1. Given an array of string words, return the first palindromic string in the array. If there is no such string return an empty string "".
A string is palindrome if it reads the same forward and backward.

Input:- words = ["abc", "car", "dance", "ada", "mom"]

Output:- "ada"

Algorithm:

1) Initialize a loop to iterate through each in the given array words.

2) check each word to see if it is a palindrome by comparing condition it with it's reverse

3) Return the first word that satisfies the palindromic condition

4) if no palindromic is found, return an empty string "".

Pseudocode:-

```
function
find first Palindrome (words):
    for each word in words:
        if word is equal to reverse (word):
            return word
    return ""
```

Flowchart:-

start

Initialize a loop to iterate through each word in the array words

check if the word is palindrome

Python code:-

```
def first_palindromic_string(words):
    for word in words:
        if word == words[::-1]:
            return word
    return ""
words = ["abc", "car", "racecar", "ada", "mom"]
print(first_palindromic_string(words))
```

Result:-

output = "racecar"

2) You are given 2 integer arrays nums1 and nums2 of size n and m, respectively. calculate the following values: answer1: the no of indices i such that nums1[i] exists in nums2. answer2: the no. of indices i such that nums2[i] exists in nums1. Return [answer1, answer2]

Input:- nums1 = [4, 3, 2, 3, 1] nums2 = [2, 2, 5, 2, 3, 6]

Output:- [3, 4]

Algorithm:-

1) Initialize counters (initialize two variables, answer1 and answer2, both set to 0)

2) check elements of num1 in num2

3) check element of num2 in num1

4) Return the result (After both loops are completed 5) end.

Pseudocode:-

```
function count_indices (nums1, nums2):
    answer1 = 0
    answer2 = 0
    for i from 0 to length (nums1):
        if nums1[i] is in nums2:
            answer1 += 1
    for i for 0 to length (nums2):
        if nums2[i] is in nums1:
            answer2 += 1
    return [answer1, answer2]
```

Flowchart :-

start

Initialize : answer1 = 0 , answer2 = 0

Loop through nums1 : if nums1[i] exists
in nums2 & increment answer1

Loop through nums2 : if nums2[i] exists
in nums1 increment answer2

Return : Return [answer1, answer2]

End

Python Code :-

nums1 :- [4, 3, 2, 3, 1]

nums2 :- [2, 2, 5, 2, 3, 6]

answer1 :- Sum (1 for num in nums1 if num in nums2)

answer2 :- Sum (1 for num in nums2 if num in nums1)

print = ([answer1, answer2])

Result :-

output : [3, 4]

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 $0 \le i \le j <$ nums.length. Then the number of distinct values in nums[i...j] is called distinct count of nums[i...j].

Return the sum of the squares of distinct count of all subarray of nums.

Input :- nums = [1,2,1]

output :- 15

Algorithm :-

1) Initialize variables [set n as the length of the array nums & total_sum to 0]

2) outer loop (subarray starting point)

3) Inner loop (subarray ending point)

4) calculate distinct count

5) update total sum (Add the square of the distinct count).

6) Repeat for all subarray

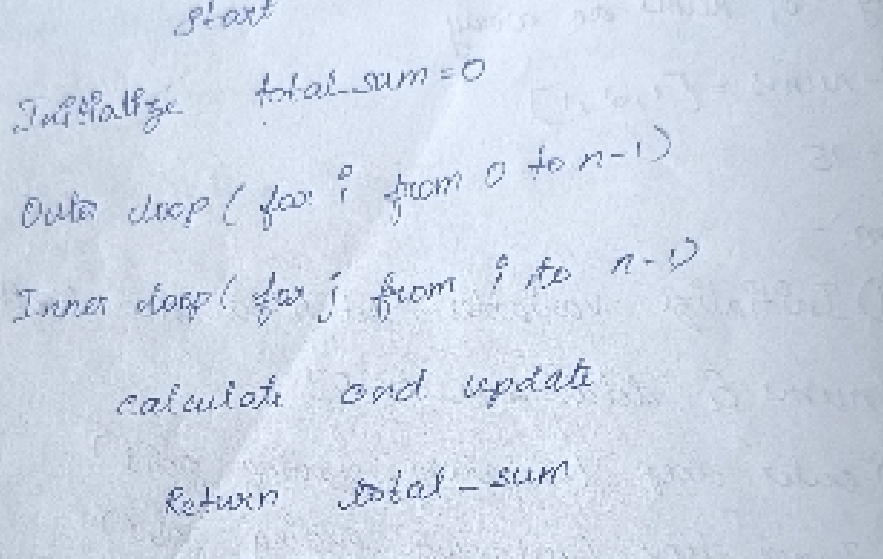
7) Return total_sum

Pseudocode :-

```
function
n = length of nums
total_sum = 0
for i from 0 to n-1 :
    distinct_elements_set()
```

for j from i to n-1:
add nums[j] to distinct elements
distinct_count = size of distinct elements
total_sum += (distinct_count ^ 2)
return total_sum.

Flowchart:-



Python Code:-

```
nums = [1, 2, 1]
n = len(nums)
total_sum = 0
for i in range(n):
    distinct = set()
    for j in range(i, n):
        distinct.add(nums[j])
    total_sum += len(distinct)
print(total_sum)
```

Result:-

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

Input :- nums = [3,1,2,2,2,1,3], k = 2

Output :- 4

Algorithm :-

1) Input
2) Initialize a variable count to 0
3) Loop through all possible pairs of (i, j) where i,
4) Return the value of count.

Pseudocode :-

```
function
count = 0
n = length (nums)
for i from 0 to n-1:
    for j from i+1 to n-1:
        if nums[i] == nums[j] AND (i*j) % k == 0:
            count = count + 1
    Return count
```

Flowchart :-

Start

Initialize count to 0

Iterate through pairs (i, j)

Check condition

Increment count if true

output count

end.

python 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
```

Result :-

output : 4.

5. write a program for the below test cases with ideal complexity.

Test case:-

1) Input:- [1,2,3,4,5] output:- 5
2) Input:- [7,7,7,7,7) Output:- 7
3) Input:- [-10,2,3,-4,5] output:- 5

Algorithm:-

1) Initialize the input list of no
2) convert the input list to a set
3) find the max element in the set
4) Return the max value found in the set as the result 5) end

Pseudocode:-

```
Function
largest_unique_element (nums):
    unique_nums = convert nums to a set
    return the max element from unique_nums.
```

Flowchart:-

Start

Input list of numbers

Convert list to set

find max element in set

max element.

Python code:

```
def largest_unique_element(nums):
    unique_nums = set(nums)
    return max(unique_nums)

nums = [1, 2, 3, 4, 5]
print(largest_unique_elements(nums))
```

Result:-

Output: 5

6) you have an algorithm that process a list of numbers. it first sorts the lists using an efficient sorting algorithm and then finds the max element in sorted list write the code for the same.

Test cases:-

Input :- [5]

output :- 5

pythoncode:-

```
nums = [5]
nums = sorted (nums)
maximum = max (nums)
print (nums)
print (maximum)
```

Result:-

output :- [5]

= 5

7) Write a program that takes an input list of n numbers and prints a new list containing only the unique elements from the original list. What is the space complexity of the algorithm?

Test case:-

Input:- [3, 7, 3, 5, 2, 5, 9, 2]

output:- [3, 7, 5, 2, 9]

Python code

```
Test_case = [
        [3, 7, 3, 5, 2, 5, 9, 2]
    ]

for nums in test_cases:

    unique_elements = list(set(nums))

    print(unique_elements)
```

Result:-

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

8) Sort an array of integers using bubble sort technique. Analyze its time complexity using O- notation

Input:- [64, 34, 25, 12, 22, 11, 90]

Output:- [11, 12, 22, 25, 34, 64, 90]

Python code:-

```
arr = [64, 34, 25, 12, 22, 11, 90]
n = len(arr)
for i in range(n):
    swapped = false
for j in range(0, n-i-1):
    if arr[j] > arr[j+1]:
        arr[j], arr[j+1] = arr[j+1], arr[j]
    swapped = True
If not swapped:
    break
print("Sorted array:", arr)
```

Result:-

output: [11, 22, 22, 25, 34, 64, 90]

a) Checks if a given number x exists in a sorted array and using binary search Analyze its time complexity using Big O notation

Input:- x = [3, 4, 6, -9, 10, 8, 9, 30], key = 10

output:- Element 10 is found at position 5.

Python code:-

```
arr = [3, 4, 6, -9, 10, 8, 9, 30]
key = 10
low = 0
high = len(arr) - 1
while low <= high:
    mid = (low + high) // 2
    if arr[mid] == key:
        print("element", key, "is found at position", mid+1)
        Break
    elif arr[mid] < key:
        low = mid + 1
    else:
        height = mid - 1
else
    print("element", key, "is not found in the array")
```

Result:-

output:- Element 10 is found at position 5.

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(n log(n)) time complexity and with the smallest space complexity

Input:- [1,14,175,0,3,709]

output:- [0,1,3,14,175,709]

Pseudocode:-

```
def insert_sort(nums):
    for i in range(1, len(nums)):
        for j in range(i, 0, -1)
            if nums[j] < nums[j-1]:
                nums[j], nums[j-1] = nums[j-1], nums[j]
    return nums

nums = [1,14,175,0,3,709]
print(insert_sort(nums))
```

Result:-

output = [0,1,3,14,175,709]