Time Complexity + 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)
```

Average-case time Complexity : O(n log n)

Worst-case time complexity: O(n^2)

where n is the length of the array

Problem 2 :

def nested_loop_example(matrix):

```
    rows, cols = len(matrix), len(matrix[0])

    total = 0
```

Time complexity : O(rows * cols), where rows is number of rows and cols is number of columns in the matrix

Problem 3 :

def example_function(arr):

```
    result = 0

    for element in arr:

        result += element

    return result
```

Time complexity : O(n), where n is the length of the array

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), where n is the length of the input array nums

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), where n is the length of the input 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

In [8]:
```
def sum_of_digits(num):
    if num <= 9:
        return num
    else:
        digit = num % 10
        return digit + sum_of_digits(num // 10)

## Example usage
print(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]

In [12]:
```python
def fibobacci_series(n):
    if n<= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0,1]
    else:
        fibo = fibobacci_series(n-1)
        fibo.append(fibo[-1] + fibo[-2])
        return fibo

##  Example usage
print(fibobacci_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

In [40]:
```python
def subset_sum(nums, target):
    # Base case: if the target is 0, an empty subset is valid
    if target == 0:
        return True

    # Base case: if the set is empty and the target is not 0, no subset is
    if not nums:
        return False

    # Recursive case: explore two possibilities - include the current eleme
    include_current = subset_sum(nums[1:], target - nums[0])
    exclude_current = subset_sum(nums[1:], target)

    # Return True if either of the possibilities leads to a valid subset
    return include_current or exclude_current

# Example usage:
nums = [3, 34, 4, 12, 5, 2]
target_sum = 9
result = subset_sum(nums, target_sum)
print(result)
```

```
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

In [38]:
```
def word_break(wordlist,word):
    if word == "":
        return True
    else:
        wordlen = len(word)
        for i in range(1,wordlen+1):
            if word[:i] in wordlist and word_break(wordlist,word[i:]):
                return True
        return False

#Example usage
print(word_break(["leet","code"],"leetcode"))
```

```
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.."]

]

In [32]:
```
# Return True if it's safe to place queen on the board
def isSafeToPlaceQueen(board,row,col,n):

    #check in the left side
    for i in range(col):
        if board[row][i] == 1:
            return False
    #check in the upper left diagonal
    for i,j in zip(range(row,-1,-1),range(col,-1,-1)):
        if board[i][j] == 1:
            return False
    #check in the lower left diagonal
    for i,j in zip(range(row,n,1),range(col,-1,-1)):
        if board[i][j] == 1:
            return False
    return True


## should return true if we are able to place all the queens
def solveNQUtil(board,col,n):
    #Base condition
    if (col>=n):
        return True #means we have been able to put queens in all the colum
    #check for all the rows
    for row in range(n):
        if isSafeToPlaceQueen(board,row,col,n):
            board[row][col] = 1 #set the queen
            #recursively try for the next columns
            if solveNQUtil(board,col+1,n):
                return True
            #Backtracking
            board[row][col] = 0 ## Queen can't be set here

    return False #won't be able to place the queen

def printBoard(board,n):
    for i in range(n):
        for j in range(n):
            if board[i][j] == 1:
                print("Q",end = " ")
            else:
                print(".",end = " ")
        print()



def solveNQ(board,n):

    if not solveNQUtil(board,0,n):
        print("Solution doesn't exist")
        return

    printBoard(board,n)
```

In [33]:
```
import numpy as np
board = np.zeros((4,4),int)
solveNQ(board,4)
```

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