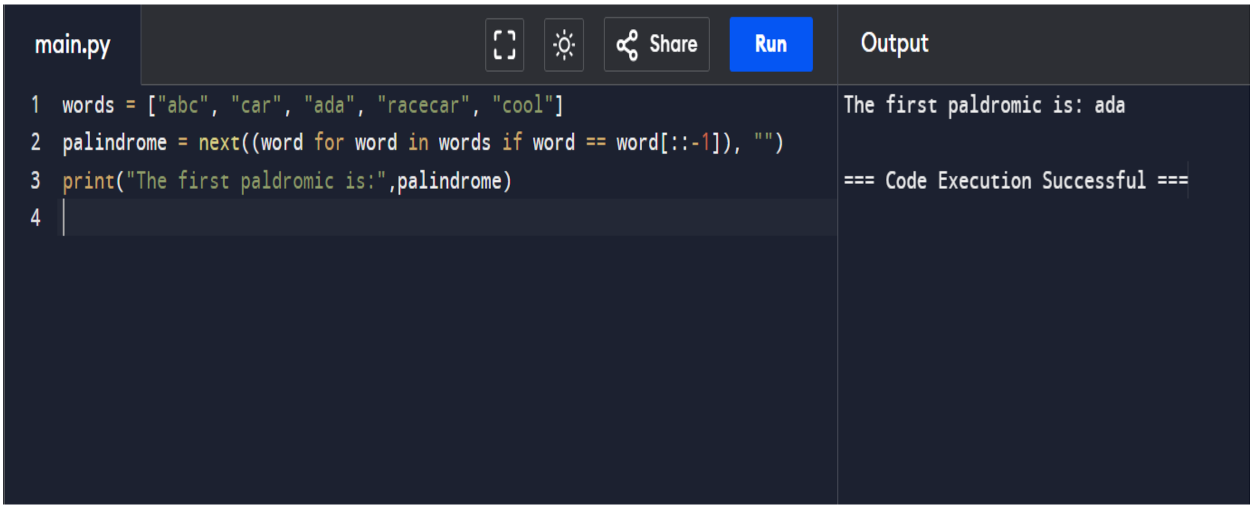
NAME:M.RAVI SAI VINAY

REG-NO: 192311035.

COURSE CODE:CSA0689.

SUBJECT: DESIGN ANALYSIS OF ALGORITHM.

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. Example 1: Input: words = ["abc","car","ada","racecar","cool"] Output: "ada"



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].

main.py

```python
nums1 = [2, 3, 2]
nums2 = [1, 2]

answer1 = len([i for i in nums1 if i in nums2])
answer2 = len([i for i in nums2 if i in nums1])

result = [answer1, answer2]
print(result)
```

Output

```
[2, 1]

=== Code Execution Successful ===
```

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. Example 1: Input: nums =

[1,2,1] Output: 15

main.py

```python
nums = [1, 2, 1]
n = len(nums)
result = 0

for i in range(n):
    distinct = set()
    for j in range(i, n):
        distinct.add(nums[j])
        distinct_count = len(distinct)
        result += distinct_count ** 2

print(result)
```

Output

```
15

=== Code Execution Successful ===
```

4.Given a 0-indexed integer array nums of length n and an integer k, return the number of pairs (i, j) where 0 <= i < j < n, such that nums[i] == nums[j] and (i * j) is divisible by k. Example 1: Input: nums = [3,1,2,2,2,1,3], k = 2 Output: 4

main.py

```python
nums = [3, 1, 2, 2, 2, 1, 3]
k = 2
n = len(nums)
```

Output

```
4

=== Code Execution Successful ===
```

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

main.py

```python
input_list = [1, 2, 3, 4, 5]
output = max(input_list)
print(output)

```

Output

```
5

=== Code Execution Successful ===
```

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.

main.py

```python
def sort_and_find_max(input_list):
    sorted_list = sorted(input_list)
    max_element = sorted_list[-1]
    return max_element

numbers = [5, 2, 8, 1, 9]
max_num = sort_and_find_max(numbers)
```

Output

```
Maximum element in the list: 9

=== Code Execution Successful ===
```

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?

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)

main.py

```python
def get_unique_elements(input_list):
    return list(set(input_list))

input_list = [3, 7, 3, 5, 2, 5, 9, 2]
unique_elements = get_unique_elements(input_list)
print(unique_elements)
```

Output

```
[2, 3, 5, 7, 9]

=== Code Execution Successful ===
```

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

main.py

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

n = len(arr)
for i in range(n):
    for j in range(0, n-i-1):
```

Output

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

=== Code Execution Successful ===
```

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

main.py

Share Run

```python
arr = [-9, 3, 4, 6, 8, 9, 10, 30]
key = 10

left = 0
right = len(arr) - 1
found = False

while left <= right:
    mid = left + (right - left) // 2
    if arr[mid] == key:
        print(f"Element {key} is found at position {mid}")
        found = True
        break
    elif arr[mid] < key:
        left = mid + 1
    else:
        right = mid - 1

if not found:
    print(f"Element {key} is not found in the array")
```

Output

```
Element 10 is found at position 6

=== Code Execution Successful ===
```

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.

```python
nums = [5, 2, 9, 1, 5, 6]
n = len(nums)
size = 1
while size < n:
    left = 0
    while left < n - size:
        mid = left + size - 1
        right = min((left + 2 * size - 1), (n - 1))

        left_sub = nums[left:mid + 1]
        right_sub = nums[mid + 1:right + 1]
```