1. First Palindromic String in Array

Aim:
To find the first palindromic string in a given list of words.

Algorithm:

- Define a function is_pal to check if a word is palindrome (word == word[::-1]).
- Iterate through each word in the list.
- If a word is palindrome, return it.
- If no palindrome exists, return empty string "".

Programming Code:

```
def is_pal(s):
    return s == s[::-1]

def first_palindrome(words):
    for w in words:
        if is_pal(w):
            return  w  return ""

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

print(first_palindrome(words))
```

Result:

The program correctly identifies the first palindrome `'ada'`.

2. Count Matches Between Two Arrays

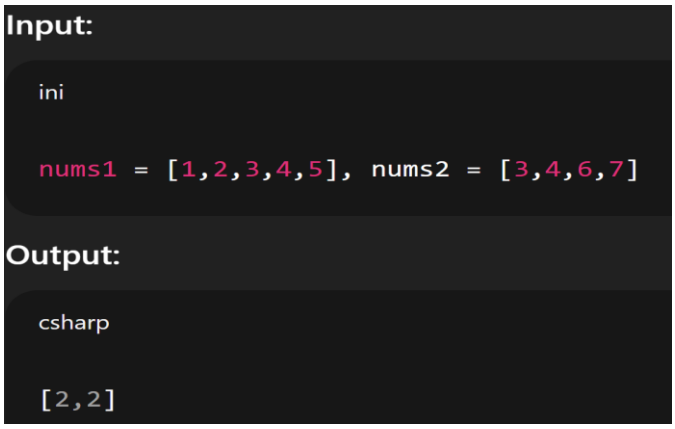
Aim:
To count how many elements of one array exist in another and vice versa.

Algorithm:

1. Iterate through `nums1`, check if each element exists in `nums2` → count1.
2. Iterate through `nums2`, check if each element exists in `nums1` → count2.
3. Return `[count1, count2]`.

Programming:

```
def count_matches(nums1, nums2):

    ans1 = sum(1 for x in nums1 if x in nums2)

    ans2 = sum(1 for x in nums2 if x in nums1)

    return [ans1, ans2]

nums1 = [1,2,3,4,5]

nums2 = [3,4,6,7]

print(count_matches(nums1, nums2))
```



Result:

The program correctly counts elements existing in the other array.

3. Sum of Squares of Distinct Counts of Subarrays

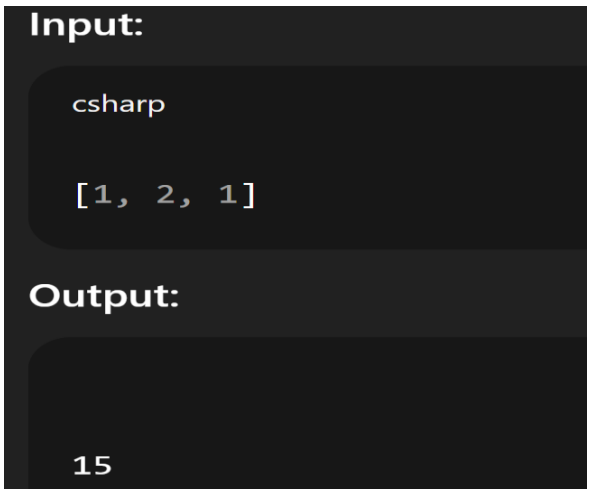
Aim:
To calculate the sum of squares of distinct element counts in all subarrays.

Algorithm:

1. For each start index `i`, initialize an empty set `seen`.
2. For each end index `j >= i`, add `nums[j]` to `seen`.
3. Add `(len(seen))^2` to total.
4. Repeat for all subarrays.

Programming:

```
def sum_of_squares(nums):
    n = len(nums)
    total = 0
    for i in range(n):
        seen = set()
        for j in range(i, n): seen.add(nums[j])
            total += len(seen)**2  return total
print(sum_of_squares([1,2,1]))
```



Result:

The program computes sum of squares of distinct elements in all subarrays.

4.Count Valid Pairs in Array

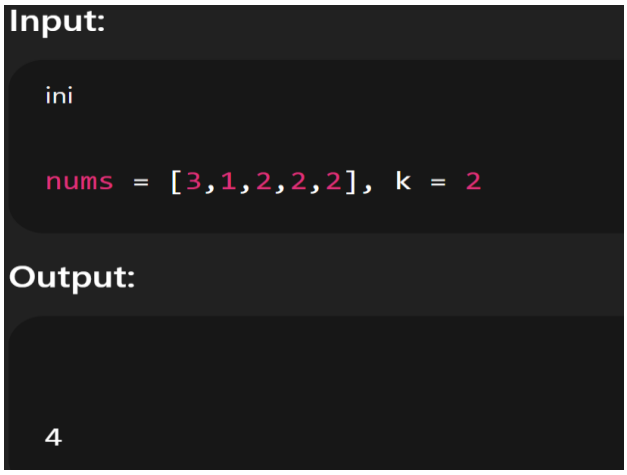
Aim:
To count pairs (i,j) such that nums[i] == nums[j] and (i*j) divisible by k.

Algorithm:

1. Iterate over all pairs (i,j) with i<j.
2. Check if nums[i] == nums[j] and (i*j) % k == 0.
3. Count all valid pairs.

Programming:

```
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
print(count_pairs([3,1,2,2,2], 2))
```



Result:

The program correctly counts valid pairs.

5.Maximum Element in Array

Aim:

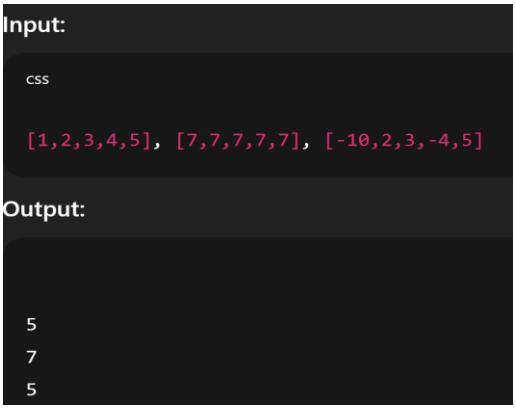
To find the largest element in an array.

Algorithm:

1. Initialize max_num = nums[0].
2. Compare each element with max_num.
3. Update max_num if a larger element is found

Programming:

```
def find_max(nums):

    return max(nums)

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

print(find_max([7,7,7,7,7]))

print(find_max([-10,2,3,-4,5]))
```



Result:

The program finds the maximum element correctly in all arrays.

6. Sort and Find Maximum Element

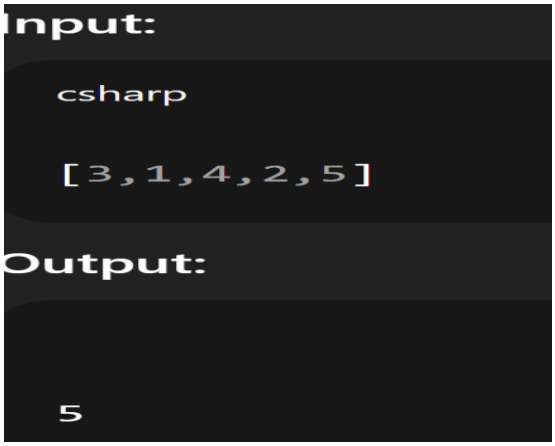
Aim:
To sort an array and find the maximum element.

Algorithm:

1. Sort the array using sort().
2. Maximum element is the last element arr[-1].

Programming:

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

nums.sort()

max_num = nums[-1]

print(max_num)
```



Result:

The program sorts and finds the maximum element.

7. Extract Unique Elements from List

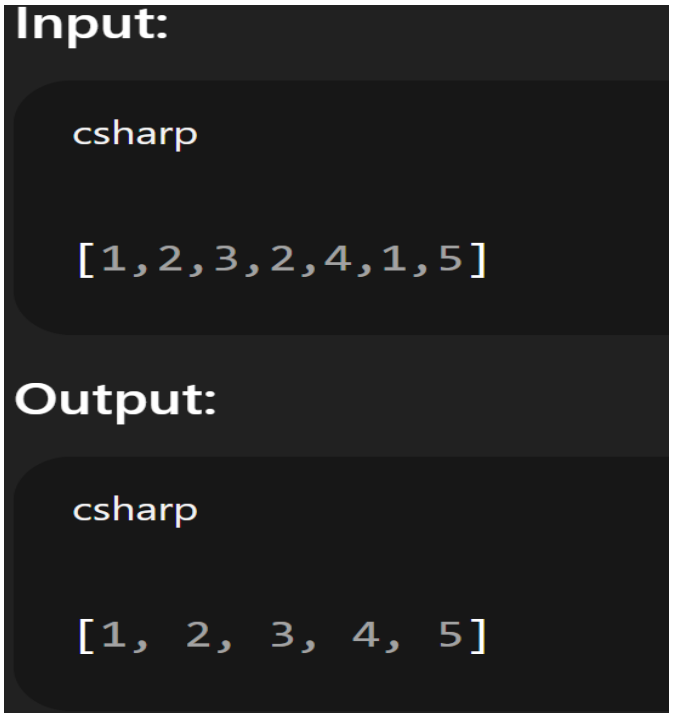
Aim:
To create a new list with only unique elements.

Algorithm:

1. Convert list to set → removes duplicates.
2. Convert back to list.

Programming:

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

unique_nums = list(set(nums))

print(unique_nums)
```



Result:

The program correctly extracts unique elements.

8. Bubble Sort

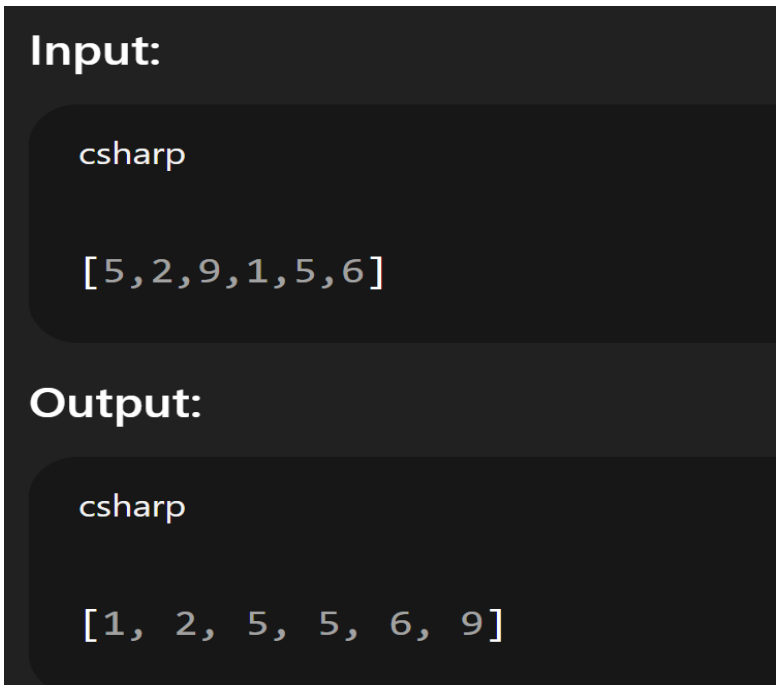
Aim:
To sort an array using bubble sort.

Algorithm:

1. Repeat `n-1` passes.
2. In each pass, compare adjacent elements and swap if out of order.
3. After each pass, largest element "bubbles" to the end.

Programming:

```
nums = [5,2,9,1,5,6]

n = len(nums)

for i in range(n-1):

    for j in range(n-1-i):

        if nums[j] > nums[j+1]:

            nums[j], nums[j+1] = nums[j+1], nums[j]

print(nums)
```



Result:

Array sorted correctly using bubble sort.

9. Binary Search

Aim:
To check if a number exists in a sorted array using binary search.

Algorithm:

1. Initialize left=0, right=n-1.
2. While left <= right:
    - Compute mid = (left+right)//2.
    - If arr[mid] == x → found.
    - If arr[mid] < x → search right half.
    - Else → search left half.

Programming:

```
Listd=[1,3,5,7,9,45]

Listd.sort()

Keys=int(input(“enter the key:”)

Found=false

L=0

H=len(listd)-1

While l<=h:

        Mid=(l+h)//2

        If listd[mid]==keys:

            Found=true

            Break

        Elif keys>listd[mid]:

             L=mid+1

        Else:

             H=mid-1

If found==true:

  Print(“key is found”)

Else:

    Print(“keys is not found”)
```

Input:

```ini
arr = [1,3,5,7,9,11], x = 7
```

Output:

```pgsql
7 exists in the array.
```

Result:

Binary search correctly finds if the number exists.

10. Merge Sort (Manual O(n log n) Sorting)

Aim:
To sort an array without using built-in functions in O(n log n) time.

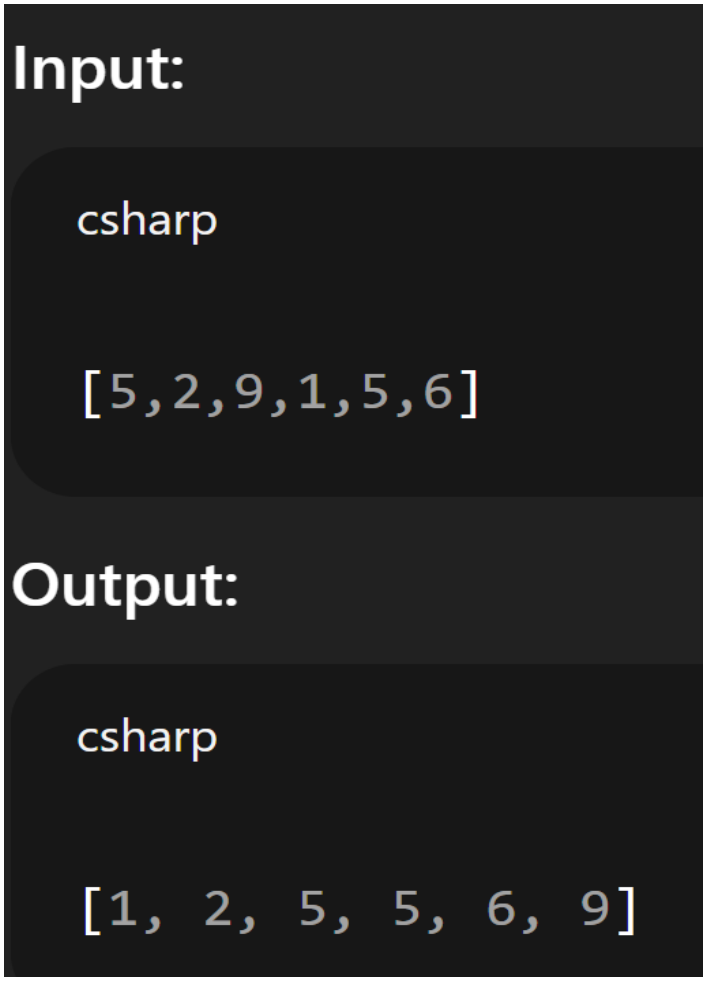
Algorithm:

1. If array length >1, split array into left and right halves.
2. Recursively sort left and right halves.
3. Merge the sorted halves into a single sorted array.

Programming:

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
```

```
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
nums = [5,2,9,1,5,6]
merge_sort(nums)
print(nums)
```



Result:

The program sorts the array correctly in O(n log n) time.