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.

Input: words = ["abc","car","ada","racecar","cool"].

Output: "ada" Explanation: The first string that is palindromic is "ada". Note that "racecar" is also palindromic, but it is not the first.

Aim: To find the first palindromic string in the array
Algorithm:
step1: intialize a function that takes an array 'words' as input
step2: iterate through each string in array
step3: check if string is palindromeic
step4: return the first palindromic string
Program:

```
1  words = ["abc","car","ada","racecar","cool"]
2  print(next((word for word in words if word == word[::-1]), ""))
```

ada

Input:

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

output:

```
Output

ada
```

Time complexity: T(n)=O(n)
Result: First palindromic string in the array program is succesfull executed.

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].

Example :
Input: nums1 = [4,3,2,3,1], nums2 = [2,2,5,2,3,6]
Output: [3,4]
Explanation:
The elements at indices 1, 2, and 3 in nums1 exist in nums2 as well. So answer1 is 3.
The elements at indices 0, 1, 3, and 4 in nums2 exist in nums1. So answer2 is 4.

Aim: To calculate the counts of indices where element from array 1 exist in array 2 and vice versa

Algorithm:
step 1: intialize 2 sets : set 1 to store elements from num1 and set 2 to store from num 2
step 2:iterate through num1 and num2 to populate set1 and set2
step 3:intialize counters ans 1 and ans2 to zero
step 4:iterate through num1 and num2

step 5: return the result

Program:

```
def find_matching_indices(nums1, nums2):
    set_nums2 = set(nums2)
    answer1 = sum(1 for num in nums1 if num in set_nums2)
    set_nums1 = set(nums1)
    answer2 = sum(1 for num in nums2 if num in set_nums1)
    return [answer1, answer2]
nums1 = [4, 3, 2, 3, 1]
nums2 = [2, 2, 5, 2, 3, 6]
print(find_matching_indices(nums1, nums2))
```

Input:

```
nums1 = [4, 3, 2, 3, 1]
nums2 = [2, 2, 5, 2, 3, 6]
```

Output:

```
[3, 4]
```

Time complexity:T(n)=O(n)

Result :counting the indicies program is successfully executed

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.

Input: nums = [1,2,1]

Output: 15

Explanation: Six possible subarrays are: [1]: 1 distinct value [2]: 1 distinct value [1]: 1 distinct value [1,2]: 2 distinct values [2,1]: 2 distinct values [1,2,1]: 2 distinct values The sum of the squares of the distinct counts in all subarrays is equal to 12 + 12 + 12 + 22 + 22 + 22 = 15.

Aim: To calculate the sum of squares of distinicts count of all possible sub array of given integer array.

Algorithm:

step1: Intilize varaible and declare an array.

step2: Ilerateover each starting point of sub arrays.

step3:track element using freuency camp.

step4:expand subarray from sorting point.

step5:calculate distint count and update result.

step6:return the result.

Program:

```python
def sumOfSquareDistinctCounts(nums):
    n = len(nums)
    # Initialize an array to store the number of distinct values in each subarray ending at
        that index
    dp = [0] * (n + 1)
    # Initialize an empty set to store the current subarray values
    st = set()
    # Initialize the sum to 0
    sum = 0
    # Iterate over the array
    for i in range(n):
        st.add(nums[i])
        dp[i + 1] = dp[i] + len(st)
    for i in range(n + 1):
        for j in range(1, n - i + 1):
            sum += (dp[i] - dp[i - j]) ** 2
    return sum

print(sumOfSquareDistinctCounts([1,2,1]))  # Output: 15
```

Input:

```python
print(sumOfSquareDistinctCounts([1,2,1]))
```

Output:

```
56
```

Time complexity: T(n)=O(n^2)

Result: sum of squares of distincts counts of all possible subarrays of given integer array is excuted

4.Given a 0-indexed integer array nums of length n and an integer k, return the number ofpairs (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

Explanation:There are 4 pairs that meet all the requirements:-

-nums[0] == nums[6], and 0 *== 0, which is divisible by 2.

- nums[2] == nums[3], and 2 * 3 == 6, which is divisible by 2.

- nums[2] == nums[4], and 2 * 4 == 8, which is divisible by 2.

- nums[3] == nums[4], and 3 * 4 == 12, which is divisible by 2.

AIM: to count pairs in array nums such that num[i]=num[j] and(i*j)/k==0 for 0<=i<j<n

ALGORITHM:

STEP1: initialize a counter variables

STEP2:iterate over all possible pairs

STEP3:for each pair check if both are equal

STEP4:if both equal check if(i*j)%k==0

STEP5:if both conditions are satisfied increament the count

STEP6:after iterating overall possible pairs return the count

PROGRAM:

```
1  nums = [3, 1, 2, 2, 2, 1, 3]
2  k = 2
3  count = sum(1 for i in range(len(nums)) for j in range(i+1, len
       (nums)) if nums[i] == nums[j] and (i * j) % k == 0)
4  print(count)
```

INPUT:

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

OUTPUT:

```
Output
4
```

TIME COMLEXITY:T(n)=O(n^n)

RESULT: counting pairs in array numbers are successfully executed.

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

Aim:To solve the problem of finding the maximum value in array eith least time complexity

Algorithm:

step1: intialize an array.
step 2:intialize maximum elements in an array
step 3:iterate through array using for loop
step 4: update maximum for each elements
step 5:return result value

Program:

```
def find_maximum_value(lst):
    if not lst:
        return None
    max_value = lst[0]
    for num in lst[1:]:
        if num > max_value:
            max_value = num
    return max_value
print(find_maximum_value([1, 2, 3, 4, 5]))
```

Input:

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

Output:

```
5
```

Time Complexity: T(n)=O(n)

Result: Maximum value in the array is successfully returned with less time complexity

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

Input: [5]

Expected Output: 5

Aim: To Solve the problem of Finding the Maximum Value in array of Integers With Least Time Complexity.

Algorithm:

Step-1: Intialize an Array.

Step-2: Intialize Maximum Element of an Aray.

Step-3: Iterate Through Array using for Loop.

Step-4: Update Maximum for each Element.

Step-5: Return Result Value.

Program:

```
def sort_and_find_max_alternative(input_list):
    sorted_list = sorted(input_list)
    max_element = sorted_list[-1]
    return max_element
input_list = [1,2,3,4,5]
result = sort_and_find_max_alternative(input_list)
print(result)
```

Input:

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

Output:

```
Output
5
```

Time Complexity: T(n)=O(n)
Result: Maximum Value in an Array with Least Time Complexity is Successfully Executed.

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? Test Cases
Some Duplicate Elements
🕐 Input: [3, 7, 3, 5, 2, 5, 9, 2] 🕐
Expected Output: [3, 7, 5, 2, 9] (Order may vary based on the algorithm used)
Negative and Positive Numbers
🕐 Input: [-1, 2, -1, 3, 2, -2]
🕐 Expected Output: [-1, 2, 3, -2] (Order may vary)
List with Large Numbers
🕐 Input: [1000000, 999999, 1000000]
🕐 Expected Output: [1000000, 999999]

Aim: to create a new list containing only unique element from the original list
Algorithm:
step1:conver input list to set which automatically remove duplicate.
step2:convert the set bact to list.
Program:

```
def unique_elements(nums):
    return list(set(nums))

# Test cases
print(unique_elements([3, 7, 3, 5, 2, 5, 9, 2]))
print(unique_elements([-1, 2, -1, 3, 2, -2]))
print(unique_elements([1000000, 999999, 1000000]))
```

Input:

```
print(unique_elements([3, 7, 3, 5, 2, 5, 9, 2]))
print(unique_elements([-1, 2, -1, 3, 2, -2]))
```

Output:

```
Output

[2, 3, 5, 7, 9]
[2, 3, -1, -2]
[1000000, 999999]
```

Time complexity:T(n)=O(n)
Result:The order of elements in the output list may vary as set data structure doesnot preserve original order of element

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

notation. Write the code

AIM: To creat a new list containing only the unique elements from original list.

ALGORITHM:

STEP1:convert input list to set which automatically remove duplicate

step2:convert the set back to list

PROGRAM:

```
arr = [64, 34, 25, 12, 22, 11, 90]

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]

print("Sorted array is:", arr)
```

INPUT:

```
arr = [64, 34, 25, 12, 22, 11, 90]
```

OUTPUT:

```
Output

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

TIME COMPLEXITY: t(n)=O(n)

RESULT:the order of elements in the output list mayas set data structure does not preserve original

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

Test Case:

Example X={ 3,4,6,-9,10,8,9,30} KEY=10

Output: Element 10 is found at position 5

Aim: The aim is to check if a given number x exists in sorted array

Algorithm:

step 1:intialize low to 0 and high to length of array-1.

step 2:while low<=high

step 3:return -1

Program:

```python
def binary_search(arr, key):
    arr.sort()
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1
    return -1
arr = [3, 4, 6, -9, 10, 8, 9, 30]
key = 10
index = binary_search(arr, key)
if index != -1:
    print(f"Element {key} is found at position {index}")
else:
    print(f"Element {key} is not found")
```

Input:

```
[3, 4, 6, -9, 10, 8, 9, 30]
```

Output:

```
Element 10 is found at position 6
```

Time complexity:T(n)=O(log(n)

Result:The binnary search algorithm is efficient for searching the large sorted arrays with time complexity O(log(n))

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.

Aim:The aim is to implement a sorting algorithm that meets the given constant

Algorithm:

step1: Divide the array into new values until each subarray into one element
step2:Merge the subarray each together in sorted order.

Program:

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

def merge(left, right):
    result = []
    i, j = 0, 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

# Test the function
print(merge_sort([5, 2, 8, 1, 9]))  # Output: [1, 2, 5, 8, 9]
```

Input:

```python
print(merge_sort([5, 2, 8, 1, 9]))
```

Output:

```
Output
[2, 3, 5, 7, 9]
[2, 3, -1, -2]
[1000000, 999999]
```

Time complexity:T(n)=O(n(log(n)

Result: the output program is the sorted array in ascending order.