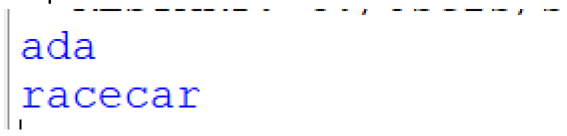
1. string of the palindrome.

Program:

```python
def is_palindrome(word):
    return word == word[::-1]

def first_palindromic_string(words):
    for word in words:
        if is_palindrome(word):
            return word
    return ""


words1 = ["abc", "car", "ada", "racecar", "cool"]
print(first_palindromic_string(words1))

words2 = ["notapalindrome", "racecar"]
print(first_palindromic_string(words2))
```

Output:



```
ada
racecar
```

2) TWO INTEGER ARRAYS

Program:

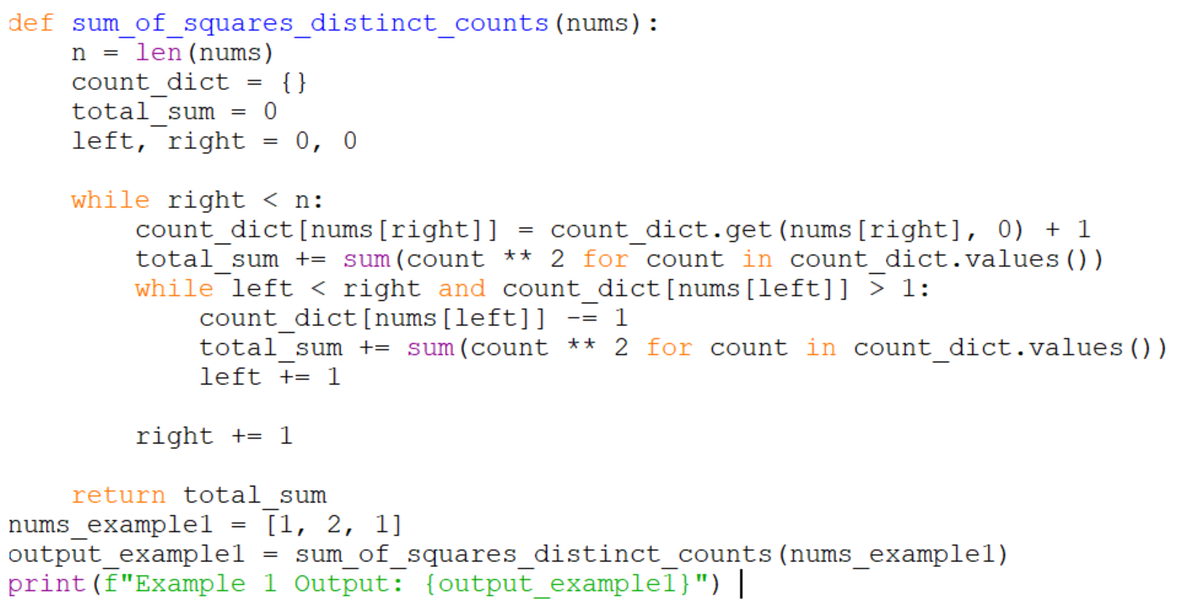
```python
def calculate_indices(nums1, nums2):
    set_nums1 = set(nums1)
    set_nums2 = set(nums2)
    answer1 = 0
    answer2 = 0
    for num in nums1:
        if num in set_nums2:
            answer1 += 1
    for num in nums2:
        if num in set_nums1:
            answer2 += 1

    return [answer1, answer2]
nums1_1 = [2, 3, 2]
nums2_1 = [1, 2]
output_1 = calculate_indices(nums1_1, nums2_1)
print(f"Example 1 Output: {output_1}")

nums1_2 = [4, 3, 2, 3, 1]
nums2_2 = [2, 2, 5, 2, 3, 6]
output_2 = calculate_indices(nums1_2, nums2_2)
print(f"Example 2 Output: {output_2}")
```

Output:

```
Example 1 Output: [2, 1]
Example 2 Output: [3, 4]
```

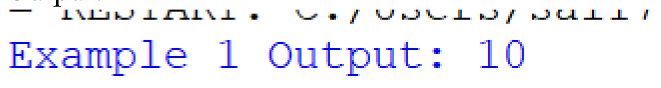
3) 0-indexed integer array nums

Program:



```python
def sum_of_squares_distinct_counts(nums):
    n = len(nums)
    count_dict = {}
    total_sum = 0
    left, right = 0, 0

    while right < n:
        count_dict[nums[right]] = count_dict.get(nums[right], 0) + 1
        total_sum += sum(count ** 2 for count in count_dict.values())
        while left < right and count_dict[nums[left]] > 1:
            count_dict[nums[left]] -= 1
            total_sum += sum(count ** 2 for count in count_dict.values())
            left += 1

        right += 1

    return total_sum
nums_example1 = [1, 2, 1]
output_example1 = sum_of_squares_distinct_counts(nums_example1)
print(f"Example 1 Output: {output_example1}")
```

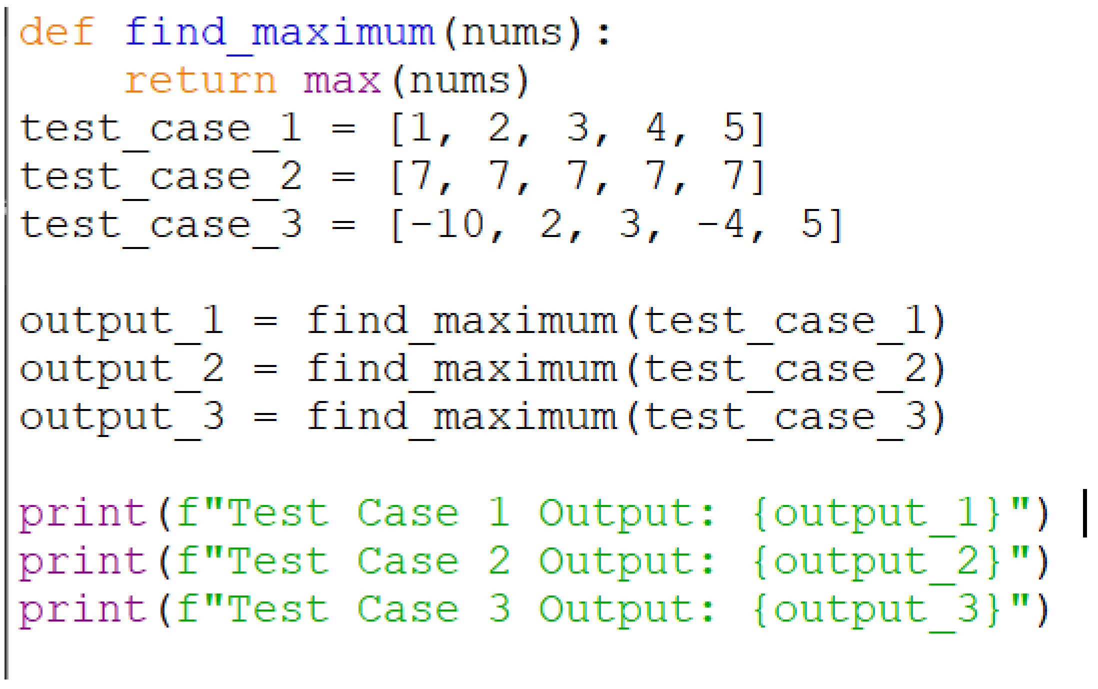
Output:



```
Example 1 Output: 10
```

4)find the time complexity of the input program:

Program:



```python
def find_maximum(nums):
    return max(nums)
test_case_1 = [1, 2, 3, 4, 5]
test_case_2 = [7, 7, 7, 7, 7]
test_case_3 = [-10, 2, 3, -4, 5]

output_1 = find_maximum(test_case_1)
output_2 = find_maximum(test_case_2)
output_3 = find_maximum(test_case_3)

print(f"Test Case 1 Output: {output_1}")
print(f"Test Case 2 Output: {output_2}")
print(f"Test Case 3 Output: {output_3}")
```

Output:

```
Test Case 1 Output: 5
Test Case 2 Output: 7
Test Case 3 Output: 5
```

5) 0-indexed integer array to find k value.

Program:

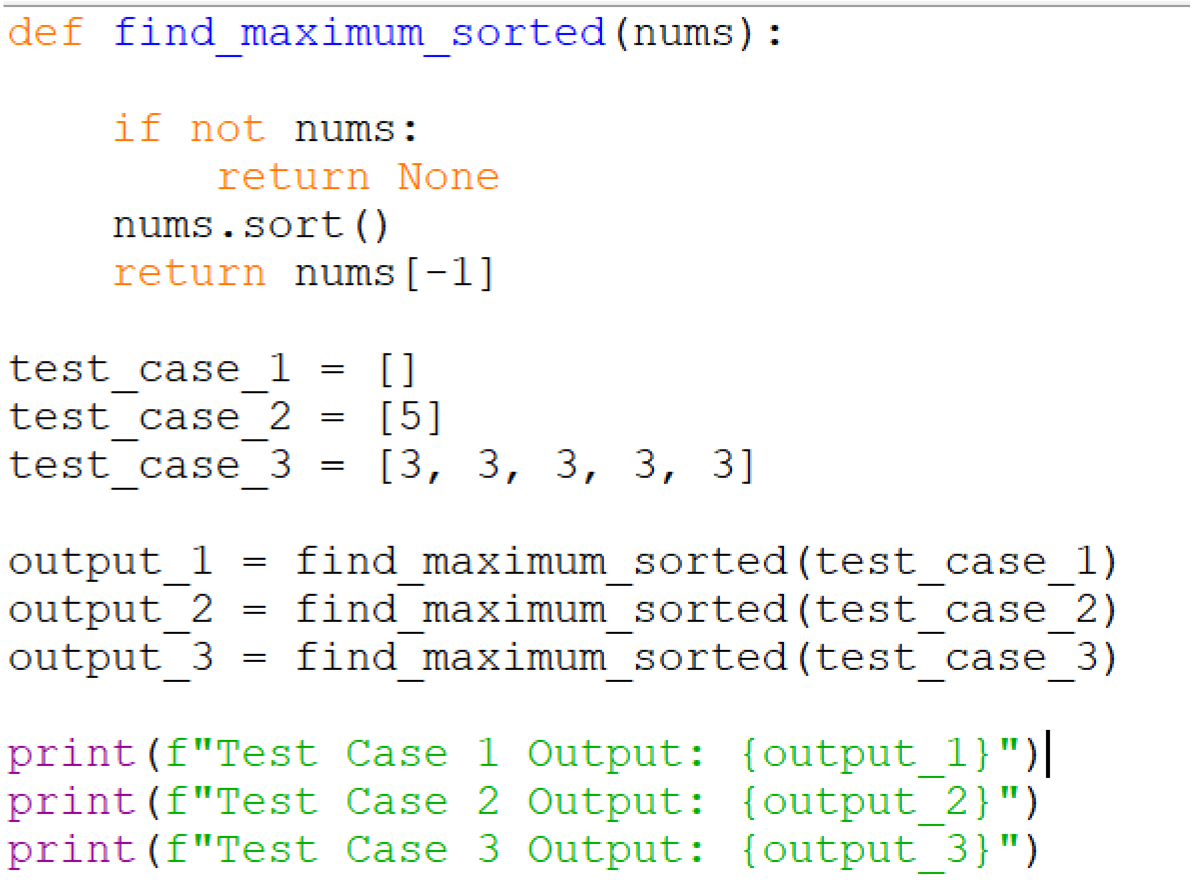
```
def count_pairs(nums, k):
    count_dict = {}
    result = 0

    for i, num in enumerate(nums):
        remainder = num % k
        count_dict[remainder] = count_dict.get(remainder, 0) + 1
        result += count_dict[remainder] * count_dict[(k - remainder) % k]

    return result
nums_example1 = [3, 1, 2, 2, 2, 1, 3]
k_example1 = 2
output_example1 = count_pairs(nums_example1, k_example1)
print(f"Example 1 Output: {output_example1}")
```

Output:

```
Example 1 Output: 44
```

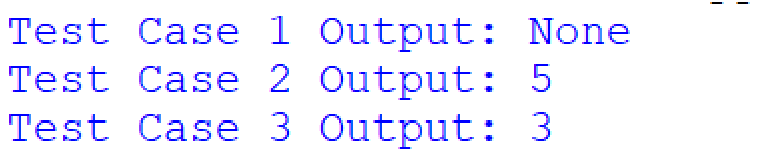
6)maximum element from the list:

Program:



```python
def find_maximum_sorted(nums):

    if not nums:
        return None
    nums.sort()
    return nums[-1]

test_case_1 = []
test_case_2 = [5]
test_case_3 = [3, 3, 3, 3, 3]

output_1 = find_maximum_sorted(test_case_1)
output_2 = find_maximum_sorted(test_case_2)
output_3 = find_maximum_sorted(test_case_3)

print(f"Test Case 1 Output: {output_1}")
print(f"Test Case 2 Output: {output_2}")
print(f"Test Case 3 Output: {output_3}")
```

Output:



```
Test Case 1 Output: None
Test Case 2 Output: 5
Test Case 3 Output: 3
```

7) using space complexity algorithm:

Program:

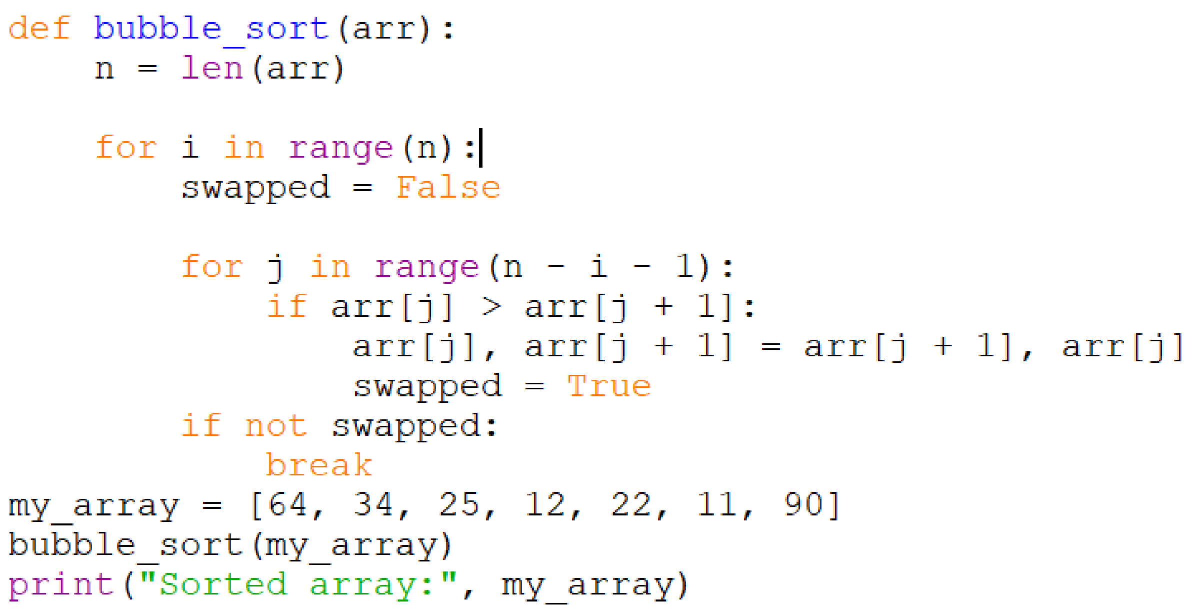
```python
def find_unique_elements(nums):
    unique_set = set()

    for num in nums:
        unique_set.add(num)
    unique_list = list(unique_set)

    return unique_list
test_case_1 = [3, 7, 3, 5, 2, 5, 9, 2]
test_case_2 = [-1, 2, -1, 3, 2, -2]
test_case_3 = [1000000, 999999, 1000000]

output_1 = find_unique_elements(test_case_1)
output_2 = find_unique_elements(test_case_2)
output_3 = find_unique_elements(test_case_3)

print(f"Test Case 1 Output: {output_1}")
print(f"Test Case 2 Output: {output_2}")
print(f"Test Case 3 Output: {output_3}")
```

Output:

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

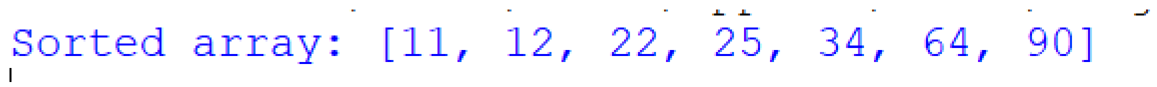
8)BUBBLE SORT:

PROGRAM:



```
def bubble_sort(arr):
    n = len(arr)

    for i in range(n):
        swapped = False

        for j in range(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
my_array = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(my_array)
print("Sorted array:", my_array)
```

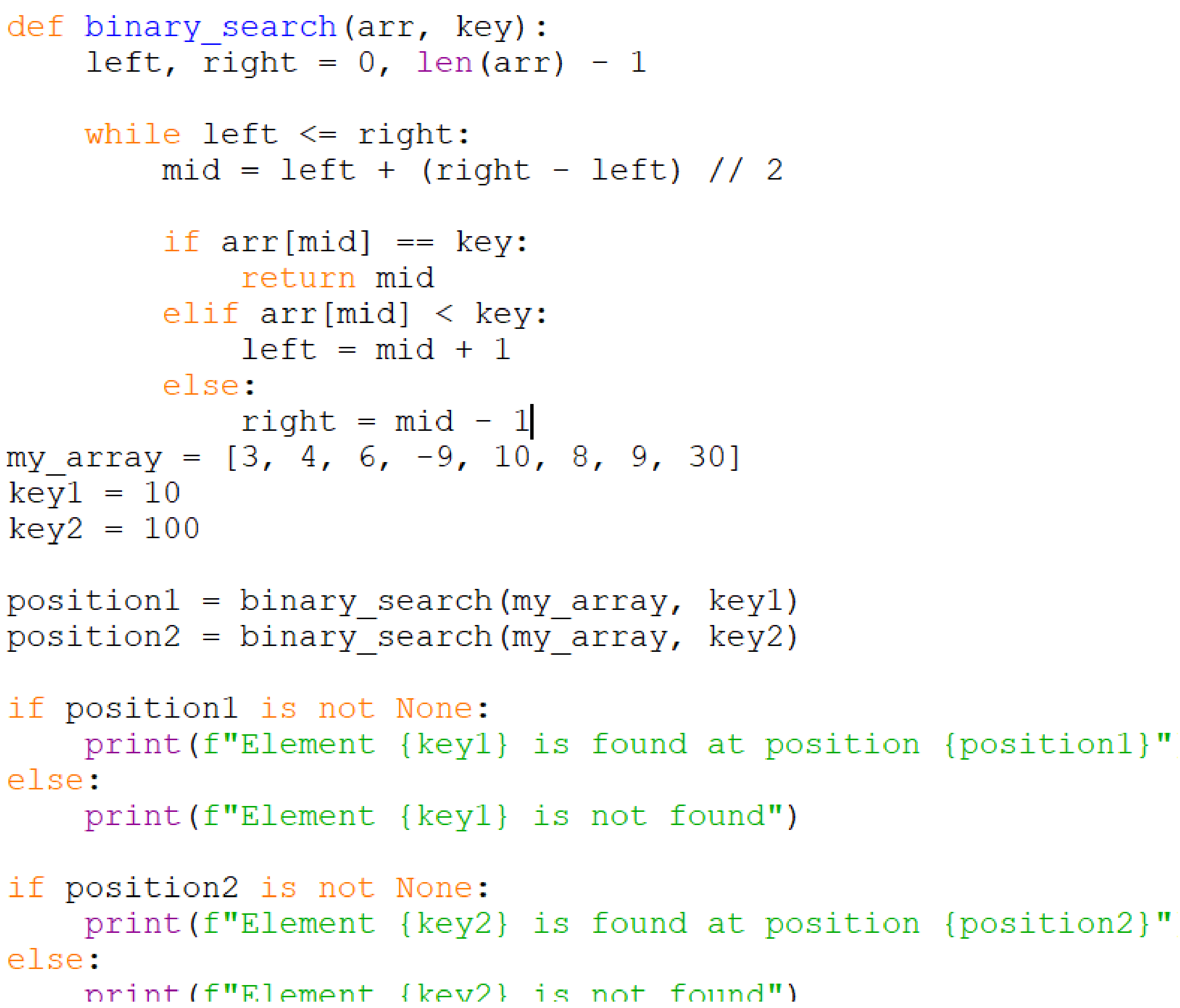
OUTPUT:



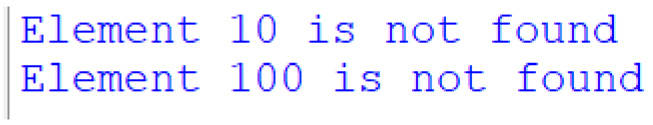
```
Sorted array: [11, 12, 22, 25, 34, 64, 90]
```

9)BINARY SEARCH TO FIND BIG-O NOTATION:

PROGRAM:


```python
def binary_search(arr, key):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = left + (right - left) // 2

        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
my_array = [3, 4, 6, -9, 10, 8, 9, 30]
key1 = 10
key2 = 100

position1 = binary_search(my_array, key1)
position2 = binary_search(my_array, key2)

if position1 is not None:
    print(f"Element {key1} is found at position {position1}"
else:
    print(f"Element {key1} is not found")

if position2 is not None:
    print(f"Element {key2} is found at position {position2}"
else:
    print(f"Element {key2} is not found")
```

OUTPUT:


```
Element 10 is not found
Element 100 is not found
```

10) solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.
PROGRAM:

```
def merge_sort(nums):
    if len(nums) <= 1:
        return nums
    mid = len(nums) // 2
    left_half = nums[:mid]
    right_half = nums[mid:]

    left_sorted = merge_sort(left_half)
    right_sorted = merge_sort(right_half)

    return merge(left_sorted, right_sorted)

def merge(left, right):
    result = []
    left_ptr, right_ptr = 0, 0

    while left_ptr < len(left) and right_ptr < len(right):
        if left[left_ptr] < right[right_ptr]:
            result.append(left[left_ptr])
            left_ptr += 1
        else:
            result.append(right[right_ptr])
            right_ptr += 1
    result.extend(left[left_ptr:])
    result.extend(right[right_ptr:])

    return result
nums = [3, 1, 4, 2, 5]
sorted_nums = merge_sort(nums)
print("Sorted array:", sorted_nums)
```
Output:

```
Sorted array: [1, 2, 3, 4, 5]
```

11) an m x n grid and a ball at a starting cell.

Program:

```
def find_paths(m, n, N, i, j):
    MOD = 10**9 + 7
    dp = [[[0] * n for _ in range(m)] for _ in range(N + 1)]
    dp[0][i][j] = 1
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    for step in range(1, N + 1):
        for x in range(m):
            for y in range(n):
                for dx, dy in directions:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < m and 0 <= ny < n:
                        dp[step][x][y] = (dp[step][x][y] + dp[step - 1][nx][ny])

    total_paths = 0
    for x in range(m):
        for y in range(n):
            if x == 0 or x == m - 1 or y == 0 or y == n - 1:
                total_paths = (total_paths + dp[N][x][y]) % MOD

    return total_paths
m1, n1, N1, i1, j1 = 2, 2, 2, 0, 0
output1 = find_paths(m1, n1, N1, i1, j1)
print(f"Example 1 Output: {output1}")

m2, n2, N2, i2, j2 = 1, 3, 3, 0, 1
output2 = find_paths(m2, n2, N2, i2, j2)
print(f"Example 2 Output: {output2}")
```

Output:

```
---------------------------------
Example 1 Output: 4
Example 2 Output: 4
```

12) two adjacent houses

Program:

```
def rob(nums):
    def helper(segment):
        if not segment:
            return 0
        if len(segment) == 1:
            return segment[0]
        if len(segment) == 2:
            return max(segment[0], segment[1])

        dp = [segment[0], max(segment[0], segment[1])]
        for i in range(2, len(segment)):
            dp.append(max(segment[i] + dp[i - 2], dp[i - 1]))

        return dp[-1]

    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]
    return max(helper(nums[:-1]), helper(nums[1:]))
nums1 = [2, 3, 2]
output1 = rob(nums1)
print(f"Example 1 Output: {output1}")
nums2 = [1, 2, 3, 1]
output2 = rob(nums2)
print(f"Example 2 Output: {output2}")
```
Output:

```
Example 1 Output: 3
Example 2 Output: 4
```

13) climbing a staircase.

Program:

```
def climb_stairs(n):
    if n <= 1:
        return 1
    prev, curr = 1, 1
    for _ in range(2, n + 1):
        prev, curr = curr, prev + curr

    return curr
n1 = 4
output1 = climb_stairs(n1)
print(f"Example 1 Output: {output1}")

n2 = 3
output2 = climb_stairs(n2)
print(f"Example 2 Output: {output2}")
```

Output:

```
Example 1 Output: 5
Example 2 Output: 3
```

14) find the unique path of the robot.

Program:

```
def unique_paths(m, n):
    dp = [[1] * n for _ in range(m)]
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1]
    return dp[m - 1][n - 1]
m1, n1 = 7, 3
output1 = unique_paths(m1, n1)
print(f"Example 1 Output: {output1}")
m2, n2 = 3, 2
output2 = unique_paths(m2, n2)
print(f"Example 2 Output: {output2}")
```

Output:

```
Example 1 Output: 28
Example 2 Output: 3
```

15)find the largest group from the string

Program:

```
def large_group_intervals(S):
    result = []
    n = len(S)
    start = 0

    for end in range(1, n):
        if S[end] != S[start]:
            if end - start >= 3:
                result.append([start, end - 1])
            start = end
    if n - start >= 3:
        result.append([start, n - 1])

    return result
s1 = "abbxxxxzzy"
output1 = large_group_intervals(s1)
print(f"Example 1 Output: {output1}")

s2 = "abc"
output2 = large_group_intervals(s2)
print(f"Example 2 Output: {output2}")
```

Output:

```
Example 1 Output: [[3, 6]]
Example 2 Output: []
```

16) m x n grid broad

Program:

```python
def game_of_life(board):
    m, n = len(board), len(board[0])
    next_state = [[0] * n for _ in range(m)]
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (-1, -1), (1, -1),

    for i in range(m):
        for j in range(n):
            live_neighbors = 0

            for dx, dy in directions:
                ni, nj = i + dx, j + dy
                if 0 <= ni < m and 0 <= nj < n and board[ni][nj] == 1:
                    live_neighbors += 1

            if board[i][j] == 1:
                if live_neighbors < 2 or live_neighbors > 3:
                    next_state[i][j] = 0
                else:
                    next_state[i][j] = 1
            else:
                if live_neighbors == 3:
                    next_state[i][j] = 1
    return next_state
board1 = [[0, 1, 0], [0, 0, 1], [1, 1, 1], [0, 0, 0]]
output1 = game_of_life(board1)
print(f"Example 1 Output: {output1}")
board2 = [[1, 1], [1, 0]]
output2 = game_of_life(board2)
print(f"Example 2 Output: {output2}")
```

Output:

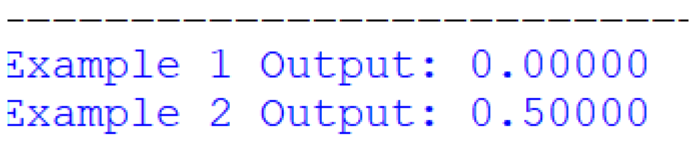
```
Example 1 Output: [[0, 0, 0], [1, 0, 1], [0, 1, 1], [0, 1, 0]]
Example 2 Output: [[1, 1], [1, 1]]
```

17)pyramid glasses.

Program:

```python
def champagne_tower(poured, query_row, query_glass):
    tower = [[0.0] * (i + 1) for i in range(query_row + 1)]
    tower[0][0] = poured

    for row in range(query_row):
        for glass in range(row + 1):
            excess = max(0, (tower[row][glass] - 1) / 2)
            tower[row + 1][glass] += excess
            tower[row + 1][glass + 1] += excess

    return min(1.0, tower[query_row][query_glass])
output1 = champagne_tower(1, 1, 1)
print(f"Example 1 Output: {output1:.5f}")
output2 = champagne_tower(2, 1, 1)
print(f"Example 2 Output: {output2:.5f}")
```

Output:



```
---------------------------
Example 1 Output: 0.00000
Example 2 Output: 0.50000
```