First Palindromic string

Aim :- To find the first palindromic string in an array of words.

Algorithm :-

1. Loop through the array of words.
2. for each word, check if it is a palindrome
3. If the word is palindrome, return it
4. If no palindrome is found, return an empty string.

Pseudo Code :-

```
Procedure first-Palindrome (words):
  for word in words:
    if word == reverse (word):
        return word
  return ""
```

Python Code :-

```
def first-Palindrome (words):
for Word in Words:
    if word == ~~reve~~word [::-1]:
      return word
return ""
words = ["abc", "car", "ada", "racecar", "cool"]
Print (first_Palindrome (words)).
```

Output :-

ada.

Count Common elements between two Arrays:

Aim:- To find the no of Common elements between two arrays.

Algorithm:

1. Read the first array, say nums1
2. Read the second array, say nums2
3. for answer 1, Count how many elements of num1 exist in num2.
4. For answer 2, Count how many elements of num2 exist in num1.
5. return the answer and display it.

Pseudo Code:-

```
Procedure Count_Common (nums1, nums2):
answer1 = Count elements in nums1 that are in nums2
answer 2= Count elements in nums 2 that are in nums1
return [answer 1, answer 2]
```

Python Code:-

```
def Count_Common (nums1, nums2):
answer 1 = Sum(1 for i in num1 if i in nums2)
answer 2 = Sum(1 for i in num2 if i in nums 1)
return [answer 1, answer 2]
nums1 = [2, 3, 2]
nums 2 = [1, 2]
Print (Count_ Common (nums 1, nums 2))
```

Output:-

[2, 1]

Sum of Squares of distinct Count in subarrays

Aim:- To calculate the Sum of squares of distinct Counts in all subarrays of an integer array

Algorithm:-

1. Generate all subarrays
2. Calculate the number of distinct elements in each subarray
3. Calculate Sum of the squares of distinct element Counts

Pseudo Code:-

```
Procedure Sum_of_Squares(nums):
 result = 0
for all subarrays:
 distinct_Count = Count distinct elements
 result += distinct_Count ^ 2
return result
```

Python Code:-

```
def Sum_of_squares(nums):
result = 0
n = len(nums)
for i in range(n):
distinct = &et()
for j in range(i,n):
distinct.add(nums[j])
result += len(distinct)** 2
return result

nums = [1, 2, 1]
print(sum_of_squares(nums))
```

Output:-

15

Pairs divisable by k

Aim:- To find pair of elements whose Product of indices of divisable by k

Algorithm:-

1. Iterate over all pairs (i,j) where i<j.
2. check if 'num[i] == num[j] and (i*j) %k == 0
3. Count each pairs.

Pseudo Code:-

```
Procedure Count(nums,k):
  Count=0
for i=0 to n-1:
 for j=i+1 to n-1:
if nums[i] == nums[j] and (i*j)%.k==0
  Count +=1
return Count
```

Python Code:-

```
def Count(nums,k):
Count = 0
for i in range(len(nums)):
for j in range(i+1, len(nums)):
 if nums[i] == nums[j] and (i*j)%.k==0;
 Count +=1
return Count
nums = [3,1,2,2,2,1,3]
K=2
Print(count(nums,k))
```

Output:-

4

Maximum Element in Array:-

Aim:- To find maximum element in an array.

Algorithm:-

1. Read an array, say arr.
2. Compare one element to another element
3. Transverse the array and keep track of the maximum value encountered.

Pseudo Code:-

```
Procedure max(arr):
    max_value = arr[0]
for element in arr:
if element > max_value:
 max_value = element
return max_value.
```

Python Code:-

```
def max(arr):
if not arr:
 return None.
return max(arr)
arr = [1,2,3,4,5]
Print (max(arr))
```

Output:-

5

Sorting and finding maximum

Aim:- To Sort an array and find it maximum.

Algorithm:-

1. Read an array, say arr
2. use a efficent Sorting algorithm.
3. Return the last element ie maximum element in the Sorted array.

Pseudo Code:-

```
Procedure Sort (arr):
sort_arr = sort (arr)
return Sorted_arr [-1].
```

Python Code:

```
def Sortand_max(arr):
if not arr:
    return None
arr.sort()
return arr[-1]

arr = [64, 34, 25, 12, 22, 11, 90]
Print (sort_and_max(arr)).
```

Output:-

90.

Unique elements in list:

Aim:- To Create a list of unique elements from an input list.

Algorithm:-

1. use a set to collect unique elements from the list.
2. Compare one element to another and if it same
3. return the element which is same is deleted.

Pseudo Code:-

```
Procedure unique (arr):
unique_list = [ ]
for element in arr:
if element not in unique_list:
unique_list.append (element)
return unique list.
```

Python Code:-

```
def unique (arr):
  return list (set (arr)
arr = [3,7,3,2,5,5,9,2]
Print (unique (arr))
```

Output:-

[3,7,5,2,9]

Bubble Sort

Aim:- To sort an array of integers using the bubble sort technique and analyze its time complexity.

Algorithm:-

1. Start with the first element of array
2. Compare it with the next element
3. If the current element is greater than the next one, swap them.
4. Repeat the process for the rest of the array until no swaps are needed in a complete the pass.

Pseudo Code:-

```
Procedure bubbleSort(arr):
n= length(arr)
for i=0 to n-1:
swapped= False
for j=0 to n-i-2:
if(arr[j] > arr[j+1]):
 swap(arr[j], arr[j+1])
swapped = True
if not swapped:
  break
return arr.
```

Python 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 = [64, 34, 25, 12, 22, 32, 11, 50]
sorted_arr = bubble_sort(arr)
Print("sorted array is", sorted_arr)
```

Output :-

[11, 12, 22, 25, 32, 34, 50, 64]

Binary Search

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

Algorithm:-

1. use binary search to find the element in the sorted array.
2. find the lowest element and the greatest element
3. mid = $\frac{low + high}{2}$. find the mid
4. if the mid greater than Search number and the mid is lesser than Search number (m-1)
   (m+1)

Pseudo Code :-

```
Procedure binary (arr, key):
left=0, right = len(arr)-1
while left <= right :
mid = (left + right )/2
if arr [mid] == key :
return mid.
elif arr[mid.] < key :
left = mid+1
else:
right = mid-1
return -1
```

Python Code:-

```
def binary(arr, key):
left, right = 0, len(arr)-1
while left<= right:
mid = (left + right) // 2
if arr[mid] == key:
return mid
elif arr[mid] < key:
left = mid+1
else:
right = mid-1
return -1
arr = sorted([1,5,-5,6,0,30,9,8])
Key = 10
Print (binary-search (arr, key))
```

Output:-

5

Sorting in O(nlogn)

Aim :- To Sort an array in O(n log n) time

Algorithm :-

1. Read an array, say arr
2. Use merge sort or quick sort, which has O(nlogn) complexity.
3. Use an efficent Sorting algorithm.
4. Return the last element in the sorted array.

Pseudo Code :-

```
Procedure quick(arr):
if len(arr)<=1:
return arr
Pivot = arr[len(arr)//2]
left = [x for x in arr if x<pivot]
middle = [x for x in arr if x==pivot]
right= [x for x in arr if x>pivot]
return quick(left) + middle + quick(right)
```

Python Code :-

```
def quick(arr):
if len(arr)<=1:
return arr
Pivot = arr[len(arr)//2]
left = [x for x in arr if x<pivot]
```

```
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]
return quick(left) + middle + quick(right)
arr = [2, 3, 6, 8, 10, 1, 1]
print (quick(arr))
```

Output:-

[1, 1, 2, 3, 6, 8, 10]