Problem 1.Given an array of n numbers, give an algorithm which gives the element appearing maximum number of times?

```
def find_max_frequency_element(arr):
    frequency_map = {}

    # Count occurrences of each element
    for num in arr:
        frequency_map[num] = frequency_map.get(num, 0) + 1

    # Find element with maximum occurrences
    max_element = None
    max_count = 0
    for key, value in frequency_map.items():
        if value > max_count:
            max_element = key
            max_count = value

    return max_element, max_count

# Example usage
arr = [1, 3, 2, 3, 4, 1, 3, 1, 1, 2, 3]
result = find_max_frequency_element(arr)
print(f"Element with max frequency: {result[0]}, Count: {result[1]}")
```

```
Element with max frequency: 1, Count: 4
```

Problem 2 : We are given a list of n-1 integers and these integers are in the range of 1 to n . There are no duplicates in the list. One of the integers is missing in the list. Give an algorithm to find that element Ex: [1,2,4,6,3,7,8] 5 is the missing num.

```
def find_missing_number_xor(arr, n):
    xor_all = 0
    xor_arr = 0

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

    # XOR all elements in the array
    for num in arr:
        xor_arr ^= num

    # The missing number is the XOR difference
    return xor_all ^ xor_arr

# Example usage
arr = [1, 2, 4, 6, 3, 7, 8]
n = len(arr) + 1  # Since one number is missing
print("Missing number:", find_missing_number_xor(arr, n))
```

```
Missing number: 5
```

Problem 3 : Given an array of n positive numbers. All numbers occurs even number of times except 1 which occurs odd number of times. Find that number in O(n) time and O(1) space. Ex: [1,2,3,2,3,1,3]. 3 is repeats odd times.

```
def find_odd_occurrence(arr):
    result = 0
    for num in arr:
        result ^= num  # XOR all elements
    return result

# Example usage
arr = [1, 2, 3, 2, 3, 1, 3]
print("Number occurring odd times:", find_odd_occurrence(arr))
```

```
Number occurring odd times: 3
```

Problem 4 : Given an array of n elements. Find two elements in the array such that their sum is equal to given element K.

```
def find_pair_with_sum(arr, K):
    num_set = set()

    for num in arr:
```

```
            complement = K - num
            if complement in num_set:
                return num, complement
            num_set.add(num)

        return None  # No pair found

    # Example usage
    arr = [1, 4, 6, 8, 10, 45]
    K = 16
    result = find_pair_with_sum(arr, K)
    print("Pair with sum", K, ":", result)
```

```
Pair with sum 16 : (10, 6)
```

Double-click (or enter) to edit

Problem 5 : Given an array of both positive and negative numbers, find two numbers such that their sum is closest to 0. Ex: [1 ,60 ,-10, 70, -80,85]. Ans : -80,85.

```
def find_closest_to_zero_pair(arr):
    arr.sort()  # Sort the array (O(n log n))
    left, right = 0, len(arr) - 1
    min_sum = float('inf')
    closest_pair = (None, None)

    while left < right:
        curr_sum = arr[left] + arr[right]

        # Update closest pair if current sum is closer to zero
        if abs(curr_sum) < abs(min_sum):
            min_sum = curr_sum
            closest_pair = (arr[left], arr[right])

        # Move pointers
        if curr_sum < 0:
            left += 1
        else:
            right -= 1

    return closest_pair

# Example usage
arr = [1, 60, -10, 70, -80, 85]
result = find_closest_to_zero_pair(arr)
print("Pair closest to zero:", result)
```

```
Pair closest to zero: (-80, 85)
```

Problem 6 : Given an array of n elements . Find three elements such that their sum is equal to the given number.

```
def find_three_numbers_with_sum(arr, K):
    arr.sort()  # Sort the array (O(n log n))
    n = len(arr)

    for i in range(n - 2):  # Fix one element
        left, right = i + 1, n - 1  # Two-pointer approach

        while left < right:
            curr_sum = arr[i] + arr[left] + arr[right]

            if curr_sum == K:
                return arr[i], arr[left], arr[right]

            if curr_sum < K:
                left += 1  # Increase sum by moving left forward
            else:
                right -= 1  # Decrease sum by moving right backward

    return None  # No triplet found

# Example usage
arr = [1, 4, 6, 8, 10, 45]
K = 22
result = find_three_numbers_with_sum(arr, K)
```

```
print("Triplet with sum", K, ":", result)
```

Problem 7 : Given an array of n elements . Find three elements i, j, k in the array such that i * i + j * j = k*k.

```
def find_pythagorean_triplet(arr):
    # Square all elements
    squared = [x * x for x in arr]
    squared.sort()  # Sorting the squared values (O(n log n))

    n = len(squared)

    # Fix the largest element as k^2 and find i^2 + j^2 = k^2
    for k in range(n - 1, 1, -1):  # Start from the largest
        left, right = 0, k - 1  # Two-pointer approach

        while left < right:
            if squared[left] + squared[right] == squared[k]:
                return int(squared[left]**0.5), int(squared[right]**0.5), int(squared[k]**0.5)

            if squared[left] + squared[right] < squared[k]:
                left += 1  # Increase sum
            else:
                right -= 1  # Decrease sum

    return None  # No triplet found

# Example usage
arr = [3, 1, 4, 6, 5]
result = find_pythagorean_triplet(arr)
print("Pythagorean triplet:", result)
```

```
Pythagorean triplet: (3, 4, 5)
```

Problem 8 : An element is a majority if it appears more than n/2 times. Give an algorithm takes an array of n element as argument and identifies a majority (if it exists).

```
def find_majority_element(arr):
    # Phase 1: Find Candidate
    candidate, count = None, 0

    for num in arr:
        if count == 0:
            candidate = num
        count += 1 if num == candidate else -1

    # Phase 2: Verify Candidate
    count = sum(1 for num in arr if num == candidate)
    if count > len(arr) // 2:
        return candidate

    return None  # No majority element

# Example usage
arr = [3, 3, 4, 2, 4, 4, 2, 4, 4]
result = find_majority_element(arr)
print("Majority element:", result)
```

```
Majority element: 4
```

Problem 9 : Given n × n matrix, and in each row all 1's are followed by 0's. Find the row with the maximum number of 0's.

```
def first_zero_index(row):
    """Binary search to find the first occurrence of 0 in a row."""
    left, right = 0, len(row) - 1
    while left <= right:
        mid = (left + right) // 2
        if row[mid] == 0:
            right = mid - 1  # Search in the left half
        else:
            left = mid + 1  # Search in the right half
    return left  # Position of first 0

def row_with_max_zeros(matrix):
    n = len(matrix)
    max_zeros = 0
```

```
    row_index = -1

    for i in range(n):
        first_zero = first_zero_index(matrix[i])
        num_zeros = n - first_zero  # Number of 0's in this row

        if num_zeros > max_zeros:
            max_zeros = num_zeros
            row_index = i

    return row_index

# Example usage
matrix = [
    [1, 1, 0, 0],  # 2 zeros
    [1, 0, 0, 0],  # 3 zeros
    [1, 1, 1, 0],  # 1 zero
    [0, 0, 0, 0]   # 4 zeros
]

result = row_with_max_zeros(matrix)
print("Row with maximum zeros:", result)
```

Problem 10 : Sort an array of 0's, 1's and 2's [or R's, G's and B's]: Given an array A[] consisting of 0's, 1's and 2's, give an algorithm for sorting A[].The algorithm should put all 0's first, then all 1's and finally all 2's at the end. Example Input = {0,1,1,0,1,2,1,2,0,0,0,1}, Output = {0,0,0,0,0,1,1,1,1,1,2,2}

```
def sort_colors(arr):
    low, mid, high = 0, 0, len(arr) - 1

    while mid <= high:
        if arr[mid] == 0:
            arr[low], arr[mid] = arr[mid], arr[low]  # Swap
            low += 1
            mid += 1
        elif arr[mid] == 1:
            mid += 1  # Just move mid forward
        else:  # arr[mid] == 2
            arr[mid], arr[high] = arr[high], arr[mid]  # Swap
            high -= 1

# Example usage
arr = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1]
sort_colors(arr)
print("Sorted array:", arr)
```

```
Sorted array: [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2]
```