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.

a=["abc","car","ada","racecar","cool"]

output=""

for i in a:

if i==i[::-1]:

output=i

break

print(output)

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].

a=[4,3,2,3,1]

b=[2,2,5,2,3,6]

c=d=0

for i in a:

if i in b:

c+=1

for i in b:

if i in a:

d+=1

print([c,d])

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.

nums = [1, 2, 1]

n = len(nums)

total\_sum = 0

for i in range(n):

distinct\_set = set()

for j in range(i, n):

distinct\_set.add(nums[j])

print(distinct\_set)

distinct\_count = len(distinct\_set)

total\_sum += distinct\_count \*\* 2

print(total\_sum)

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.

nums=[3,1,2,2,2,1,3]

k=2

n=len(nums)

count=0

for i in range(n-1):

for j in range(i+1,n):

if nums[i]==nums[j] and (i\*j)%k==0:

count+=1

print(count)

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

Test Cases: -

print(max(1,2,3,4,5))

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.

def process\_list(nums):

if not nums:

print("The list is empty.")

return None

nums.sort()

return nums[-1]

test\_cases = [[], [5], [3,3,3,3,3], [4,7,2,9,1 ]

for case in test\_cases:

result = process\_list(case)

print(f"Input: {case} -> Output: {result}")

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 = [[3, 7, 3, 5, 2, 5, 9, 2], [-1, 2, -1, 3, 2, -2], [1000000, 999999, 1000000]]

for case in test\_cases:

unique\_list = list(set(case))

print(f"Input: {case} -> Unique Elements: {unique\_list}")

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

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

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

test\_cases = [[64, 34, 25, 12, 22, 11, 90],[1, 2, 3, 4, 5],[5, 4, 3, 2, 1],[]]

for case in test\_cases:

bubble\_sort(case)

print(f"Sorted Array: {case}")

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

def binary\_search(arr, key):

low = 0

high = len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] == key:

return mid

elif key < arr[mid]:

high = mid - 1

else:

low = mid + 1

return -1

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

key1 = 10

result1 = binary\_search(arr, key1)

if result1 != -1:

print(f"Element {key1} is found at position {result1}")

else:

print(f"Element {key1} is not found.")

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.

def merge(left, right):

sorted\_list = []

i = j = 0

while i < len(left) and j < len(right):

if left[i] < right[j]:

sorted\_list.append(left[i])

i += 1

else:

sorted\_list.append(right[j])

j += 1

sorted\_list.extend(left[i:])

sorted\_list.extend(right[j:])

return sorted\_list

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left\_half = merge\_sort(arr[:mid])

right\_half = merge\_sort(arr[mid:])

return merge(left\_half, right\_half)

nums1 = [5, 2, 9, 1, 5, 6]

nums2 = [0, -1, -5, 3, 2, 1]

nums3 = [100, 100, 100, 100]

nums4 = []

print("Sorted Array:", merge\_sort(nums1))

print("Sorted Array:", merge\_sort(nums2))

print("Sorted Array:", merge\_sort(nums3))

print("Sorted Array:", merge\_sort(nums4))