Sorting

Practice Questions

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

In [5]:
```
def most_freq_ele(arr):
    most_count = 0
    most_frequent = None
    for num in arr:
        count = 0
        for other_num in arr:
            if num == other_num:
                count += 1
        if count >= most_count:
            most_count = count
            most_frequent = num
    return most_frequent

arr = [1,2,1,1,3,4,5,2,3,1,4,1,9]
print(most_freq_ele(arr))


## Time complexity: O(n^2): Not efficient
```

```
1
```

In [4]:
```
## Better approach
##Time complexity: O(n)

def find_most_frequent_ele(arr):
    frequency_map = {}
    # Counting the frequency of each element
    for ele in arr:
        if ele in frequency_map:
            frequency_map[ele] += 1
        else:
            frequency_map[ele] = 1

    # Finding out the most frequent element
    most_count = 0
    most_frequent_ele = None
    for key, value in frequency_map.items():
        if value > most_count:
            most_count = value
            most_frequent_ele = key

    return most_frequent_ele

#example usage:

arr = [1,2,1,1,3,4,5,2,3,1,4,1,9]
print(find_most_frequent_ele(arr))
```

```
1
```

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.

In [12]:
```
def find_missing_ele(arr):
    n = len(arr) + 1
    expected_sum = n*(n+1)//2
    actual_sum = sum(arr)
    missing_ele = expected_sum - actual_sum
    return missing_ele

arr = [1,2,4,6,3,7,8]
print("The missing element is:",find_missing_ele(arr))
```

```
The missing element is: 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.

In [20]:
```
'''
the XOR operator works by comparing the bits of two numbers and returning 1
if the bits are different and 0 if the bits are the same
'''
def find_odd_occurrence(arr):
    result = 0
    for num in arr:
        result ^= num
    return result

# Example usage:
arr = [1, 2, 3, 2, 3, 1, 3]
result = find_odd_occurrence(arr)
print(result)
```

```
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.

In [28]:
```
def find_ele(arr, k):
    seen = set()
    for ele in arr:
        complement = k - ele
        if complement in seen:
            return (ele, complement)
        seen.add(ele)
    return None

#Example usage
arr = [1, 3, 5, 6, 7, 8, 3]
print(find_ele(arr, 10))
```

```
(7, 3)
```

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.

In [3]:
```
def closest_numbers_to_zero(arr):
    #sort the array
    arr.sort()

    left = 0
    right = len(arr) - 1
    closest_sum = float('inf')
    closest_pair = (None, None)

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

        #Update the closest pair
        if abs(current_sum) < abs(closest_sum):
            closest_sum = current_sum
            closest_pair = (arr[left], arr[right])

        #Move pointers
        if current_sum < 0:
            left += 1
        elif current_sum > 0:
            right -= 1
        else:
            break  # If the sum is zero, no need to continue

    return closest_pair

# Example usage
arr = [1, 60, -10, 70, -80, 85]
result = closest_numbers_to_zero(arr)
print("Two numbers with sum closest to zero:", result)
```

```
Two numbers with sum 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.

In [6]:
```
def find_three_elements(arr, target_sum):
    arr.sort()

    for i in range(len(arr) - 2):

        left = i + 1
        right = len(arr) - 1

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

            if current_sum == target_sum:
                return arr[i], arr[left], arr[right]
            elif current_sum < target_sum:
                left += 1
            else:
                right -= 1

    return None

# Example usage

arr = [1,2,5,9,6,7]
target_sum = 10
result = find_three_elements(arr, target_sum)

if result:
    print(f"Three elements with sum {target_sum}: {result}")
else:
    print("No such three elements found.")
```

```
Three elements with sum 10: (1, 2, 7)
```

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.

In [7]:
```
def find_triplets(arr):
    n = len(arr)
    for i in range(n):
        for j in range(i+1,n):
            for k in range(j+1,n):
                # Check if i*i + j*j equals k*k
                if arr[i] * arr[i] + arr[j] * arr[j] == arr[k] * arr[k] or
                   arr[j] * arr[j] + arr[k] * arr[k] == arr[i] * arr[i] or
                   arr[k] * arr[k] + arr[i] * arr[i] == arr[j] * arr[j]:
                    return arr[i], arr[j], arr[k]

#example usage

arr = [1,2,3,4,5]
result = find_triplets(arr)
if result:
    print("Triplet found:",result)
else:
    print("No triplet found!")
```

```
Triplet found: (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).

In [14]:
```
def find_majority(arr):
    n = len(arr)
    unique_ele = set()

    for ele in arr:
        unique_ele.add(ele)

    for ele in unique_ele:
        count = arr.count(ele)
        if count > n/2:
            return ele

arr = [1,2,3,4,5,6,2,3,2,2,2,2,2,2,2,2]
print(find_majority(arr))
```

```
2
```

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.

In [15]:
```
def find_row_with_max_zeros(matrix):
    max_zeros_row = 0
    max_zeros_count = 0

    for i in range(len(matrix)):
        zeros_count = count_zeros(matrix[i])
        if zeros_count > max_zeros_count:
            max_zeros_count = zeros_count
            max_zeros_row = i

    return max_zeros_row

def count_zeros(row):
    left, right = 0, len(row) - 1

    while left <= right:
        mid = (left + right) // 2

        if row[mid] == 0:
            # If the mid element is 0, check for more zeros on the left sid
            right = mid - 1
        else:
            # If the mid element is 1, move to the right side
            left = mid + 1

    return len(row) - left

# Example usage
matrix = [
    [1, 1, 1, 0, 0],
    [1, 1, 0, 0, 0],
    [1, 0, 0, 0, 0],
    [1, 1, 1, 1, 0],
    [1, 1, 1, 0, 0]
]

result = find_row_with_max_zeros(matrix)
print("Row with maximum number of zeros:", result)
```

```
Row with maximum number of zeros: 2
```

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}

In [17]:
```
def sort_num(arr):
    low, mid, high = 0, 0, len(arr) - 1

    while mid <= high:
        if arr[mid] == 0:
            arr[low], arr[mid] = arr[mid], arr[low]
            low += 1
            mid += 1
        elif arr[mid] == 1:
            mid += 1
        else:
            arr[mid], arr[high] = arr[high], arr[mid]
            high -= 1

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

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