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 = ["notapalindrome","racecar"]

Output: "racecar"

Explanation: The first and only string that is palindromic is "racecar".

Program:

def first\_palindromic\_string(words):

for word in words:

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

return word

return ""

words1 = ["notapalindrome", "racecar"]

print(first\_palindromic\_string(words1))

output:

ada

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:

Program:

def count\_indices(nums1, nums2):

answer1 = sum(1 for i in nums1 if i in nums2)

answer2 = sum(1 for i in nums2 if i in nums1)

return [answer1, answer2]

nums1 = [2, 3, 2]

nums2 = [1, 2]

print(count\_indices(nums1, nums2))

output:

[2, 1]

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

Program:

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

n = len(nums)

result = 0

for i in range(n):

distinct\_elements = set()

for j in range(i, n):

distinct\_elements.add(nums[j])

distinct\_count = len(distinct\_elements)

result += distinct\_count \*\* 2

return result

nums = [1, 2, 1]

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

Output:

15

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

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))

Output:

4

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

Program:

def find\_max(nums):

return max(nums)

test\_case\_1 = [1, 2, 3, 4, 5]

test\_case\_2 = [7, 7, 7, 7, 7]

test\_case\_3 = [-10, 2, 3, -4, 5]

print(find\_max(test\_case\_1))

print(find\_max(test\_case\_2))

print(find\_max(test\_case\_3))

output:

5

7

5

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

Program:

def process\_list(nums):

if not nums:

return "The list is empty."

nums.sort()

return nums[-1]

test\_case\_1 = []

test\_case\_2 = [5]

print(process\_list(test\_case\_1))

print(process\_list(test\_case\_2))

output:

The list is empty.

5

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]

Program:

def unique\_elements(nums):

unique\_set = set()

unique\_list = []

for num in nums:

if num not in unique\_set:

unique\_set.add(num)

unique\_list.append(num)

return unique\_list

test\_case\_1 = [3, 7, 3, 5, 2, 5, 9, 2]

test\_case\_2 = [-1, 2, -1, 3, 2, -2]

test\_case\_3 = [1000000, 999999, 1000000]

print(unique\_elements(test\_case\_1))

print(unique\_elements(test\_case\_2))

print(unique\_elements(test\_case\_3))

output:

[3, 7, 5, 2, 9]

[-1, 2, 3, -2]

[1000000, 999999]

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

Program:

def bubble\_sort(nums):

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

return nums

test\_case = [64, 34, 25, 12, 22, 11, 90]

sorted\_list = bubble\_sort(test\_case)

print("Sorted list:", sorted\_list)

output:

Sorted list: [11, 12, 22, 25, 34, 64, 90]

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 6

Program:

def binary\_search(arr, key):

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

while left <= right:

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

if arr[mid] == key:

return mid

elif arr[mid] < key:

left = mid + 1

else:

right = mid - 1

return -1

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

key = 10

position = binary\_search(arr, key)

if position != -1:

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

else:

print(f"Element {key} is not found in the array")

output:

Element 10 is found at position 6

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.

Program:

def merge\_sort(nums):

if len(nums) > 1:

mid = len(nums) // 2

left\_half = nums[:mid]

right\_half = nums[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]:

nums[k] = left\_half[i]

i += 1

else:

nums[k] = right\_half[j]

j += 1

k += 1

while i < len(left\_half):

nums[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

nums[k] = right\_half[j]

j += 1

k += 1

return nums

nums = [38, 27, 43, 3, 9, 82, 10]

sorted\_nums = merge\_sort(nums)

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

output:

Sorted array: [3, 9, 10, 27, 38, 43, 82]