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.

Code:

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

for word in words:

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

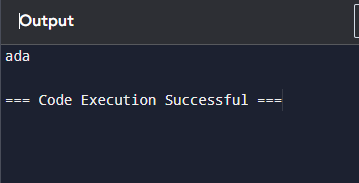
print(word)

break

else:

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].

Code:

nums1 = [1, 2, 3, 4]

nums2 = [3, 4, 5, 6]

answer1 = 0

answer2 = 0

for i in range(len(nums1)):

if nums1[i] in nums2:

answer1 += 1

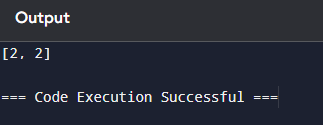
for i in range(len(nums2)):

if nums2[i] in nums1:

answer2 += 1

print([answer1, answer2])

output:



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.

Code:

nums = [1, 2]

result = 0

for i in range(len(nums)):

distinct\_elements = set()

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

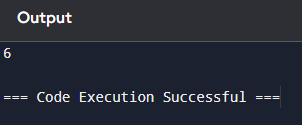
distinct\_elements.add(nums[j])

distinct\_count =len(distinct\_elements)

result += distinct\_count \*\* 2

print(result)

output:



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.

Code:

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

k = 3

count = 0

n = len(nums)

for i in range(n):

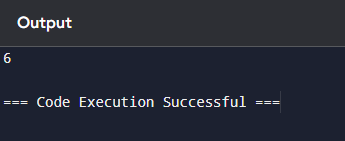
for j in range(i + 1, n):

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

count += 1

print(count)

output:



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

Test Cases: -

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

Input: {7, 7, 7, 7, 7} Expected Output: 7

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

Code:

nums1 = [1, 2, 3, 4, 5]

max\_value = nums1[0]

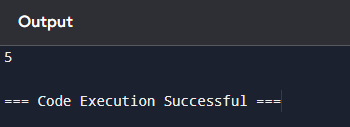
for i in range(1, len(nums1)):

if nums1[i] > max\_value:

max\_value = nums1[i]

print(max\_value)

output:



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.

Code:

nums = [3, 1, 4, 1, 5, 9, 2, 6]

n = len(nums)

for i in range(n):

min\_index = i

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

if nums[j] < nums[min\_index]:

min\_index = j

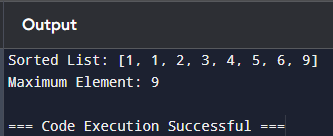
nums[i],nums[min\_index]=nums[min\_index],nums[i]

max\_value = nums[-1]

print("Sorted List:", nums)

print("Maximum Element:", max\_value)

output:



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?

Code:

original\_list = [1, 2, 3, 1, 2, 4, 5, 3]

unique\_list = []

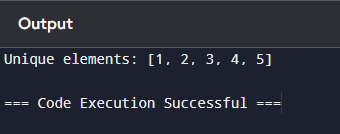
for num in original\_list:

if num not in unique\_list:

unique\_list.append(num)

print("Unique elements:", unique\_list)

output:



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

Code:

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

n = len(nums)

for i in range(n):

swapped = False

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

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

nums[j], nums[j + 1] = nums[j + 1], nums[j]

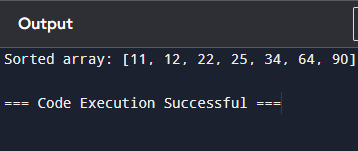
swapped = True

if not swapped:

break

print("Sorted array:", nums)

output:



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

Code:

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

x = 7

left = 0

right = len(arr) - 1

found = False

while left <= right:

mid = left + (right - left) // 2

if arr[mid] == x:

found = True

break

elif arr[mid] > x:

right = mid - 1

else:

left = mid + 1

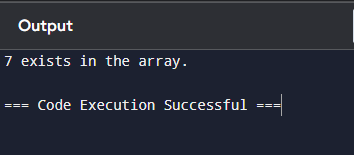
if found:

print(x, "exists in the array.")

else:

print(x, "does not exist in the array.")

output:



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.

Code:

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

n = len(nums)

for i in range(n):

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

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

nums[j], nums[j + 1] = nums[j + 1], nums[j]

print("Sorted array:", nums)

output:

