Problem 1 :

def quicksort(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 quicksort(left) + middle + quicksort(right)

Ans===
Best/Average Case: O (n log n) O(nlogn)

Worst Case: O (n 2) O(n 2)

Space Complexity: O (n) O(n) worst case, O (log n) O(logn) best case.

Problem 2 :

def nested_loop_example(matrix): rows, cols = len(matrix), len(matrix[0]) total = 0 for i in range(rows): for j in range(cols): total += matrix[i][j] return total

Ans=== time complexity is O(n × m)

The space complexity is O(1)

Problem 3 :

def example_function(arr): result = 0 for element in arr: result += element return result

Ans== Time Complexity: O(n)

Space Complexity: O(1)

Problem 4 :

def longest_increasing_subsequence(nums): n = len(nums) lis = [1] * n for i in range(1, n): for j in range(0, i): if nums[i] > nums[j] and lis[i] < lis[j] + 1: lis[i] = lis[j] + 1 return max(lis)

Ans== Time Complexity: O(n²)

Space Complexity: O(n)

Problem 5 :

def mysterious_function(arr): n = len(arr) result = 0 for i in range(n): for j in range(i, n): result += arr[i] * arr[j] return result

ANs== Time Complexity: O(n²)

Space Complexity: O(1)

Problem 6 : Sum of Digits

Write a recursive function to calculate the sum of digits of a given positive integer.

sum_of_digits(123) -> 6

Ans== Time Complexity: O(d)

Space Complexity: O(d)

```
def sum_of_digits(n):
    if n == 0:
        return 0
    return (n % 10) + sum_of_digits(n // 10)

# Example usage:
print(sum_of_digits(123))  # Output: 6
```

```
6
```

Problem 7: Fibonacci Series

Write a recursive function to generate the first n numbers of the Fibonacci series.

fibonacci_series(6) -> [0, 1, 1, 2, 3, 5]

Ans== Time Complexity: O(n)

Space Complexity: O(n)

```
def fibonacci_series(n, series=[0, 1]):
    if len(series) >= n:
        return series[:n]
    series.append(series[-1] + series[-2])
    return fibonacci_series(n, series)

# Example usage:
print(fibonacci_series(6))  # Output: [0, 1, 1, 2, 3, 5]
```

```
[0, 1, 1, 2, 3, 5]
```

Problem 8 : Subset Sum

Given a set of positive integers and a target sum, write a recursive function to determine if there exists a subset of the integers that adds up to the target sum.

subset_sum([3, 34, 4, 12, 5, 2], 9) -> True

ANs== Time Complexity: $O(2^n)$

Space Complexity: O(n)

```
def subset_sum(nums, target, index=0):
    if target == 0:
        return True
    if index >= len(nums) or target < 0:
        return False

    # Include the current number and check
    include = subset_sum(nums, target - nums[index], index + 1)

    # Exclude the current number and check
    exclude = subset_sum(nums, target, index + 1)

    return include or exclude

# Example usage:
print(subset_sum([3, 34, 4, 12, 5, 2], 9))  # Output: True
```

```
True
```

Problem 9: Word Break

Given a non-empty string and a dictionary of words, write a recursive function to determine if the string can be segmented into a space-separated sequence of dictionary words.

word_break(leetcode , [leet , code]) -> True

Ans== Time Complexity: $O(2^n)$

Space Complexity: O(n)

```
def word_break(s, word_dict):
    if not s:
        return True  # If the string is empty, it can be segmented

    for word in word_dict:
        if s.startswith(word):  # Check if the string starts with the dictionary word
            if word_break(s[len(word):], word_dict):  # Recur on the remaining substring
                return True

    return False  # No valid segmentation found

# Example usage:
print(word_break("leetcode", ["leet", "code"]))  # Output: True
print(word_break("applepenapple", ["apple", "pen"]))  # Output: True
print(word_break("catsandog", ["cats", "dog", "sand", "and", "cat"]))  # Output: False
```

```
True
True
False
```

Problem 10 : N-Queens

Implement a recursive function to solve the N Queens problem, where you have to place N queens on an N×N chessboard in such a way that no two queens threaten each other.

n_queens(4)

[[".Q..", "...Q", "Q...", "..Q."], ["..Q.", "Q...", "...Q", ".Q.."]]

```
def solve_n_queens(n):
    def is_safe(board, row, col):
        for i in range(row):
            if board[i] == col or \
               board[i] - i == col - row or \
               board[i] + i == col + row:
                return False
        return True

    def solve(board, row, solutions):
        if row == n:
            solutions.append(["." * c + "Q" + "." * (n - c - 1) for c in board])
            return
        for col in range(n):
            if is_safe(board, row, col):
                board[row] = col
                solve(board, row + 1, solutions)
                board[row] = -1

    solutions = []
    solve([-1] * n, 0, solutions)
    return solutions

# Example usage:
n = 4
result = solve_n_queens(n)
for solution in result:
    for row in solution:
        print(row)
    print()
```

```
.Q..
...Q
Q...
..Q.

..Q.
Q...
...Q
.Q..
```