Lab 2

Problem 1

Look at the following function:

```
def countdown(n):

    if n == 0:

        return

    print(n)

    countdown(n - 1)
```

If we call `countdown(3)`, what will be printed? Explain step by step how the function works.

Problem 2

What will be the output of calling `sum_numbers(3)` in the following function?

```
def sum_numbers(n):

    if n == 0:

        return 0

    return n + sum_numbers(n - 1)
```

Show the calculation step by step.

Problem 3

What is the Big-O complexity of the following function?

```
def find_max(arr):

    max_value = arr[0]  # Line 1
    for num in arr:  # Line 2

        if num > max_value:  # Line 3

            max_value = num  # Line 4

    return max_value  # Line 5
```

Write T(n) (the exact count of operations) and then simplify it to find O(n).

Problem 4

Both functions below calculate the sum of numbers from 1 to n. Analyze their time complexity and determine which one is faster.

Code 1: Using a Loop

```
def sum_loop(n):

    total = 0

    for i in range(1, n + 1):

        total += i

    return total
```

Code 2: Using a Formula

```
def sum_formula(n):

    return (n * (n + 1)) // 2
```

1. Determine which function is faster.
2. Explain your reasoning based on time complexity.

Problem 5

Compare the growth strategies of arrays and linked lists when storing large amounts of data. Which one is better for frequent insertions and deletions?

Problem 6

Consider the following queue implementation using a list:

```
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        self.queue.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.queue.pop(0)
        return "Queue is empty"

    def is_empty(self):
        return len(self.queue) == 0
```

What is the time complexity of the `dequeue()` operation?

Problem 7

Consider the following stack implementation using an array:

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        return "Stack is empty"
    def is_empty(self):
        return len(self.stack) == 0
```

1. What happens when we call `pop()` on an empty stack?
2. Modify the `pop()` method to return `None` instead of a string when the stack is empty.

Problem 8

Write a recursive function to compute the product of the first n natural numbers (n!), i.e., factorial of n. Then, analyze its time complexity.

Example Cases:

- `factorial(5) → 120`
- `factorial(3) → 6`
- `factorial(0) → 1`

Problem 9

In a hospital emergency room, patients are generally treated in the order they arrive. However, patients come in with varying degrees of severity. Patients with critical conditions (urgent) must be treated before those with less severe conditions (regular). You need to design a system that always processes patients with higher severity first. Which data structure should you use?

Problem 10

The newest huge maths company, South Pacific Computation Corporation (SPPC), is here! SPPC has shares that they need to distribute to their employees. Assume that you are the CEO for SPPC, and you will receive your shares over days. On day one, you receive *n* shares. On day two, you receive *n/2* shares (rounded down). On day three, you receive *n/3* shares (rounded down). One day *i*, you receive *n/i* shares (rounded down). On the final day (day n), you receive *n/n = 1* share. For example, if *n* = 3, then you would receive 3+1+1 = 5 shares.

How many shares in total do you receive?

Input

The input consists of a single line containing one integer ***n (1 <= n <= 10^12)***, which is the number of days over which you receive your shares.

Output

Display the number of shares in total that you receive.

Sample Input	Output
1	1
3	5
4	8
5	10
10	27