timecomplexity

October 6, 2024

Time Complexity + Recursion

[7]:
```python
# 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)


arr = [3, 6, 8, 10, 1, 2, 1]
sorted_arr = quicksort(arr)
print(sorted_arr)
```

```
[1, 1, 2, 3, 6, 8, 10]
```

[9]:
```python
# 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
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
```

```
]
result = nested_loop_example(matrix)
print(result)
```

45

[11]:
```
# Problem 3 :
def example_function(arr):
    result = 0

    for element in arr:
        result += element

    return result
arr = [1, 2, 3, 4, 5]
total = example_function(arr)
print(total)
```

15

[13]:
```
# 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)
nums = [10, 22, 9, 33, 21, 50, 41, 60, 80]
length = longest_increasing_subsequence(nums)
print(length)
```

6

[15]:
```
# 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
arr = [1, 2, 3]
output = mysterious_function(arr)
```

```
print(output)
```

25

Solve the following problems on recursion

[17]:
```
# 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
def sum_of_digits(n):
    # Base case
    if n == 0:
        return 0
    else:
        # Recursive case
        return n % 10 + sum_of_digits(n // 10)

# Example usage
result = sum_of_digits(123)
print(result)
```

6

[19]:
```
#  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]
def fibonacci_series(n):
    # Base cases
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]


    series = fibonacci_series(n - 1)

    next_fib = series[-1] + series[-2]
    series.append(next_fib)
    return series


result = fibonacci_series(6)
print(result)
```

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

[21]:
```
# 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
def subset_sum(nums, target):

    if target == 0:
        return True

    if not nums:
        return False


    last_element = nums[-1]


    if last_element <= target:

        return subset_sum(nums[:-1], target - last_element) or subset_sum(nums[:
 ↪-1], target)


    return subset_sum(nums[:-1], target)


result = subset_sum([3, 34, 4, 12, 5, 2], 9)
print(result)
```

```
True
```

[23]:
```
# 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
def word_break(s, word_dict):

    if not s:
        return True


    for i in range(1, len(s) + 1):
        prefix = s[:i]
        if prefix in word_dict:

            if word_break(s[i:], word_dict):
```

```
                return True


    return False


result = word_break("leetcode", ["leet", "code"])
print(result)
```

True

[25]:
```
# 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 n_queens(n):
    def is_safe(board, row, col):

        for i in range(row):
            if board[i][col] == 'Q':
                return False


        for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
            if board[i][j] == 'Q':
                return False


        for i, j in zip(range(row, -1, -1), range(col, n)):
            if board[i][j] == 'Q':
                return False

        return True

    def solve_n_queens(row, board, solutions):
        if row == n:

            solutions.append([''.join(r) for r in board])
            return

        for col in range(n):
            if is_safe(board, row, col):
                board[row][col] = 'Q'
                solve_n_queens(row + 1, board, solutions)
                board[row][col] = '.'
```

```
    solutions = []
    board = [['.' for _ in range(n)] for _ in range(n)]
    solve_n_queens(0, board, solutions)
    return solutions


result = n_queens(4)
for solution in result:
    print(solution)
```

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

[ ]:
```
#
```

[ ]:
```
#
```