

Topic 1- Introduction

Q1. First Palindromic String

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

Algorithm:

1. Take input list of words.
2. Traverse each word.
3. Check if word == reverse of word.
4. Print first palindrome found, else print "

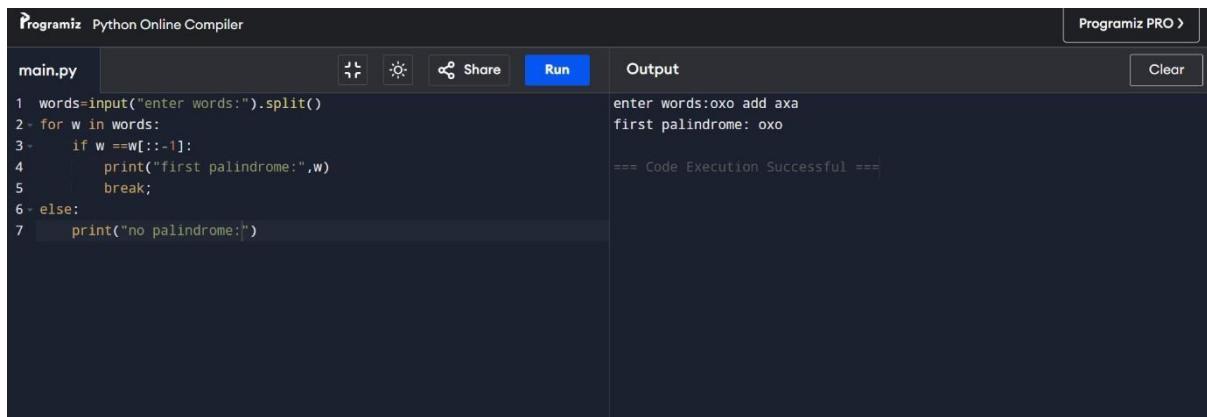
Code (Python):

```
words = input("Enter words: ").split() ans  
= "" for w in words: if w == w[::-1]:  
    ans = w break print("First
```

Palindrome:", ans) **Input:**

abc car ada racecar cool **Output:**

First Palindrome: ada



The screenshot shows a Python online compiler interface. The code in main.py is:

```
1 words=input("enter words:").split()  
2 for w in words:  
3     if w ==w[::-1]:  
4         print("first palindrome:",w)  
5         break;  
6 else:  
7     print("no palindrome:")
```

The output window shows the following execution results:

```
enter words:oxo add axa  
first palindrome: oxo  
==== Code Execution Successful ===
```

Q2. Count Common Indices in Two Arrays

Aim: To count how many elements of one list exist in another.

Algorithm:

1. Take input arrays nums1 and nums2.

2. For each element in nums1, check if in nums2 → count1.
3. For each element in nums2, check if in nums1 → count2.
4. Print [count1, count2].

Code (Python):

```
nums1 = list(map(int, input("Enter nums1: ").split()))
nums2 = list(map(int, input("Enter nums2: ").split()))
set1, set2 = set(nums1), set(nums2) answer1 = sum(1
for x in nums1 if x in set2) answer2 = sum(1 for x in
nums2 if x in set1) print("Output:", [answer1, answer2])
```

Input:

Enter nums1: 4 3 2 3 1 Enter nums2:

2 2 5 2 3 6

Output:

[3, 4]

The screenshot shows the Programiz Python Online Compiler interface. The code in the editor is:

```
main.py
1 nums1=list(map(int,input("nums1:").split()))
2 nums2=list(map(int,input("nums2:").split()))
3 ans1=sum(1 for x in nums1 if x in nums2)
4 ans2=sum(1 for x in nums2 if x in nums1)
5 print([ans1 ,ans2])
```

The output window shows the results of the execution:

```
Build with AI. Win prizes. Get featured on the Wall. Join Challenge X
Programiz PRO >
main.py
nums1:2 3 2
nums2:1 2
[2, 1]
== Code Execution Successful ==
```

Q3. Sum of Squares of Distinct Counts of Subarrays Aim:

To calculate sum of (distinct_count²) for all subarrays.

Algorithm:

1. Take input array nums.
2. For each starting index i → create empty set.
3. For each ending index j → add element, count distincts.
4. Add square of count to total.

5. Print total.

Code (Python):

```
nums = list(map(int, input("Enter numbers: ").split()))
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 print("Sum of squares:", total)
```

Input:

Enter array: 1 2 1

Output:

Result: 15

main.py	Run	Output
1 nums = list(map(int, input("Enter numbers: ").split())) 2 n = len(nums) 3 total = 0 4 for i in range(n): 5 seen = set() 6 for j in range(i, n): 7 seen.add(nums[j]) 8 total += len(seen) ** 2 9 print("Sum of squares:", total) 10		Enter numbers: 1 2 1 Sum of squares: 15 ==== Code Execution Successful ===

Q4. Count Pairs with Condition

Aim: To count pairs (i, j) where $\text{nums}[i] == \text{nums}[j]$ and $(i*j) \% k == 0 \rightarrow \text{count}++$.

Algorithm:

1. Take nums and k.
2. Loop through all pairs ($i < j$).
3. If $\text{nums}[i] == \text{nums}[j]$ and $i*j \% k == 0 \rightarrow \text{count}++$.
4. Print count.

Code (Python):

```
nums = list(map(int, input("Enter numbers: ").split()))
k = int(input("Enter k: "))
count = 0
n = len(nums)
for i in range(n):
    for j in range(i+1, n):
        if nums[i] == nums[j] and (i*j) % k == 0:
            count += 1
```

```
== nums[j] and (i*j) % k == 0:
```

```
count += 1 print("Pairs
```

```
count:", count) Input: Enter
```

```
nums: 3 1 2 2 2 1 3
```

```
Enter k: 2 Output:
```

```
Pairs: 4
```

The screenshot shows a code editor interface with a dark theme. On the left, the code file is named 'main.py'. The code itself is as follows:

```
main.py
1  nums = list(map(int, input("Enter numbers: ").split()))
2  k = int(input("Enter k: "))
3  count = 0
4  n = len(nums)
5  for i in range(n):
6      for j in range(i+1, n):
7          if nums[i] == nums[j] and (i*j) % k == 0:
8              count += 1
9  print("Pairs count:", count)
10 
```

On the right, there is an 'Output' panel. It contains the user input and the program's output:

```
Enter numbers: 3 1 2 2 2 1 3
Enter k: 2
Pairs count: 4
== Code Execution Successful ==
```

Q5. Max of Array (with given test cases)

Aim: To find maximum element in array with least time complexity.

Algorithm:

1. Take input array.
2. Use built-in max() for efficiency.
3. Print max element.

Code (Python):

```
arr = list(map(int, input("Enter numbers: ").split())) if
not arr:    print("Array is empty") else:    max_val
= arr[0]    for num in arr:        if num > max_val:
max_val = num    print("Maximum:",
max_val)
```

Input:

```
Enter array: -10 2 3 -4 5
```

Output:

```
Max: 5
```

```
main.py
1 arr = list(map(int, input("Enter numbers: ").split()))
2 if not arr:
3     print("Array is empty")
4 else:
5     max_val = arr[0]
6     for num in arr:
7         if num > max_val:
8             max_val = num
9     print("Maximum:", max_val)
10
```

Output

```
Enter numbers: -5 5 3 -1 10
Maximum: 10
== Code Execution Successful ==
```

Q6. Sort and Find Maximum

Aim: Sort array and return maximum element. **Algorithm:**

1. Input array.
2. Sort array using efficient algorithm (Python uses Timsort).
3. Print sorted list and last element as max.

Python Code:

```
arr = list(map(int, input("Enter numbers: ").split())) if
not arr:    print("Array is empty") else:
    arr.sort()    print("Maximum after sorting:",
arr[-1])
```

Input: 3 3 3 3 3 **Output:**

Sorted: [3, 3, 3, 3, 3]

Max: 3

```
main.py
1 arr = list(map(int, input("Enter numbers: ").split()))
2 if not arr:
3     print("Array is empty")
4 else:
5     arr.sort()
6     print("Maximum after sorting:", arr[-1])
7
```

Output

```
Enter numbers: 1 2 3 4 5
Maximum after sorting: 5
== Code Execution Successful ==
```

Q7. Unique Elements of List

Aim: Remove duplicates and return only unique elements. **Algorithm:**

1. Input list.
2. Convert to set (removes duplicates).
3. Convert back to list.
4. Print. **Python Code:**

```
arr = list(map(int, input("Enter numbers: ").split()))
unique = list(set(arr)) print("Unique elements:", unique)
```

Input: 3 7 3 5 2 5 9 2

Output: Unique: [2, 3, 5, 7, 9] (order may vary)

Space Complexity: O(n) because a set stores unique elements.

```
1 arr = list(map(int, input("Enter numbers: ").split()))
2 unique = list(set(arr))
3 print("Unique elements:", unique)
4 
```

Enter numbers: 3 7 3 5 2 5 9 2
Unique elements: [2, 3, 5, 7, 9]
== Code Execution Successful ==

Q8. Bubble Sort

AIM:

Sort an array of integers using the **Bubble Sort** technique.

Algorithm/Method:

1. Start from the first element, compare it with the next element.
2. If the first element is greater, swap them.
3. Move to the next element and repeat the process for the whole array.
4. Repeat the above steps for all elements until no swaps are needed (array is sorted).

Input:

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

[11, 12, 22, 25, 34, 64, 90]

Code:

```
def bubble_sort(arr):    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    return arr
```

arr

```
arr = [64, 34, 25, 12, 22, 11, 90] print("Sorted array:", bubble_sort(arr))
```

```
main.py
```

```
1 arr = list(map(int, input("Enter numbers: ").split())) 2 n = len(arr) 3 for i in range(n): 4     for j in range(0, n-i-1): 5         if arr[j] > arr[j+1]: 6             arr[j], arr[j+1] = arr[j+1], arr[j] 7 print("Sorted array:", arr) 8 # Time Complexity: O(n^2)
```

Run Output

```
Enter numbers: 64 34 25 12 22 11 90
```

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

```
*** Code Execution Successful ***
```

Q9. Binary Search

AIM:

Check if a given number exists in a sorted array using **Binary Search**.

Algorithm/Method:

1. Sort the array (if not already sorted).
2. Set low = 0 and high = n-1.
3. Find mid = (low + high)//2.
4. If arr[mid] == key, element is found.
5. If arr[mid] < key, search in the right half.
6. If arr[mid] > key, search in the left half.

7. Repeat until element is found or low > high.

INPUT:

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

Key = 10

OUTPUT:

Element 10 is found at position 6

CODE:

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

arr = [3, 4, 6, -9, 10, 8, 9, 30]
key = 10
pos = binary_search(arr, key)
if pos
    != -1:
        print(f"Element {key} is found at position {pos}")
    else:
        print(f"Element {key} is not found")
```

The screenshot shows a Python online compiler interface. The code in the editor is a binary search algorithm. The output window displays the user input and the program's response.

```
Program2 - Python Online Compiler
main.py | Run | Output | Clear
1 arr = list(map(int, input("Enter numbers: ").split()))
2 key = int(input("Enter key to search: "))
3 arr.sort() # Binary search needs sorted array
4 low, high = 0, len(arr)-1
5 found = -1
6 while low <= high:
7     mid = (low + high) // 2
8     if arr[mid] == key:
9         found = mid
10        break
11    elif arr[mid] < key:
12        low = mid + 1
13    else:
14        high = mid - 1
15 if found != -1:
16     print(f"Element {key} found at position {found}")
17 else:
18     print(f"Element {key} not found")
19 # Time Complexity: O(log n)
20
```

Output:

```
Enter numbers: 10 9 7 4 6 3 8 3
Enter key to search: 4
Element 4 found at position 2
== Code Execution Successful ==
```

Q10. Merge Sort ($O(n \log n)$ Sorting)

AIM:

Sort an array of integers in ascending order using **Merge Sort** without built-in functions.

Algorithm/Method:

1. Divide the array into two halves recursively until each sub-array has one element.
2. Merge two sorted arrays by comparing elements one by one.
3. Repeat merging until the entire array is sorted.

Input:

[5, 2, 9, 1, 5, 6] **Output:**

[1, 2, 5, 5, 6, 9]

Code:

```

def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

```

```

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

```

```

nums = [5, 2, 9, 1, 5, 6]
print("Sorted array:", merge_sort(nums))

```

The screenshot shows the Python Online Compiler interface. The code in `main.py` is identical to the one above. In the Output panel, the user enters the numbers `5 2 9 1 5 6`, and the program outputs the sorted array `[1, 2, 5, 5, 6, 9]`. A message at the bottom indicates `== Code Execution Successful ==`.

```

main.py
1 def merge_sort(arr):
2     if len(arr) > 1:
3         mid = len(arr)//2
4         L = arr[:mid]
5         R = arr[mid:]
6         merge_sort(L)
7         merge_sort(R)
8         i = j = k = 0
9         while i < len(L) and j < len(R):
10            if L[i] < R[j]:
11                arr[k] = L[i]
12                i += 1
13            else:
14                arr[k] = R[j]
15                j += 1
16            k += 1
17         while i < len(L):
18             arr[k] = L[i]
19             i += 1
20             k += 1
21         while j < len(R):
22             arr[k] = R[j]
23             j += 1

```

