Q 01.first palindromic string in the array??

CODE:

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

result = ""

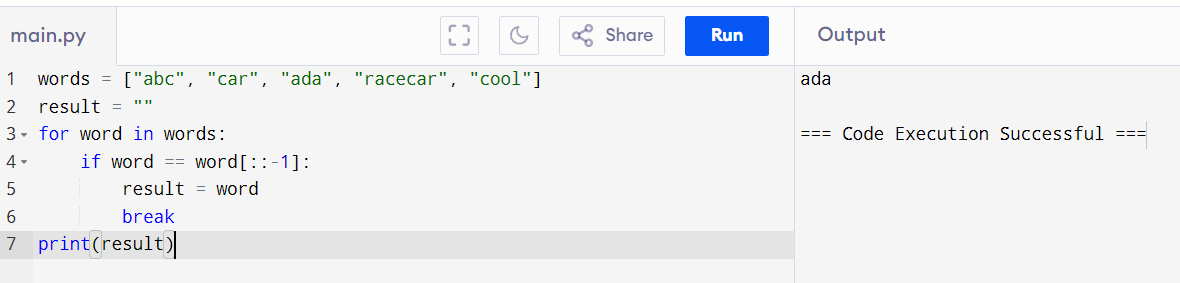
for word in words:

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

result = word

break

print(result)

OUTPUT: car

Q 02. Two integer arrays nums1 and nums2 of sizes n and m ??

CODE:

nums1 = [1, 2, 3, 4]

nums2 = [3, 4, 5, 6]

answer1 = 0

answer2 = 0

for i in nums1:

if i in nums2:

answer1 += 1

for i in nums2:

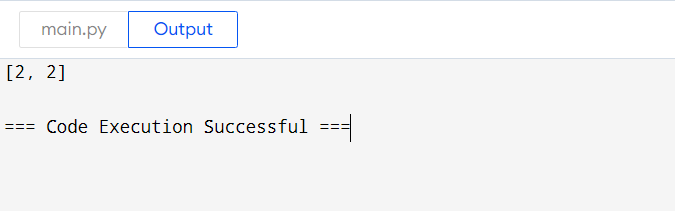
if i in nums1:

answer2 += 1

result = [answer1, answer2]

print(result)

OUTPUT:[2,2]



Q 03.sum of the squares of distinct counts of all subarrays of nums??

CODE:

nums = [1, 2, 1]

n = len(nums)

total\_sum = 0

for i in range(n):

distinct\_set = set()

for j in range(i, n):

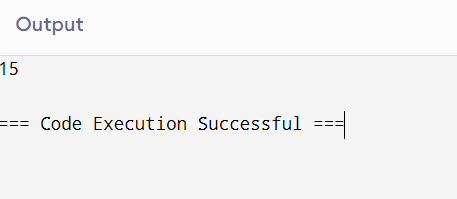
distinct\_set.add(nums[j])

distinct\_count = len(distinct\_set)

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

print(total\_sum)

OUTPUT:15



Q 04.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 = [3,1,2,2,2,1,3]

k = 2

count = 0

for i in range(len(nums)):

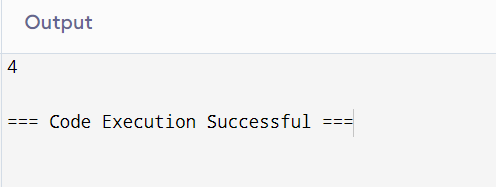
for j in range(i + 1, len(nums)):

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

count += 1

print(count)

OUTPUT:4



Q 05. Least time complexity??

CODE:

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

max\_element = arr[0]

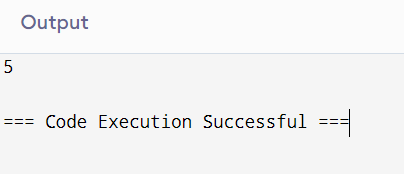
for i in range(1, len(arr)):

if arr[i] > max\_element:

max\_element = arr[i]

print(max\_element)

OUTPUT:5



Q 06. finds the maximum element in sorted list ??

CODE:

nums = [3,3,3,3,3]

if len(nums) == 0:

print("The list is empty. No maximum element.")

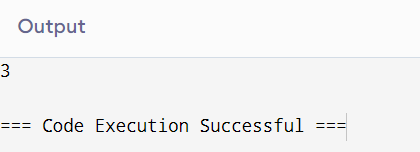
else:

sorted\_nums = sorted(nums)

max\_element = sorted\_nums[-1]

print(max\_element)

OUTPUT:3



Q 07.unique elements from the original list. What is the space complexity of the algorithm?

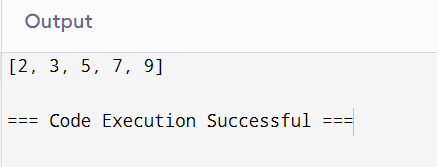
CODE:

nums = [3,7,3,5,2,5,9,2]

unique\_elements = list(set(nums))

print(unique\_elements)

OUTPUT:[2, 3, 5, 7, 9]



Q 08.Sort an array of integers using the bubble sort technique. time complexity using Big-O notation.

CODE:

nums = [3,3,3,3,3]

if len(nums) == 0:

print("The list is empty. No maximum element.")

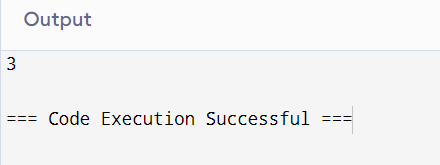
else:

sorted\_nums = sorted(nums)

max\_element = sorted\_nums[-1]

print(max\_element)

OUTPUT:3



Q 09.Checks if a given number x exists in a sorted array arr using binary search. Time complexity using Big-O notation.

CODE:

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

arr.sort()

key = 10

left, right = 0, len(arr) - 1

found = False

while left <= right:

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

if arr[mid] == key:

print(f"Element {key} is found at position {mid}")

found = True

break

elif arr[mid] < key:

left = mid + 1

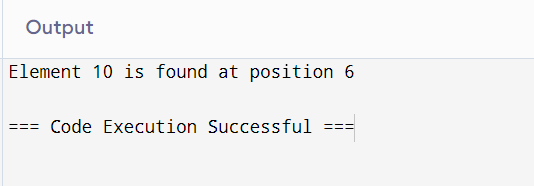
else:

right = mid - 1

if not found:

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

OUTPUT: Element 10 is found at position 6



Q 10.Sort the array in ascending order and return it.without using any built-in functions in O(nlog(n)) time

complexity and with the smallest space complexity possible.??

CODE:

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

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

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

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

merge\_sort(nums)

print(nums)

OUTPUT:[-9, 3, 4, 6, 8, 9, 10, 30]

