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 1: Input: words = ["abc","car","ada","racecar","cool"] Output: "ada" Explanation: The first string that is palindromic is "ada". Note that "racecar" is also palindromic, but it is not the first. Example 2: Input: words = ["notapalindrome","racecar"] Output: "racecar" Explanation: The first and only string that is palindromic is "racecar".

Ans:

def first\_palindrome(words):

for word in words:

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

return word

return ""

# Example usage

words1 = ["abc", "car", "ada", "racecar", "cool"]

print(first\_palindrome(words1)) # Output: "ada"

words2 = ["notapalindrome", "racecar"]

print(first\_palindrome(words2)) # Output: "racecar"

1. 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 1: Input: nums1 = [2,3,2], nums2 = [1,2] Output: [2,1] Explanation: Example 2: Input: nums1 = [4,3,2,3,1], nums2 = [2,2,5,2,3,6] Output: [3,4] Explanation: The elements at indices 1, 2, and 3 in nums1 exist in nums2 as well. So answer1 is 3. The elements at indices 0, 1, 3, and 4 in nums2 exist in nums1. So answer2 is 4.

Ans:

def count\_elements(nums1, nums2):

set\_nums2 = set(nums2)

answer1 = sum(1 for num in nums1 if num in set\_nums2)

set\_nums1 = set(nums1)

answer2 = sum(1 for num in nums2 if num in set\_nums1)

return [answer1, answer2]

# Example usage:

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

print(count\_elements([4, 3, 2, 3, 1], [2, 2, 5, 2, 3, 6])) # Output: [3, 4]

1. 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 Explanation: Six possible subarrays are: [1]: 1 distinct value [2]: 1 distinct value [1]: 1 distinct value [1,2]: 2 distinct values [2,1]: 2 distinct values[1,2,1]: 2 distinct values The sum of the squares of the distinct counts in all subarrays is equal to 12 + 12 + 12 + 22 + 22 + 22 = 15. Example 2: Input: nums = [1,1] Output: 3 Explanation: Three possible subarrays are: [1]: 1 distinct value [1]: 1 distinct value [1,1]: 1 distinct value The sum of the squares of the distinct counts in all subarrays is equal to 12 + 12 + 12 = 3.

Ans:

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

n = len(nums)

total\_sum = 0

for start in range(n):

freq = {}

distinct\_count = 0

for end in range(start, n):

num = nums[end]

if num in freq:

if freq[num] == 0:

distinct\_count += 1

freq[num] += 1

else:

freq[num] = 1

distinct\_count += 1

# Add the square of the current distinct count to the total sum

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

return total\_sum

# Example usage:

print(sum\_of\_squares\_of\_distinct\_counts([1, 2, 1])) # Output: 15

print(sum\_of\_squares\_of\_distinct\_counts([1, 1])) # Output: 3

1. 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 Explanation: There are 4 pairs that meet all the requirements: - nums[0] == nums[6], and 0 \* 6 == 0, which is divisible by 2. - nums[2] == nums[3], and 2 \* 3 == 6, which is divisible by 2. - nums[2] == nums[4], and 2 \* 4 == 8, which is divisible by 2. - nums[3] == nums[4], and 3 \* 4 == 12, which is divisible by 2. Example 2: Input: nums = [1,2,3,4], k = 1 Output: 0 Explanation: Since no value in nums is repeated, there are no pairs (i,j) that meet all the requirements.

Ans:

def countPairs(nums, k):

from collections import defaultdict

# Step 1: Create a dictionary to hold indices of each number

index\_map = defaultdict(list)

for i in range(len(nums)):

index\_map[nums[i]].append(i)

count = 0

# Step 2: Iterate over each unique number and their indices

for indices in index\_map.values():

m = len(indices)

# Step 3: Generate all pairs of indices

for i in range(m):

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

# Check if (i \* j) is divisible by k

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

count += 1

return count

# Example usage:

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

k1 = 2

print(countPairs(nums1, k1)) # Output: 4

nums2 = [1, 2, 3, 4]

k2 = 1

print(countPairs(nums2, k2)) # Output: 0

1. 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: 5

Ans:

def find\_maximum(nums):

if not nums:

return None

max\_value = nums[0]

for num in nums:

if num > max\_value:

max\_value = num

return max\_value

# Test cases

test\_cases = [

([1, 2, 3, 4, 5], 5),

([7, 7, 7, 7, 7], 7),

([-10, 2, 3, -4, 5], 5)

]

for i, (input\_data, expected) in enumerate(test\_cases):

result = find\_maximum(input\_data)

print(f"Test Case {i + 1}: Input: {input\_data}, Expected Output: {expected}, Actual Output: {result}")

1. 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 1. Input: [] 2. Expected Output: None or an appropriate message indicating that the list is empty. 2. Single Element List 1. Input: [5] 2. Expected Output: 5 3. All Elements are the Same 1. Input: [3, 3, 3, 3, 3] 2. Expected Output: 3

Ans:

def find\_max\_in\_sorted\_list(numbers):

if not numbers:

return "The list is empty."

sorted\_numbers = sorted(numbers)

return sorted\_numbers[-1]

# Test Cases

test\_cases = [

[], # Empty List

[5], # Single Element List

[3, 3, 3, 3, 3] # All Elements are the Same

]

for i, test\_case in enumerate(test\_cases, start=1):

result = find\_max\_in\_sorted\_list(test\_case)

print(f"Test Case {i} - Input: {test\_case} => Output: {result}")

1. 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 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) Negative and Positive Numbers • Input: [-1, 2, -1, 3, 2, -2] • Expected Output: [-1, 2, 3, -2] (Order may vary) List with Large Numbers • Input: [1000000, 999999, 1000000] • Expected Output: [1000000, 999999].

Ans:

def unique\_elements(input\_list):

unique\_set = set(input\_list)

unique\_list = list(unique\_set)

return unique\_list

# Test Cases

print("Test Case 1: Some Duplicate Elements")

input1 = [3, 7, 3, 5, 2, 5, 9, 2]

output1 = unique\_elements(input1)

print(f"Input: {input1} -> Output: {output1}")

print("\nTest Case 2: Negative and Positive Numbers")

input2 = [-1, 2, -1, 3, 2, -2]

output2 = unique\_elements(input2)

print(f"Input: {input2} -> Output: {output2}")

print("\nTest Case 3: List with Large Numbers")

input3 = [1000000, 999999, 1000000]

output3 = unique\_elements(input3)

print(f"Input: {input3} -> Output: {output3}")

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

Ans:

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]:

# Swap if they are in the wrong order

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

swapped = True

if not swapped:

break

return arr

# Example usage

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

sorted\_array = bubble\_sort(array)

print("Sorted array:", sorted\_array)

1. 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 Example X={ 3,4,6,-9,10,8,9,30} KEY=100 Output : Element 100 is not found.

Ans:

def binary\_search(arr, key):

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

while left <= right:

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

if arr[mid] == key:

return f"Element {key} is found at position {mid}"

elif arr[mid] < key:

left = mid + 1

else:

right = mid - 1

return f"Element {key} is not found"

# Test cases

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

key1 = 10

key2 = 100

sorted\_arr = sorted(arr)

print(binary\_search(sorted\_arr, key1))

print(binary\_search(sorted\_arr, key2))

1. 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.

Ans:

def heapify(nums, n, i):

largest = i

left = 2 \* i + 1 # left = 2\*i + 1

right = 2 \* i + 2 # right = 2\*i + 2

if left < n and nums[left] > nums[largest]:

largest = left

if right < n and nums[right] > nums[largest]:

largest = right

if largest != i:

nums[i], nums[largest] = nums[largest], nums[i] # Swap

heapify(nums, n, largest)

def heap\_sort(nums):

n = len(nums)

for i in range(n // 2 - 1, -1, -1):

heapify(nums, n, i)

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

nums[i], nums[0] = nums[0], nums[i] # Swap

heapify(nums, i, 0)

return nums

# Example usage:

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

sorted\_nums = heap\_sort(nums)

print(sorted\_nums)