**1.Given an array of strings words, return the first palindromic string in the array. If there is no such string, return an empty string "". A string is palindromic if it reads the same forward and backward.**

**Example :**

**Input: words = ["notapalindrome","racecar"]**

**Output: "racecar"**

**program:**

def first\_palindrome(words):

for word in words:

if word == word[::-1]:

return word

return ""

print(first\_palindrome(["notapalindrome","racecar"]))

Output: "racecar"

**2.You are given two integer arrays nums1 and nums2 of sizes n and m, respectively.**

**Calculate the following values: answer1 : the number of indices i such that nums1[i]**

**exists in nums2. answer2 : the number of indices i such that nums2[i] exists in nums1**

**Return [answer1,answer2].**

**Example :**

**Input: nums1 = [2,3,2], nums2 = [1,2]**

**Output: [2,1]**

**program:**

def find\_indices(nums1, nums2):

set1 = set(nums1)

set2 = set(nums2)

answer1 = sum(1 for num in nums1 if num in set2)

answer2 = sum(1 for num in nums2 if num in set1)

return [answer1, answer2]

print(find\_indices([2,3,2], [1,2]))

Output: [2,1]

**3.You are given a 0-indexed integer array nums. The distinct count of a subarray of nums is defined as: Let nums[i..j] be a subarray of nums consisting of all the indices from i to j such that 0 <= i <= j < nums.length. Then the number of distinct values in nums[i..j] is called the distinct count of nums[i..j]. Return the sum of the squares of distinct counts of all subarrays of nums. A subarray is a contiguous non-empty sequence of elements within an array.**

**Example 1:**

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

**Output: 15**

**program:**

def sum\_of\_squares\_of\_distinct\_counts(nums):

res = 0

for i in range(len(nums)):

freq = set()

for j in range(i, len(nums)):

freq.add(nums[j])

res += len(freq) \*\* 2

return res

print(sum\_of\_squares\_of\_distinct\_counts([1,2,1]))

Output: 15

**4.Given a 0-indexed integer array nums of length n and an integer k, return the number of pairs (i, j) where 0 <= i < j < n, such that nums[i] == nums[j] and (i \* j) is divisible by k.**

**Example 1:**

**Input: nums = [3,1,2,2,2,1,3], k = 2**

**Output: 4**

**program:**

def count\_pairs\_with\_divisible\_product(nums, k):

freq = {}

for num in nums:

freq[num] = freq.get(num, 0) + 1

return sum(v \* (v - 1) // 2 for num, v in freq.items() if v > 1 and (k == 0 or num == 0 or k % num == 0))

print(count\_pairs\_with\_divisible\_product([3,1,2,2,2,1,3], 2))

Output: 4

**5.Write a program FOR THE BELOW TEST CASES with least time complexity**

**Test Cases:**

**1) Input: {1, 2, 3, 4, 5} Expected Output: 5**

**2) Input: {7, 7, 7, 7, 7} Expected Output: 7**

**3) Input: {-10, 2, 3, -4, 5} Expected Output: 10**

**program:**

1)def max\_absolute\_value(nums):

return max(abs(num) for num in nums)

print(max\_absolute\_value([1, 2, 3, 4, 5]))

Output: 5

2)def max\_absolute\_value(nums):

return max(abs(num) for num in nums)

print(max\_absolute\_value([7, 7, 7, 7, 7]))

Output: 7

3)def max\_absolute\_value(nums):

return max(abs(num) for num in nums)

print(max\_absolute\_value([-10, 2, 3, -4, 5]))

Output: 10

**6. You have an algorithm that process a list of numbers. It firsts sorts the list using an efficient sorting algorithm and then finds the maximum element in sorted list. Write the code for the same.**

**Test Cases**

**1. Empty List**

**Input: []**

**Expected Output: None or an appropriate message indicating that the list is empty.**

**2. Single Element List**

**Input: [5]**

**Expected Output: 5**

**3. All Elements are the Same**

**Input: [3, 3, 3, 3, 3]**

**Expected Output: 3**

**program:**

**1)Empty List:**

def find\_max\_element(nums):

if not nums:

return None

nums.sort()

return nums[-1]

print(find\_max\_element([]))

Output: None

**2)Single Element List:**

def find\_max\_element(nums):

if not nums:

return None

nums.sort()

return nums[-1]

print(find\_max\_element([5]))

Output: 5

**3)All Elements are the Same:**

def find\_max\_element(nums):

if not nums:

return None

return nums[-1]

print(find\_max\_element([3, 3, 3, 3, 3]))

Output: 3

**7.Write a program that takes an input list of n numbers and creates a new list containing only the unique elements from the original list. What is the space complexity of the algorithm?**

**Test Cases**

**1)Some Duplicate Elements**

**Input: [3, 7, 3, 5, 2, 5, 9, 2]**

**Expected Output: [3, 7, 5, 2, 9] (Order may vary based on the algorithm used)**

**2)Negative and Positive Numbers**

**Input: [-1, 2, -1, 3, 2, -2]**

**Expected Output: [-1, 2, 3, -2] (Order may vary)**

**3)List with Large Numbers**

**Input: [1000000, 999999, 1000000]**

**Expected Output: [1000000, 999999]**

**program:**

def unique\_elements(nums):

return list(set(nums))

# Test Case 1: Some Duplicate Elements

print(unique\_elements([3, 7, 3, 5, 2, 5, 9, 2])) # Output: [3, 7, 5, 2, 9]

# Test Case 2: Negative and Positive Numbers

print(unique\_elements([-1, 2, -1, 3, 2, -2])) # Output: [-1, 2, 3, -2]

# Test Case 3: List with Large Numbers

print(unique\_elements([1000000, 999999, 1000000])) # Output: [1000000, 999999]

**8.Sort an array of integers using the bubble sort technique. Analyze its time complexity using Big-O notation. Write the code**

**program:**

def bubble\_sort(arr):

n = len(arr)

for i in range(n-1):

swapped = False

for j in range(0, n-i-1):

if arr[j] > arr[j+1]:

arr[j], arr[j+1] = arr[j+1], arr[j]

swapped = True

if not swapped:

break

return arr

arr = [64, 34, 25, 12, 22, 11, 90]

print("Original array:", arr)

print("Sorted array:", bubble\_sort(arr))

output:Original array: [64, 34, 25, 12, 22, 11, 90]

Sorted array: [11, 12, 22, 25, 34, 64, 90]

**9.Checks if a given number x exists in a sorted array arr using binary search. Analyze its time complexity using Big-O notation.**

**Test Case:**

**Example X={ 3,4,6,-9,10,8,9,30} KEY=10**

**Output: Element 10 is found at position 5**

**program:**

def binary\_search(arr, x):

low, high = 0, len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] == x: return mid

elif arr[mid] < x: low = mid + 1

else: high = mid - 1

return -1

arr = [3, 4, 6, -9, 10, 8, 9, 30]

arr.sort()

x = 10

result = binary\_search(arr, x)

if result != -1:

print(f"Element {x} is found at position {result+1}")

else:

print(f"Element {x} is not found in the array")

**10. Given an array of integers nums, sort the array in ascending order and return it. You must solve the problem without using any built-in functions in O(nlog(n)) time**

**complexity and with the smallest space complexity possible.**

**program:**

def quick\_sort(nums):

if len(nums) <= 1:

return nums

pivot = nums[len(nums) // 2]

left = [x for x in nums if x < pivot]

middle = [x for x in nums if x == pivot]

right = [x for x in nums if x > pivot]

return quick\_sort(left) + middle + quick\_sort(right)

nums = [5, 2, 8, 3, 1, 6, 4]

print("Original array:", nums)

print("Sorted array:", quick\_sort(nums))