**1.1** 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.

**AIM:**

**ALGORITHM:**

**Start**

 Iterate through each string s in the array words:

* Check if s is a palindrome:
  + Compare characters from the beginning and end moving inward.
  + If all characters match, then s is a palindrome.
* If s is palindrome:
  + Return s.

 If no palindrome is found after checking all strings:

* Return "".

**End**

**PROGRAM:**

def is\_palindrome(s: str) -> bool:

return s == s[::-1]

def first\_palindrome(words):

for w in words:

if is\_palindrome(w):

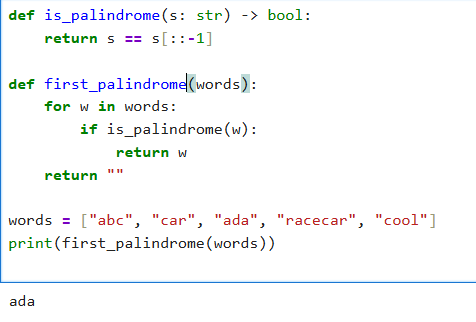
return w

return ""

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

print(first\_palindrome(words))

**INPUT AND OUTPUT:**

****

**RESULT:**

Thus the program to find first palindrome string in a array is successfully executed and output is verified.

**1.2** 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].

**AIM:**

**ALGORITHM:**

**Start**

 Convert nums2 into a **set** (for O(1) lookup).

 Initialize answer1 = 0.

* For each element in nums1:
  + If element exists in set(nums2), increment answer1.

 Convert nums1 into a **set** (for O(1) lookup).

 Initialize answer2 = 0.

* For each element in nums2:
  + If element exists in set(nums1), increment answer2.

Return [answer1, answer2].

**End**

**PROGRAM:**

def count\_matches(nums1, nums2):

set1, set2 = set(nums1), set(nums2)

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

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

return [answer1, answer2]

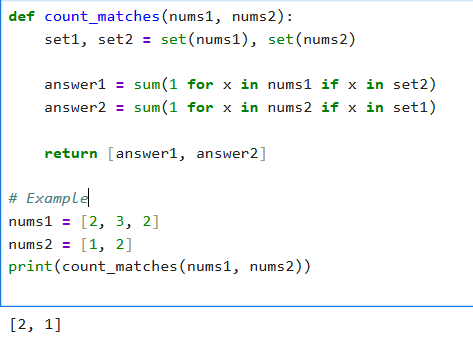
# Example

nums1 = [2, 3, 2]

nums2 = [1, 2]

print(count\_matches(nums1, nums2))

**INPUT AND OUTPUT:**

****

**RESULT:**

Thus the program to Find Indices with Elements Present in Both Arrays excuted successfully and output is verified.

**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.

**AIM:**

Sum of Squares of Distinct Counts of All Subarrays

**ALGORITHM:**

 Initialize answer = 0.

 For each starting index i (0 → n-1):

* Create a frequency map (hashmap).
* For each ending index j (i → n-1):
  + Insert nums[j] into the map (track counts).
  + The number of distinct elements = size of map.
  + Add (distinct\_count)^2 to answer.

 Return answer.

**PROGRAM:**

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

n = len(nums)

total = 0

for i in range(n):

seen = set()

for j in range(i, n):

seen.add(nums[j])

distinct\_count = len(seen)

total += distinct\_count \*\* 2

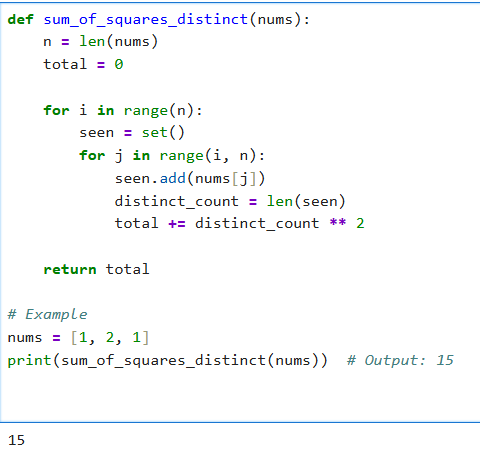
return total

# Example

nums = [1, 2, 1]

print(sum\_of\_squares\_distinct(nums))

**INPUT AND OUTPUT:**



**RESULT:**

Thus the program for Sum of Squares of Distinct Counts of All Subarrays executed successfully and output is verified.

**1.4** 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.

**AIM:**

Count Equal Pairs with Index Product Divisible by k

**ALGORITHM:**

* Initialize count = 0.
* Loop through all pairs (i, j) with i < j.
* If nums[i] == nums[j] and (i \* j) % k == 0, increment count.
* Return count.

**PROGRAM:**

def count\_pairs(nums, k):

n = len(nums)

count = 0

for i in range(n):

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

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

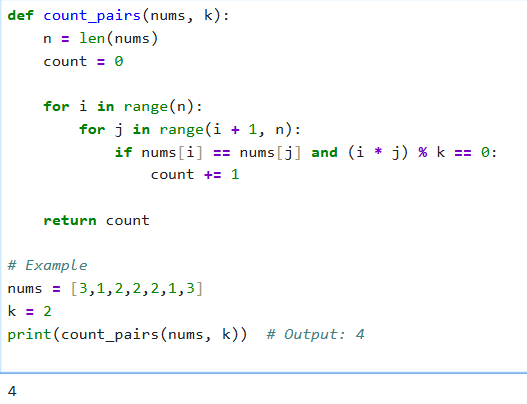
count += 1

return count

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

k = 2

print(count\_pairs(nums, k))

**INPUT AND OUTPUT:**

**RESULT:**

Thus the program to find Count Equal Pairs with Index Product Divisible by k

Executed successfully and output is verified.

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

**AIM:**

Program to find Least time complexity

**ALGORITHM:**

 Initialize maxVal = nums[0].

 Traverse the array once.

* If nums[i] > maxVal, update maxVal = nums[i].

 After loop ends, return maxVal.

**PROGRAM:**

def find\_max(arr):

return max(arr)

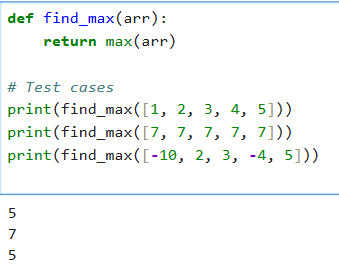
# Test cases

print(find\_max([1, 2, 3, 4, 5])) # 5

print(find\_max([7, 7, 7, 7, 7])) # 7

print(find\_max([-10, 2, 3, -4, 5]))

**INPUT AND OUTPUT:**

****

**RESULT:**

Thus the program to find least time complexity executed successfully and output is verified.

**1.6** 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.

**AIM:**

Find Maximum Element in a List

**ALGORITHM:**

 **Input:** A list of integers nums.

 Initialize maxVal = nums[0].

 Traverse each number x in nums:

* If x > maxVal, update maxVal = x.

 After traversal, return maxVal.

**Output:** The maximum element of the list.

**PROGRAM:**

def process\_list(nums):

if not nums: # Empty list

return None

nums.sort() # Efficient sort (Timsort in Python, O(n log n))

return nums[-1] # Last element is maximum

# Test Cases

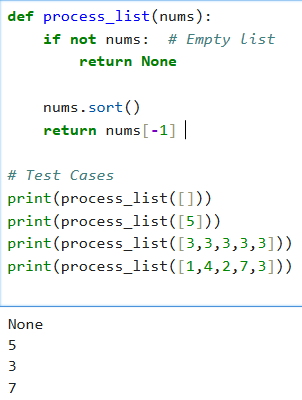
print(process\_list([]))

print(process\_list([5]))

print(process\_list([3,3,3,3,3]))

print(process\_list([1,4,2,7,3]))

**INPUT AND OUTPUT:**

****

**RESULT:**

Thus the program to Find Maximum Element in a List ran succesully and op is verified.

**1.7** 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?

**AIM:** Extract Unique Elements from a List

**ALGORITHM:**

**** Input: A list of n numbers.

 Create an empty set seen.

 Traverse each element x in the list:

* If x is not in seen, add it to seen.

 Convert seen into a list unique\_list.

 Output: Return unique\_list.

**PROGRAM:**

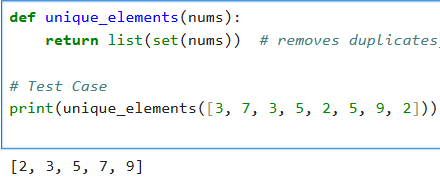
def unique\_elements(nums):

return list(set(nums)) # removes duplicates, order not preserved

# Test Case

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

**INPUT AND OUTPUT:**

****

**RESULT:**

thus code for Extract Unique Elements from a List executed successfully and op is verified

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

**AIM:**

Bubble Sort Implementation and Time Complexity Analysis

**ALGORITHM:**

 **Input:** An array of integers arr of size n.

 Repeat the following n-1 times:

* Traverse the array from index 0 to n-i-2.
* Compare adjacent elements arr[j] and arr[j+1].
* If arr[j] > arr[j+1], swap them.

 After each iteration, the largest element among the unsorted part "bubbles up" to its correct position.

 **Output:** Sorted array.

**PROGRAM:**

def bubble\_sort(arr):

n = len(arr)

for i in range(n-1):

swapped = False

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

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

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

swapped = True

# If no two elements were swapped in the inner loop, break

if not swapped:

break

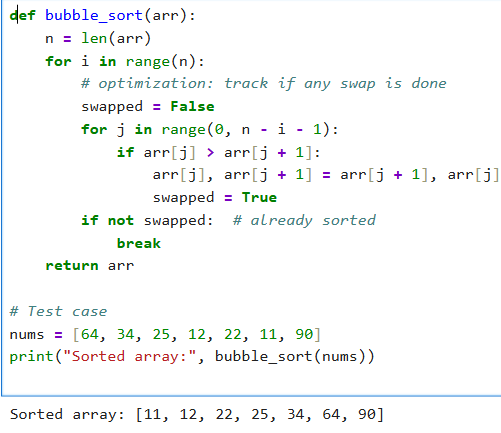
return arr

# Test

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

print("Sorted Array:", bubble\_sort(nums))

**INPUT AND OUTPUT:**

****

**RESULT:**

Thus the program Bubble Sort Implementation and Time Complexity Analysis executed successfully and op is verified

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

**AIM:**

Binary Search in a Sorted Array

**ALGORITHM:**

**Input:** A sorted array arr and a target element key.

1. Initialize two pointers:
   * low = 0
   * high = len(arr) - 1
2. While low <= high:
   * Compute mid = (low + high) // 2.
   * If arr[mid] == key, return mid (position).
   * If arr[mid] < key, search right half (low = mid + 1).
   * Else search left half (high = mid - 1).
3. If not found, return -1.

**PROGRAM:**

def binary\_search(arr, key):

arr.sort() # ensure sorted

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

while low <= high:

mid = (low + high) // 2

if arr[mid] == key:

return mid + 1 # position (1-based index)

elif arr[mid] < key:

low = mid + 1

else:

high = mid - 1

return -1

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

KEY = 10

pos = binary\_search(X, KEY)

if pos != -1:

print(f"Element {KEY} is found at position {pos}")

else:

print(f"Element {KEY} not found")

**INPUT AND OUTPUT:**

**RESULT:**