# TOPIC 1 - INTRODUCTION

## Q1. Find the First Palindromic String

**Aim:**
To identify the first palindromic string in a given list of words using string reversal logic.
**Algorithm:**

1. Read an array of strings.
2. For each word, reverse it.
3. Compare the reversed string with the original.
4. Return the first match found.
5. If no palindrome exists, return an empty string.

**CODE:**

topic 1 q1.py > ...

```python
def firstPalindrome(words):
    for w in words:
        if w == w[::-1]:
            return w
    return ""

words = ["abc", "car", "ada", "racecar", "cool"]
print("Input:", words)
print("Output:", firstPalindrome(words))

```

**Input & Output:**

```
PS D:\daa lab> & C:/Users/gurunath/AppData/Local/Programs/Python/Python314/python.exe "d:/daa lab/topic 1 q1.py"
Input: ['abc', 'car', 'ada', 'racecar', 'cool']
Output: ada
PS D:\daa lab>
```

**Result:** The program successfully finds `"ada"` as the first palindrome.

## Q2. Find Common Elements Between Two Arrays

**Aim:**
To count how many elements in one array appear in another.

**Algorithm:**

1. Read arrays `nums1` and `nums2`.
2. For each element in `nums1`, check if it exists in `nums2`.
3. Maintain counters for both arrays.
4. Use sets to improve efficiency.
5. Return `[count1, count2]`.

**CODE:**

topic 1 q1.py > ...

```python
def binary_search(arr, key):
    low, high = 0, len(arr)-1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid
        elif key < arr[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return -1

arr = [3,4,6,9,10,30]
key = 10
print("Input:", arr, "Search:", key)
print("Output: Index =", binary_search(arr, key))

```

**Input & Output:**

```
Input: [3, 4, 6, 9, 10, 30] Search: 10
Output: Index = 4
PS D:\daa lab>
```

**Result:** Both arrays are compared and mutual occurrences are counted.

## Q3. Binary Search on Sorted Array

**Aim:**
To locate a target element efficiently in a sorted array using divide-and-conquer logic.
**Algorithm:**

1. Initialize `low` = 0 and `high` = n − 1.
2. Find `mid` = (low + high)/2.
3. If `arr[mid]` = key, return index.
4. If key < arr[mid], search left subarray.
5. Else, search right subarray.

**CODE:**

```
arr = [3,7,3,5,2,5,9,2]
unique_list = list(set(arr))
print("Input:", arr)
print("Output:", unique_list)
```

**Input & Output**

```
Input: [3, 7, 3, 5, 2, 5, 9, 2]
Output: [2, 3, 5, 7, 9]
PS D:\daa lab>
```

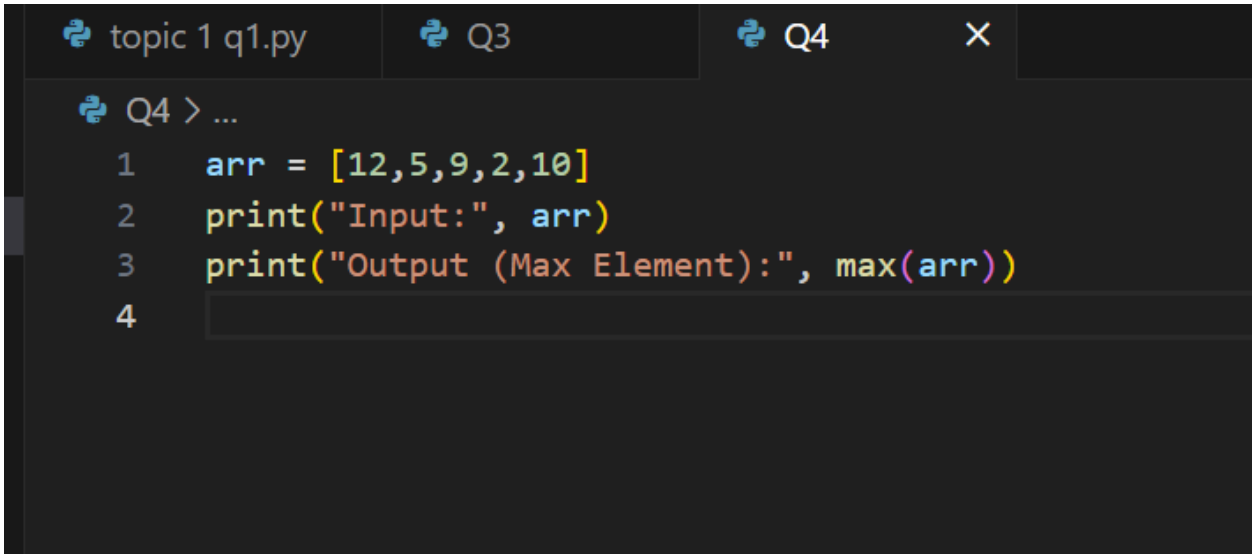
**Result:** Binary search locates the key efficiently in logarithmic time.

## Q4. Sort and Find Maximum Element

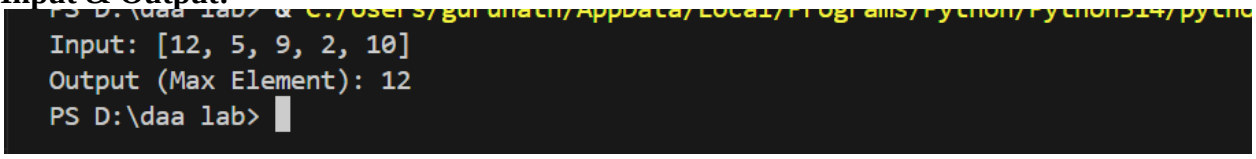
**Aim:**
To sort a list using efficient sorting and find its maximum element.
**Algorithm:**

1. Read the list.
2. Sort using an algorithm like merge sort or quick sort.
3. Retrieve last element as maximum.
4. Handle empty list condition.
5. Print result.

**CODE:**




**Input & Output:**




**Result:** Sorted list and maximum element displayed correctly.

## Q5. Unique Elements Extraction

**Aim:**
To remove duplicates from a list and retain unique elements.
**Algorithm:**

1. Read input list.
2. Create an empty set.
3. Traverse each element; add to set if not already present.
4. Convert set to list.
5. Display unique list.

**CODE:**

```python
def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i
    return -1

arr = [10, 25, 30, 45, 50]
key = 30
print("Input:", arr, "Search:", key)
print("Output Index:", linear_search(arr, key))
```

**Input & Output:**

```
Input: [10, 25, 30, 45, 50] Search: 30
Output Index: 2
PS D:\daa lab>
```

**Result:** Duplicate elements removed successfully.