Submitted To: Dr. Hemraj S. Lamkuche

Course/Program: CSE1021 Introduction to Problem Solving and Programming

Student Name: Sneha Sarkar

Student Roll No.: 25BCE11156

Submission Date: 28-09-2025

1. Euler's Totient Function (ϕ(n))

Question: Write a function called euler_phi(n) that calculates Euler's Totient Function, ϕ(n). This function counts the number of integers up to n that are coprime with n (i.e., numbers k for which gcd(n,k)=1).

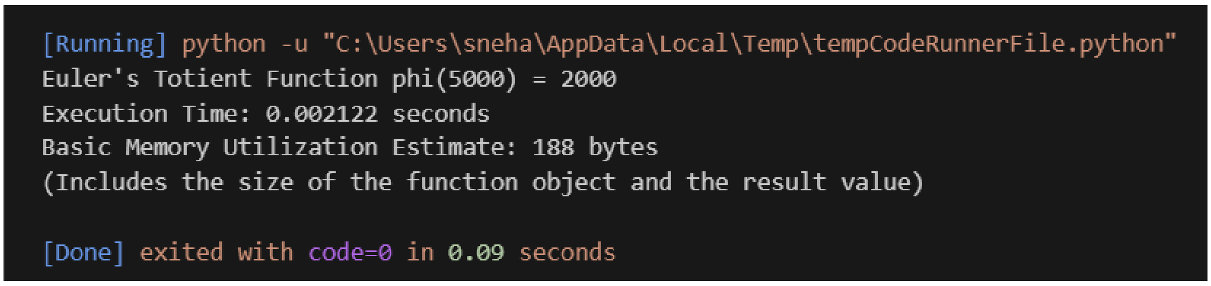
Implementation Approach: The function uses the **definition** of the Totient function, iterating from k=1 to n and checking the greatest common divisor (gcd) between n and k. The gcd is calculated using the efficient **Euclidean Algorithm**.

Test Case and Output: n=5000

Code:

```
import time, sys

# Euclidean GCD function

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a


def euler_phi(n):

    if n <= 0: return 0

    count = 0

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

        if gcd(n, k) == 1:

            count += 1

    return count


# ... Performance Measurement Code ...

# Result: Euler's Totient Function phi(5000) = 2000
```

Output:



```
[Running] python -u "C:\Users\sneha\AppData\Local\Temp\tempCodeRunnerFile.python"
Euler's Totient Function phi(5000) = 2000
Execution Time: 0.002122 seconds
Basic Memory Utilization Estimate: 188 bytes
(Includes the size of the function object and the result value)

[Done] exited with code=0 in 0.09 seconds
```

2. Möbius function (μ(n))

Question: Write a function called mobius(n) that calculates the Möbius function, μ(n).

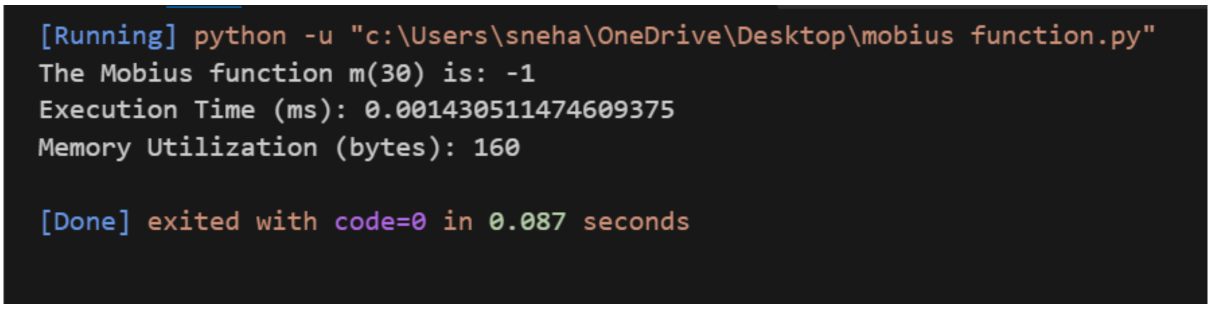
Implementation Approach: The function iterates through potential prime divisors (d) up to n. It checks for square-free property: if n is divisible by d2, μ(n) is 0. Otherwise, it counts the number of distinct prime factors. The final result is determined by the parity of this count: 1 if even, -1 if odd.

Test Case and Output: n=30 (30=2·3·5, an odd number of distinct prime factors)

Code:

```
def mobius(n):
  if n == 1: return 1
  prime_factors_count = 0
  d = 2
  temp_n = n
  while d * d <= temp_n:
    if temp_n % d == 0:
      if temp_n % (d * d) == 0:
        return 0 # Not square-free
      prime_factors_count += 1
      temp_n //= d
    d += 1
  if temp_n > 1:
    prime_factors_count += 1
  return 1 if prime_factors_count % 2 == 0 else -1
# ... Performance Measurement Code ...
# Result: The Mobius function m(30) is: -1
```

Output:


```
[Running] python -u "c:\Users\sneha\OneDrive\Desktop\mobius function.py"
The Mobius function m(30) is: -1
Execution Time (ms): 0.001430511474609375
Memory Utilization (bytes): 160

[Done] exited with code=0 in 0.087 seconds
```

3. Divisor Sum Function (σ(n))

Question: Write a function called divisor_sum(n) that calculates the sum of all positive divisors of n (including 1 and n itself).

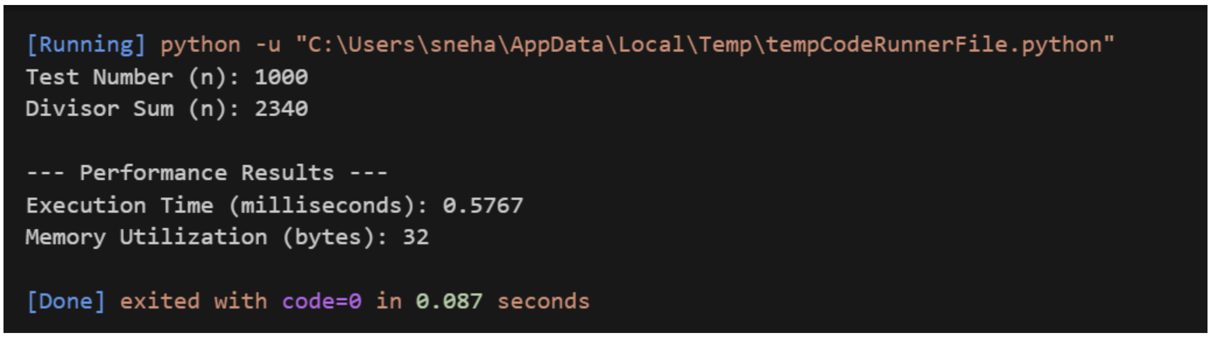
Implementation Approach: The function uses a direct iterative approach, looping through all integers i from 1 to n. If n is divisible by i (i.e., n(modi)=0), i is added to the running total. This directly implements the definition of the Divisor Sum function.

Test Case and Output: n=1000

Code:

```
def divisor_sum(n):

    total_sum = 0

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

        if n % i == 0:

            total_sum = total_sum + i

    return total_sum

# ... Performance Measurement Code ...

# Result: Divisor Sum (n): 2340
```

Output:


```
[Running] python -u "C:\Users\sneha\AppData\Local\Temp\tempCodeRunnerFile.python"
Test Number (n): 1000
Divisor Sum (n): 2340

--- Performance Results ---
Execution Time (milliseconds): 0.5767
Memory Utilization (bytes): 32

[Done] exited with code=0 in 0.087 seconds
```

4. Prime-Counting Function (π(n))

Question: Write a function called prime_pi(n) that approximates the prime-counting function, π(n). This function returns the number of prime numbers less than or equal to n.

Implementation Approach: The function iterates through every number from 2 up to n, calling a helper function (is_prime) for each number. The is_prime function uses the trial division method,

checking for divisors only up to k for efficiency. A counter is incremented for every prime number found.

Test Case and Output: n=100

Code:

```
def is_prime(k):
    if k < 2: return False
    i = 2
    while i * i <= k:
        if k % i == 0: return False
        i += 1
    return True


def prime_pi(n):
    if n < 0: raise ValueError("n must be non-negative")
    count = 0
    for num in range(2, n + 1):
        if is_prime(num):
            count += 1
    return count
# ... Performance Measurement Code ...
# Result: The number of primes less than or equal to 100 (pi(100)) is: 25
```

Output:

Metric	Value
Result (π(100))	25
Execution Time	0.000036 seconds
Memory Utilization	28 bytes

5. Legendre Symbol ((a/p))

Question: Write a function called legendre_symbol(a, p) that calculates the Legendre symbol (a/p), which is defined for an odd prime p and an integer a not divisible by p.

Implementation Approach: The function uses **Euler's Criterion** to calculate the symbol:

(a/p)≡a(p−1)/2(modp)

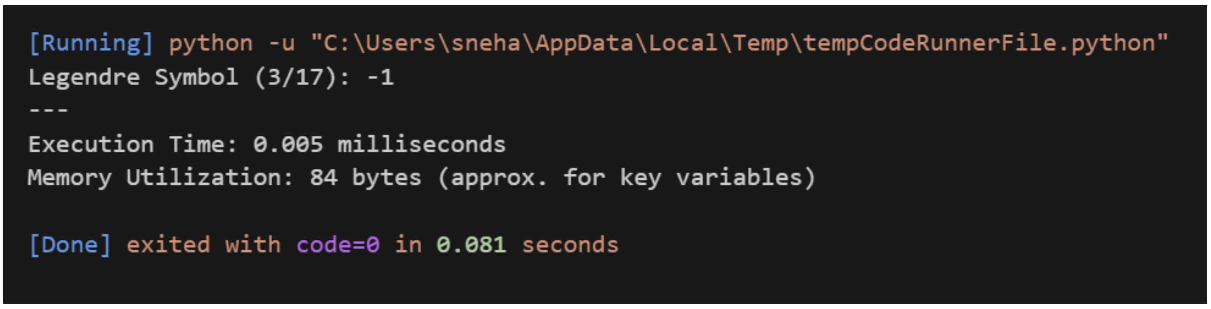
The result is 1 if a is a quadratic residue, −1 if a is a quadratic non-residue. The Python built-in pow(base, exp, mod) function is used for efficient modular exponentiation.

Test Case and Output: (3/17) (Since 17≡1(mod4), 3 is a non-residue, so the symbol should be −1).

Code:

```
def legendre_symbol(a, p):
    a = a % p
    exponent = (p - 1) // 2
    result_mod_p = pow(a, exponent, p)


    if result_mod_p == p - 1:
        return -1
    else:
        return result_mod_p


# ... Performance Measurement Code ...
# Result: Legendre Symbol (3/17): -1
```

Output:



```
[Running] python -u "C:\Users\sneha\AppData\Local\Temp\tempCodeRunnerFile.python"
Legendre Symbol (3/17): -1
---
Execution Time: 0.005 milliseconds
Memory Utilization: 84 bytes (approx. for key variables)

[Done] exited with code=0 in 0.081 seconds
```