Problem 1

```
python
Copy code
def first_palindromic_string(words):
    for word in words:
        if word == word[::-1]:
            return word
    return ""

# Example usage:
print(first_palindromic_string(["abc", "car", "ada", "racecar", "cool"]))
# Output: "ada"
print(first_palindromic_string(["notapalindrome", "racecar"]))  # Output:
"racecar"
```

Problem 2

```
python
Copy code
def common_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]

# Example usage:
print(common_indices([2, 3, 2], [1, 2]))  # Output: [2, 1]
print(common_indices([4, 3, 2, 3, 1], [2, 2, 5, 2, 3, 6]))  # Output: [3,
4]
```

Problem 3

```
python
Copy code
def sum_of_squares(nums):
    n = len(nums)
    result = 0
    for i in range(n):
        distinct_count = len(set())
        for j in range(i, n):
            distinct_count.add(nums[j])
            result += len(distinct_count) ** 2
    return result

# Example usage:
print(sum_of_squares([1, 2, 1]))  # Output: 15
print(sum_of_squares([1, 1]))  # Output: 3
```

Problem 4

```
python
Copy code
def count_pairs(nums, k):
    n = len(nums)
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            if nums[i] == nums[j] and (i * j) % k == 0:
                count += 1
    return count

# Example usage:
print(count_pairs([3, 1, 2, 2, 2, 1, 3], 2))  # Output: 4
print(count_pairs([1, 2, 3, 4], 1))  # Output: 0
```

Problem 5

```
python
Copy code
def find_max(nums):
    if not nums:
        return None
    return max(nums)

# Example usage:
print(find_max([1, 2, 3, 4, 5]))  # Output: 5
print(find_max([7, 7, 7, 7, 7]))  # Output: 7
print(find_max([-10, 2, 3, -4, 5]))  # Output: 5
```

Problem 6

```
python
Copy code
def sort_and_find_max(nums):
    if not nums:
        return "The list is empty."
    nums.sort()
    return nums[-1]

# Example usage:
print(sort_and_find_max([]))  # Output: The list is empty.
print(sort_and_find_max([5]))  # Output: 5
print(sort_and_find_max([3, 3, 3, 3, 3]))  # Output: 3
```

Problem 7

```
python
Copy code
def unique_elements(nums):
    return list(set(nums))

# Example usage:
print(unique_elements([3, 7, 3, 5, 2, 5, 9, 2]))  # Output: [2, 3, 5, 7, 9]
(Order may vary)
print(unique_elements([-1, 2, -1, 3, 2, -2]))  # Output: [-1, 2, 3, -2]
(Order may vary)
print(unique_elements([1000000, 999999, 1000000]))  # Output: [1000000,
999999]
```

Space Complexity: O(n), where n is the number of elements in the input list.

Problem 8

```
python
Copy code
def bubble_sort(nums):
    n = len(nums)
    for i in range(n):
        for j in range(0, n - i - 1):
            if nums[j] > nums[j + 1]:
                nums[j], nums[j + 1] = nums[j + 1], nums[j]
    return nums

# Example usage:
print(bubble_sort([3, 2, 1, 4, 5]))  # Output: [1, 2, 3, 4, 5]
```

Time Complexity: O(n^2) where n is the number of elements in the array.

Problem 9

```
python
Copy code
def binary_search(arr, key):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == key:
            return f"Element {key} is found at position {mid}"
        elif arr[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    return f"Element {key} is not found"

# Example usage:
print(binary_search([3, 4, 6, -9, 10, 8, 9, 30], 10))  # Output: Element 10
is found at position 5
print(binary_search([3, 4, 6, -9, 10, 8, 9, 30], 100))  # Output: Element
100 is not found
```

Time Complexity: O(log n) where n is the number of elements in the array.

Problem 10

```
python
Copy code
def quick_sort(nums):
    if len(nums) <= 1:
        return nums
    pivot = nums[len(nums) // 2]
    left = [x for x in nums if x < pivot]
    middle = [x for x in nums if x == pivot]
    right = [x for x in nums if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# Example usage:
print(quick_sort([3, 6, 8, 10, 1, 2, 1]))  # Output: [1, 1, 2, 3, 6, 8, 10]
```

Time Complexity: O(n log n)

Problem 11

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

    for step in range(1, N + 1):
        for r in range(m):
            for c in range(n):
                for dr, dc in ((0, 1), (1, 0), (0, -1), (-1, 0)):
                    nr, nc = r + dr, c + dc
                    if 0 <= nr < m and 0 <= nc < n:
                        dp[step][r][c] += dp[step - 1][nr][nc]
                    else:
                        dp[step][r][c] += 1
                    dp[step][r][c] %= MOD
    return sum(dp[N][r][c] for r in range(m) for c in range(n)) % MOD

# Example usage:
print(find_paths(2, 2, 2, 0, 0))  # Output: 6
print(find_paths(1, 3, 3, 0, 1))  # Output: 12
```

Problem 12

```
python
Copy code
def rob(nums):
    if len(nums) == 1:
        return nums[0]
    def rob_linear(houses):
        rob1, rob2 = 0, 0
        for n in houses:
            new_rob = max(rob1 + n, rob2)
            rob1 = rob2
            rob2 = new_rob
        return rob2

    return max(rob_linear(nums[1:]), rob_linear(nums[:-1]))

# Example usage:
print(rob([2, 3, 2]))  # Output: 3
print(rob([1, 2, 3, 1]))  # Output: 4
```

Problem 13

```
python
Copy code
def climb_stairs(n):
    if n <= 2:
        return n
    a, b = 1, 2
    for _ in range(3, n + 1):
        a, b = b, a + b
    return b

# Example usage:
```

```
print(climb_stairs(4))  # Output: 5
print(climb_stairs(3))  # Output: 3
```

Problem 14

```
python
Copy code
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[-1][-1]

# Example usage:
print(unique_paths(7, 3))  # Output: 28
print(unique_paths(3, 2))  # Output: 3
```

Problem 15

```
python
Copy code
def large_group_positions(S):
    res = []
    i = 0
    for j in range(len(S)):
        if j == len(S) - 1 or S[j] != S[j + 1]:
            if j - i + 1 >= 3:
                res.append([i, j])
            i = j + 1
    return res

# Example usage:
print(large_group_positions("abbxxxxzzy"))  # Output: [[3,6]]
print(large_group_positions("abc"))  # Output: []
```

Problem 16

```
python
Copy code
def game_of_life(board):
    def count_live_neighbors(r, c):
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1),
(1, -1), (1, 1)]
        count = 0
        for dr, dc in directions:
            nr, nc = r + dr, c + dc
            if 0 <= nr < len(board) and 0 <= nc < len(board[0]) and
abs(board[nr][nc]) == 1:
                count += 1
        return count

    for r in range(len(board)):
        for c in range(len(board[0])):
            live_neighbors = count_live_neighbors(r, c)
            if board[r][c] == 1 and (live_neighbors < 2 or live_neighbors >
3):
                board[r][c] = -1
            if board[r][c] == 0 and live_neighbors == 3:
                board[r][c] = 2

    for r in range(len(board)):
        for c in range(len(board[0])):
            if board[r][c] > 0:
                board[r][c] = 1
            else:
                board[r][c] = 0

# Example usage:
board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
game_of_life(board)
print(board)  # Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]
```

Problem 17

```
python
Copy code
def champagne_tower(poured, query_row, query_glass):
    dp = [[0] * (i + 1) for i in range(101)]
    dp[0][0] = poured
    for r in range(query_row + 1):
        for c in range(r + 1):
            if dp[r][c] > 1:
                excess = (dp[r][c] - 1) / 2.0
                dp[r][c] = 1
                dp[r + 1][c] += excess
                dp[r + 1][c + 1] += excess
    return min(1, dp[query_row][query_glass])

# Example usage:
print(champagne_tower(1, 1, 1))  # Output: 0.00000
print(champagne_tower(2, 1, 1))
```