DAY-01

Practice Questions

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.

Code:

```
def first_palindromic_string(words):
   for word in words:
      if word == word[::-1]:
       return word
   return ""
example_1 = ["abc", "car", "ada", "racecar", "cool"]
print(first_palindromic_string(example_1))
```

Output: ada

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].

Code:

```
def calculate_values(nums1, nums2):
      answer1 = sum(1 for num in nums1 if num in nums2)
      answer2 = sum(1 for num in nums2 if num in nums1)
   return [answer1, answer2]
nums1 = [2, 3, 2]
nums2 = [1, 2]
print(calculate_values(nums1, nums2))
```

Output: [2, 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.

Code:

```
def sum_of_squares_of_distinct_counts(nums):
    n = len(nums)
    result = 0
    for start in range(n):
        freq = {}
        distinct_count = 0
        for end in range(start, n):
            if nums[end] not in freq:
                freq[nums[end]] = 0
                distinct_count += 1
            freq[nums[end]] += 1
            result += distinct_count ** 2
    return result
nums = [1, 2, 1]
output = sum_of_squares_of_distinct_counts(nums)
print(output)
```

Output: 15

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.

Code:

```
def count_valid_pairs(nums, k):
    from collections import defaultdict
    indices_map = defaultdict(list)
    for index, num in enumerate(nums):
        indices_map[num].append(index)
    count = 0
    for indices in indices_map.values():
        for i in range(len(indices)):
            for j in range(i + 1, len(indices)):
                if (indices[i] * indices[j]) % k == 0:
                    count += 1
    return count
nums = [3, 1, 2, 2, 2, 1, 3]
k = 2
output = count_valid_pairs(nums, k)
print(output)
```

Output: 4

5. 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

Code:

```
def count_pairs(nums):
    n = len(nums)
    return n * (n - 1) // 2
 nums = [1, 2, 3, 4, 5]
output = count_pairs(nums)
print(output)
```

Output: 5

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. 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

Code:

```
def process_numbers(numbers):
    return "The list is empty." if not numbers else max(sorted(numbers))
test_cases = [
    [],
    [5],
    [3, 3, 3, 3, 3]
]
for case in test_cases:
    result = process_numbers(case)
    print(f"Input: {case}, Output: {result}")
```

Output:

Input: [], Output: The list is empty.
Input: [5], Output: 5
Input: [3, 3, 3, 3, 3], Output: 3

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?

Code:

```
def unique_elements(numbers):
    unique_set = set(numbers)
   return list(unique_set)
input_list = [3, 7, 3, 5, 2, 5, 9, 2]
output = unique_elements(input_list)
print(f " Output: {output}")
```

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

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

Code:

```
def bubble_sort(arr):
   n = len(arr)
   for i in range(n):
      for j in range(0, n-i-1):
         if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
   return arr
input_array = [64, 34, 25, 12, 22, 11, 90]
sorted_array = bubble_sort(input_array)
print(f "Sorted Output: {sorted_array}")
```

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

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

Code:

```
def binary_search(arr, key):
```

```
    arr.sort()
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right)
        if arr[mid] == key:
            return mid
        left = mid + 1 if arr[mid] < key else left
        right = mid - 1 if arr[mid] > key else right
    return -1
X = [3, 4, 6, -9, 10, 8, 9, 30]
key = 10
position = binary_search(X, key)
print(f"Element {key} is found at position {position}." if position != -1 else f"Element {key} is not found.")
```

Output:
Element 10 is found at position 6.

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(nlog(n)) time complexity and with the smallest space complexity possible.

Code:

```
def quicksort(arr, low=0, high=None):
    if high is None:
        high = len(arr) - 1
    if low < high:
        pivot_index = partition(arr, low, high)
        quicksort(arr, low, pivot_index - 1)
        quicksort(arr, pivot_index + 1, high)

def partition(arr, low, high):
    pivot = arr[high]
    i = low
    for j in range(low, high):
        if arr[j] < pivot:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1
    arr[i], arr[high] = arr[high], arr[i]
    return i
nums = [3, 7, 2, 5, 10, -1, 4]
quicksort(nums)
print(f"Sorted Output: {nums}")
```

Output:
Sorted Output: [-1, 2, 3, 4, 5, 7, 10]