Time Complexity and Recursion Assignment Find time complexity of below code blocks : 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) Time complexity for average case is o(nlog n),worst case o(n^2) Explanation each recursive call processes n elemnts to partition the array into left,middle,right, in avg case the depth of the recursion tree is log(n)

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

Time complexity for o(rows*cols) Explanation every element of the matrix is visited onces.

Problem 3 : def example_function(arr): result = 0 for element in arr: result += element return result Time complexity :o(n) Explanation array of n element pass onces.

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)

Time complexity:o(n^2) Explanation:Foer every i,we check all j<i leading to nested loop

Practice Question - Creating Classes 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 Time complexity: o(n^2) Explanation:The nested loop go from i to n ,forming a triangular iteration over the array.

Solve the following problems on recursion 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

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]

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 Solve the following problems on recursion

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

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.."]]

```python
#6.sum of digits
def sum_digits(n):
    if n==0:
        return 0
    return n%10 + sum_digits(n//10)

print(sum_digits(123))
```

```
6
```

```python
#7. fibonnaci series
def  fibonnaci_series(n): #native exponential time -o(2^n)
    def fib(i):
        if i<=1:
            return i
        return fib(i-1) + fib(i-2)
    return [fib(i) for i in range(n)]
print(fibonnaci_series(6))
```

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

```python
def fibonacci_series(n):#Memoized Recursive Fibonacci (Optimized)
    memo = {}

    def fib(i):
        if i in memo:
            return memo[i]
        if i <= 1:
            memo[i] = i
        else:
            memo[i] = fib(i - 1) + fib(i - 2)
        return memo[i]

    return [fib(i) for i in range(n)]

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

```
6
```

```python
#8.Subset Sum
def Subset_Sum(arr,target):
    def helper(i,target):
        if target==0:
            return True
        if i == len(arr):
            return False
        if arr[i]<=target:
            if helper(i+1, target -arr[i]):
                return True
        return helper(i+1,target)
    return helper(0,target)
print(Subset_Sum([3, 34, 4, 12, 5, 2], 9))
```

```
True
```

```python
#9.Word Break
def word_break(s,word_dict):
    def helper(start):
        if start ==len(s):
            return True
        for end in range(start+1,len(s)+1):
```

```
            if s[start:end] in word_dict and helper(end):
                return True
        return False
    return helper(0)
print(word_break("leetcode",["leet","code"]))
print(word_break("leet",["le","et"]))
```

```
True
True
```

```
#10.
def n_queens(n):
    res = []

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

    solve(0, [0] * n)
    return res

# Example
from pprint import pprint
pprint(n_queens(4))
```

```
[['.Q..', '...Q', 'Q...', '..Q.'], ['..Q.', 'Q...', '...Q', '.Q..']]
```