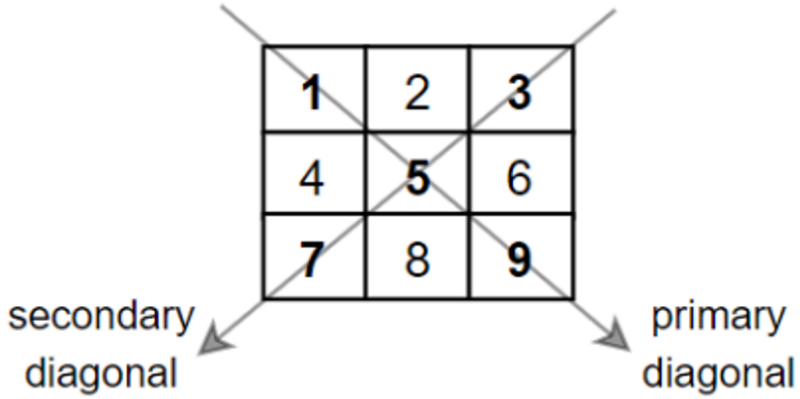
```
            for (let col = right - 1; col >= left; col--) {
                result.push(matrix[down][col]);
            }
        }
        // Make sure we are now on a different column.
        if (left !== right) {
            // Traverse upwards.
            for (let row = down - 1; row > up; row--) {
                result.push(matrix[row][left]);
            }
        }
        left++;
        right--;
        up++;
        down--;
    }

    return result;
}
```

💡 Question 2

Given a square matrix mat, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:



Input: mat = [[1,2,3],

[4,5,6],

[7,8,9]]

Output: 25

Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25

Notice that element mat[1][1] = 5 is counted only once.

Solution:

TC: O(m+n)

SC: O(1)

```
function diagonalSum(mat) {
  const n = mat.length;
  let ans = 0;

  for (let i = 0; i < n; i++) {
    // Add elements from primary diagonal.
    ans += mat[i][i];
    // Add elements from secondary diagonal.
```

```
        ans += mat[n - 1 - i][i];
    }

    // If n is odd, subtract the middle element as it's added twice.
    if (n % 2 !== 0) {
        ans -= mat[Math.floor(n / 2)][Math.floor(n / 2)];
    }
    return ans;
}
```

💡 Question 3

Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return *the number of negative numbers in* grid.

Example 1:

Input: grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

Output: 8

Explanation: There are 8 negatives number in the matrix.

TC: O(m*n)

SC : O(1)

```
class Solution {
    countNegatives(grid) {
        let count = 0;
        const n = grid[0].length;
        let currRowNegativeIndex = n - 1;

    for (let row of grid) {
        while (currRowNegativeIndex >= 0 && row[currRowNegativeIndex] < 0) {
                currRowNegativeIndex--;
            }
            count += (n - (currRowNegativeIndex + 1));
        }

        return count;
    }
}
```

💡 Question 4

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return *the wealth that the richest customer has.*

A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Example 1:

Input: accounts = [[1,2,3],[3,2,1]]

Output: 6

Explanation:

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

Solution:

TC: O(m*n)

SC : O(1)

```
class Solution {
  maximumWealth(accounts) {
    let maxWealthSoFar = 0;

    for (let account of accounts) {
      let currCustomerWealth = account.reduce((acc, curr) => acc + curr, 0);
      maxWealthSoFar = Math.max(maxWealthSoFar, currCustomerWealth);
    }

    return maxWealthSoFar;
  }
}
```

-

Java

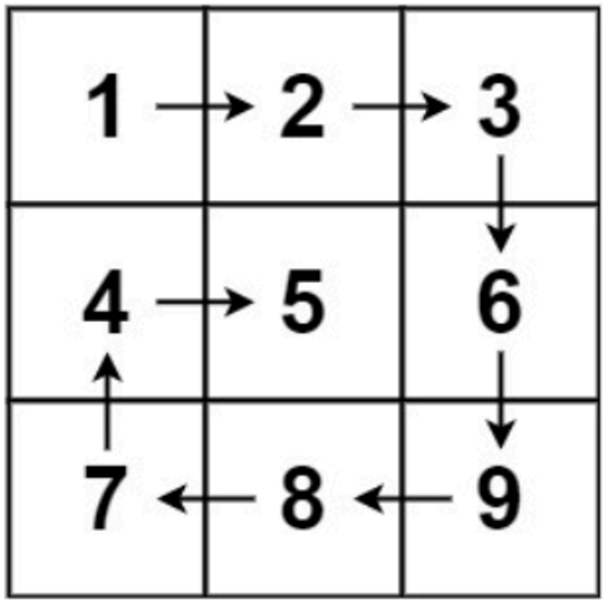
💡 **Question 1**

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

Example 1:

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

Output: [1,2,3,6,9,8,7,4,5]



Solution:

TC: O(n)

SC: O(1)

```
class Solution {
   public List<Integer> spiralOrder(int[][] matrix) {
       List<Integer> result = new ArrayList<>();
       int rows = matrix.length;
       int columns = matrix[0].length;
```

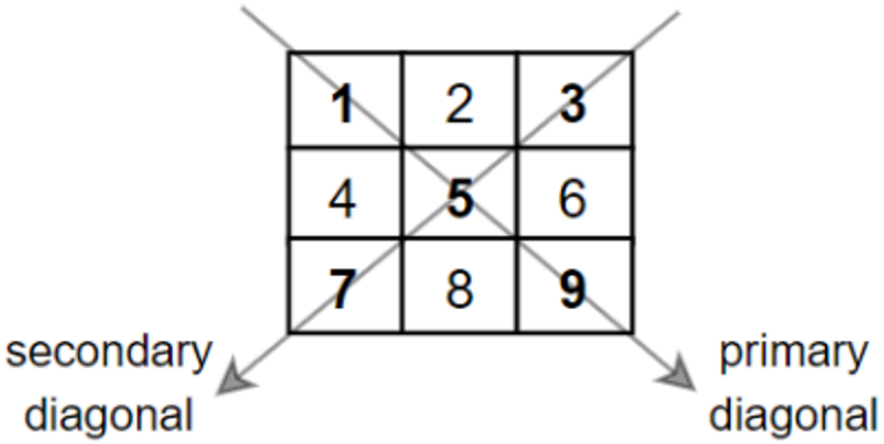
```
        int up = 0;
        int left = 0;
        int right = columns - 1;
        int down = rows - 1;


        while (result.size() < rows * columns) {
            // Traverse from left to right.
            for (int col = left; col <= right; col++) {
                result.add(matrix[up][col]);
            }
            // Traverse downwards.
            for (int row = up + 1; row <= down; row++) {
                result.add(matrix[row][right]);
            }
            // Make sure we are now on a different row.

                // Traverse from right to left.
                for (int col = right - 1; col >= left; col--) {
                    result.add(matrix[down][col]);
                }
                // Traverse upwards.
                for (int row = down - 1; row > up; row--) {
                    result.add(matrix[row][left]);
                }
            left++;
            right--;
            up++;
            down--;
        }
        return result;
    }
}
```

💡 **Question 2**

Given a square matrix mat, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:



Input: mat = [[1,2,3],

[4,5,6],

[7,8,9]]

Output: 25

Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25

Notice that element mat[1][1] = 5 is counted only once.

Solution:

TC: O(m+n)

SC: O(1)

```
class Solution {
  public int diagonalSum(int[][] mat) {
      int n = mat.length;
      int ans = 0;

      for (int i = 0; i < n; i++) {
```

```
            // Add elements from primary diagonal.
            ans += mat[i][i];
            // Add elements from secondary diagonal.
            ans += mat[n - 1 - i][i];
        }

        // If n is odd, subtract the middle element as it's added twice.
        if (n % 2 != 0) {
            ans -= mat[n / 2][n / 2];
        }

        return ans;
    }
}
```

💡 Question 3

Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return *the number of negative numbers in* grid.

Example 1:

Input: grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

Output: 8

Explanation: There are 8 negatives number in the matrix.

TC: O(m*n)

SC : O(1)

```
class Solution {
    public int countNegatives(int[][] grid) {
        int count = 0;
    int n = grid[0].length;
    int currRowNegativeIndex = n - 1;


    // Iterate on all rows of the matrix one by one.
    for (int[] row : grid) {
        // Decrease 'currRowNegativeIndex' so that it points to current row's last positive element.
        while (currRowNegativeIndex >= 0 && row[currRowNegativeIndex] < 0) {
```

```
            currRowNegativeIndex--;
        }
        // 'currRowNegativeIndex' points to the last positive element,
        // which means 'n - (currRowNegativeIndex + 1)' is the number of all negative elements.
        count += (n - (currRowNegativeIndex + 1));
    }
    return count;
    }
}
```

💡 Question 4

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return *the wealth that the richest customer has.*

A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Example 1:

Input: accounts = [[1,2,3],[3,2,1]]

Output: 6

Explanation:

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

Solution:

TC: O(m*n)

SC : O(1)

```
class Solution {
    public int maximumWealth(int[][] accounts) {
        // Initialize the maximum wealth seen so far to 0 (the minimum wealth possible)
        int maxWealthSoFar = 0;
```

```
        // Iterate over accounts
        for (int[] account : accounts) {
            // For each account, initialize the sum to 0
            int currCustomerWealth = 0;
            // Add the money in each bank
            for (int money : account) {
                currCustomerWealth += money;
            }
            // Update the maximum wealth seen so far if the current wealth is greater
            // If it is less than the current sum
            maxWealthSoFar = Math.max(maxWealthSoFar, currCustomerWealth);
        }

        // Return the maximum wealth
        return maxWealthSoFar;
    }
}
```

-

Cpp

💡 **Question 1**

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

Example 1:

https://lh5.googleusercontent.com/Oizk5nIieWitNwfvhjLdG9Hpaip-CKuBZXW6ohgUmqSXWSQA-9nc1m04ssQ5Mxyb-AP8dEInhNztTJPFSiNcX1OQqzARfeBie9JHpvOf09TsRs2DIgzyRLHJvVgzhvRb70sshVsW_3qt41Pvo6bMrcU

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

Output: [1,2,3,6,9,8,7,4,5]

Example 2:

https://lh6.googleusercontent.com/thZju1ZdPEsT3gOaKzHBNjWY-agE6c_jQf6LPjFA1dpHmUCZkOJFMwtIgUNUrDhyIfIL-8_bUUKfTwptx1tr4vOWmEkBmHokMJ9k9CC4k_CauvZ17a6RE5-T30JcVLLYFI4ta7uV9Y3Z-wNEfU9X2K4

Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

Output: [1,2,3,4,8,12,11,10,9,5,6,7]

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
      vector<int> ans;
      int m=matrix.size();
   int l=0;
   int n=matrix[0].size();
   int k=0;
   while(k<n && l<m)
   {
      for(int i=k;i<n;i++)
      {
         ans.push_back(matrix[l][i]);
      }
      l++;
      for(int i=l;i<m;i++)
      {
         ans.push_back(matrix[i][n-1]);
      }
      n--;
      if(l<m)
      {
         for(int i=n-1;i>=k;i--)
         {
            ans.push_back(matrix[m-1][i]);
         }
         m--;
      }
      if(k<n)
      {
         for(int i=m-1;i>=l;i--)
         {
            ans.push_back(matrix[i][k]);
         }
         k++;
      }
   }
   return ans;
   }
```

💡 **Question 2**

Given an n x n matrix mat, return the sum of all elements of the matrix diagonally from the upper left to the lower right (i.e., sum(mat[i][i]) for all i).

Example 1:

Input: mat = [[1,2,3],

[4,5,6],

[7,8,9]]

Output: 25

Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25

Notice that element mat[1][1] = 5 is counted only once.

Solution:

TC: O(n)

SC: O(1)

```
class Solution {
public:
    int diagonalSum(vector<vector<int>>& mat) {
        int n = mat.size();
        int ans = 0;

        for (int i = 0; i < n; i++) {
            // Add elements from primary diagonal.
            ans += mat[i][i];
            // Add elements from secondary diagonal.
            ans += mat[n - 1 - i][i];
        }

        // If n is odd, subtract the middle element as it's added twice.
        if (n % 2 != 0) {
            ans -= mat[n / 2][n / 2];
        }
```

```
        return ans;
    }
};
```

💡 **Question 3**

Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return the number of negative numbers in grid.

Example 1:

Input: grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

Output: 8

Explanation: There are 8 negatives number in the matrix.

TC: O(m*n)

SC : O(1)

```
class Solution {
public:
    int countNegatives(vector<vector<int>>& grid) {
        int count = 0;
        int n = grid[0].size();
        int currRowNegativeIndex = n - 1;

        for (vector<int>& row : grid) {
            // Decrease 'currRowNegativeIndex' so that it points to current row's last positive element.
            while (currRowNegativeIndex >= 0 && row[currRowNegativeIndex] < 0) {
                currRowNegativeIndex--;
            }
            // 'currRowNegativeIndex' points to the last positive element,
            // which means 'n - (currRowNegativeIndex + 1)' is the number of all negative elements.
            count += (n - (currRowNegativeIndex + 1));
        }
        return count;
```

💡 **Question 4**

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return *the wealth that the richest customer has.*

A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Example 1:

Input: accounts = [[1,2,3],[3,2,1]]

Output: 6

Explanation:

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

Solution:

TC: O(m*n)

SC : O(1)

```
class Solution {
public:
    int maximumWealth(vector<vector<int>>& accounts) {
        // Initialize the maximum wealth seen so far to 0 (the minimum wealth possible)
        int maxWealthSoFar = 0;

        // Iterate over accounts
        for (vector<int>& account : accounts) {
            // For each account, initialize the sum to 0
            int currCustomerWealth = 0;
            // Add the money in each bank
            for (int money : account) {
                currCustomerWealth += money;
```

```
            }
            // Update the maximum wealth seen so far if the current wealth is greater
            // If it is less than the current sum
            maxWealthSoFar = max(maxWealthSoFar, currCustomerWealth);
        }

        // Return the maximum wealth
        return maxWealthSoFar;
    }
};
```

-