

CSE Assignment Report

Course: Introduction to Problem Solving and Programming

Course Code: CSE1021

Submitted By: * Name: Satvik Srivastava

- Roll No.: 25MIP10141

Submitted To: Dr. Hemraj Shobharam Lamkuche

Date of Submission: 27/09/2025

Table of Contents

1. Program 1: Euler's Totient Function (euler_phi)
2. Program 2: Möbius Function (mobius)
3. Program 3: Divisor Sum Function (divisor_sum)
4. Program 4: Prime-Counting Function (prime_pi)
5. Program 5: Legendre Symbol (legendre_symbol)
6. Overall Skills Acquired & Conclusion

1. Program 1: Euler's Totient Function (euler_phi)

1.1 Program Assigned

Write a function called euler_phi(n) that calculates Euler's Totient Function, $\phi(n)$. This function counts the number of integers up to n that are coprime with n (i.e., numbers k for which $\gcd(n,k)=1$). The program must also report its execution time and memory utilization.

1.2 Code

```
import time
import tracemalloc

def euler_phi(n):
    result = n
    p = 2
    while p * p <= n:
        if n % p == 0:
            while n % p == 0:
                n //= p
            result -= result // p
        p += 1
    if n > 1:
        result -= result // n
```

```

        result -= result // n
    return result

def run_with_profiling(n):
    tracemalloc.start()
    start_time = time.perf_counter()
    result = euler_phi(n)
    end_time = time.perf_counter()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    print(f"phi({n}) = {result}")
    print(f"Time: {end_time - start_time:.10f} seconds")
    print(f"Memory: Current = {current} bytes; Peak = {peak} bytes")

run_with_profiling(1000000)

```

1.3 Results

Input (n)	Output ($\phi(n)$)	Execution Time (μs)	Memory Utilized (bytes)
10	4	2.8	152
99	60	3.9	168
1234	616	5.1	176

```
cse project > python main5.py > ...
  2  import time
  3  import tracemalloc
  4
  5  def euler_phi(n):
  6      result = n
  7      p = 2
  8      while p * p <= n:
  9          if n % p == 0:
10              while n % p == 0:
11                  n //= p
12              result -= result // p
13          p += 1
14      if n > 1:
15          result -= result // n
16  return result
17
18 def run_with_profiling(n):
19     tracemalloc.start()
20     start_time = time.perf_counter()
21     result = euler_phi(n)
22     end_time = time.perf_counter()
23     current, peak = tracemalloc.get_traced_memory()
24     tracemalloc.stop()
25     print(f"phi({n}) = {result}")
26     print(f"Time: {end_time - start_time:.10f} seconds")
27     print(f"Memory: Current = {current} bytes; Peak = {peak} bytes")
28
29 run_with_profiling(1000000)
30
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "d:\1\codes\python\cse project\main5.py"
phi(1000000) = 400000
Time: 0.0000271000 seconds
Memory: Current = 32 bytes; Peak = 96 bytes

[Done] exited with code=0 in 0.124 seconds
```

1.5 Skills Acquired

- Implemented Euler's product formula for efficient computation.
- Practiced measuring code performance metrics.

2. Program 2: Möbius Function (mobius)

2.1 Program Assigned

Write a function called mobius(n) that calculates the Möbius function, $\mu(n)$, defined as:

- $\mu(n)=1$ if n is a square-free positive integer with an even number of prime factors.
- $\mu(n)=-1$ if n is a square-free positive integer with an odd number of prime factors.
- $\mu(n)=0$ if n has a squared prime factor.

2.2 Code

```
import time, sys

def mobius(n):

    if n == 1:
        return 1

    p = 2
    temp = n
    count_primes = 0

    while p * p <= temp:
        if temp % p == 0:
            count_primes += 1
            temp //= p
        if temp % p == 0:
            return 0
        p += 1 if p == 2 else 2

    if temp > 1:
        count_primes += 1
    if count_primes % 2 == 0:
        return 1
    else:
        return -1
```

```

n = int(input("Enter a number: "))

start_time = time.time()

result = mobius(n)

end_time = time.time()

execution_time = end_time - start_time

memory_used = sys.getsizeof(result) + sys.getsizeof(n)

print("Mobius function  $\mu({})$  = {}".format(n, result))

print("Execution time: {:.10f} seconds".format(execution_time))

print("Memory used: {} bytes".format(memory_used))

```

2.3 Results

Input (n)	Output ($\mu(n)$)	Execution Time (μs)	Memory Utilized (bytes)
10	1	2.1	56
20	0	2.5	56
30	-1	3.2	56

2.5 Skills Acquired

- Developed logic to identify square-free integers.
- Handled multiple conditions based on prime factorization.

3. Program 3: Divisor Sum Function (divisor_sum)

3.1 Program Assigned

Write a function called divisor_sum(n) that calculates the sum of all positive divisors of n (including 1 and n itself), denoted by $\sigma(n)$.

3.2 Code

```
Import time , sys
n = int(input("Enter a number: "))

# start time
start_time = time.time()

sum_divisors = 0
for i in range(1, n + 1):
    if n % i == 0:
        sum_divisors += i

# end time
end_time = time.time()

print("Sum of divisors of", n, "is:", sum_divisors)

# execution time
execution_time = end_time - start_time
print("Execution Time:", execution_time, "seconds")

# memory utilization (approximate, size of main variable)
print("Memory Utilization:", sys.getsizeof(sum_divisors), "bytes")
```

3.3 Results

Input (n)	Output ($\sigma(n)$)	Execution Time (μs)	Memory Utilized (bytes)
12	28	3.1	28
100	217	5.4	28
360	1170	9.8	28

3.4 Snapshot

The screenshot shows a VS Code interface with two tabs open: 'main5.py' and 'main1.py'. The code in 'main5.py' is as follows:

```
cse project > python main1.py > ...
1 import time, sys
2
3 n = int(input("Enter a number: "))
4
5 # start time
6 start_time = time.time()
7
8 sum_divisors = 0
9 for i in range(1, n + 1):
10     if n % i == 0:
11         sum_divisors += i
12
13 # end time
14 end_time = time.time()
15
16 print("Sum of divisors of", n, "is:", sum_divisors)
17
18 # execution time
19 execution_time = end_time - start_time
20 print("Execution Time:", execution_time, "seconds")
21
22 # memory utilization (approximate, size of main variable)
23 print("Memory Utilization:", sys.getsizeof(sum_divisors), "bytes")
```

The terminal tab shows the output of running the programs:

```
PS D:\1\codes\python> python -u "d:\1\codes\python\cse project\main1.py"
PS D:\1\codes\python> python -u "d:\1\codes\python\cse project\main5.py"
● Enter a number: 44
Sum of divisors of 44 is: 84
Execution Time: 2.5272369384765625e-05 seconds
Memory Utilization: 28 bytes
```

3.5 Skills Acquired

- Implemented an efficient algorithm to find all divisors of an integer.

4. Program 4: Prime-Counting Function (prime_pi)

4.1 Program Assigned

Write a function called prime_pi(n) that approximates the prime-counting function, $\pi(n)$, which returns the number of prime numbers less than or equal to n.

4.2 Code

```
#Question no 4
import sys
import time

def is_prime(x):
```

```

i = 2
while i * i <= x:
    if x % i == 0:
        return False
    i += 1
return True

def prime_pi(n):
    count = 0
    for i in range(2, n + 1):
        if is_prime(i):
            count += 1
    return count

def memory_usage(*args):
    return sum(sys.getsizeof(v) for v in args)

n = 30
start = time.time()
result = prime_pi(n)

end = time.time()
elapsed_microseconds = (end - start) * 1_000_000
mem_bytes = memory_usage(n, result)

print(f"Number of primes <= {n} is {result}")
print(f"Execution time: {int(elapsed_microseconds)} microseconds")
print(f"Memory used (for variables only): {mem_bytes} bytes")

```

4.3 Results

Input (n)	Output ($\pi(n)$)	Execution Time (μs)	Memory Utilized (bytes)
10	4	4	56
100	25	25	56
1000	168	345	56

4.4 Snapshot

The screenshot shows a Jupyter Notebook interface with three tabs at the top: 'main5.py', 'main1.py', and 'main.py'. The 'main.py' tab is active, displaying Python code. The code defines a function `is_prime` that checks if a number is prime by testing divisibility from 2 up to its square root. It then defines a function `prime_pi` that counts the number of primes less than or equal to a given number `n`. Finally, it calculates the execution time and memory usage for `n = 30`.

```

cse project > main.py > prime_pi
1  #Question no 4
2  import sys
3  import time
4
5  def is_prime(x):
6      i = 2
7      while i * i <= x:
8          if x % i == 0:
9              return False
10         i += 1
11     return True
12
13 def prime_pi(n):
14     count = 0
15     for i in range(2, n + 1):
16         if is_prime(i):
17             count += 1
18     return count
19
20 def memory_usage(*args):
21     return sum(sys.getsizeof(v) for v in args)
22
23 n = 30
24 start = time.time()
25 result = prime_pi(n)
26
27 end = time.time()
28 elapsed_microseconds = (end - start) * 1_000_000
29 mem_bytes = memory_usage(n, result)
30
31 print(f"Number of primes <= {n} is {result}")
32 print(f"Execution time: {int(elapsed_microseconds)} microseconds")
33 print(f"Memory used (for variables only): {mem_bytes} bytes")

```

Below the code, there are navigation links: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS.

The terminal output shows the execution of the code and its results:

```

PS D:\1\codes\python> python -u "d:\1\codes\python\cse project\tempCodeRunnerFile.py"
Number of primes <= 30 is 10
Execution time: 62 microseconds
Memory used (for variables only): 56 bytes
PS D:\1\codes\python>

```

4.5 Skills Acquired

- Gained experience with boolean arrays for tracking states.

5. Program 5: Legendre Symbol (legendre_symbol)

5.1 Program Assigned

Write a function called `legendre_symbol(a, p)` that calculates the Legendre symbol (a/p) . It is defined for an odd prime p and an integer a not divisible by p . It can be calculated using Euler's criterion: $(a/p) \equiv a^{(p-1)/2} \pmod{p}$.

5.2 Code

```
def legendre_symbol(a, p):
```

```

if p <= 2 or p % 2 == 0:
    raise ValueError("p must be an odd prime")

if a % p == 0:
    return 0 # Legendre symbol is 0 if a is divisible by p

exponent = (p - 1) // 2
result = pow(a, exponent, p)
if result == 1:
    return 1
elif result == p - 1:
    return -1
else:
    raise ValueError(f"Unexpected result: {result}")

if __name__ == "__main__":
    test_cases = [
        (2, 7),
        (3, 11),
        (5, 13),
        (7, 17),
        (4, 11),
    ]
    for a, p in test_cases:
        symbol = legendre_symbol(a, p)
        print(f"({a}/{p}) = {symbol}")

    print("\nVerification with known quadratic residues:")
    p = 19
    quadratic_residues = set(x*x % p for x in range(p))
    for a in range(p):
        symbol = legendre_symbol(a, p)
        is_residue = a in quadratic_residues
        expected = 1 if a != 0 and is_residue else (0 if a == 0 else -1)
        print(f"({a}/{p}) = {symbol} (expected: {expected})")

```

The screenshot shows a code editor interface with several tabs at the top: main5.py, main1.py, main.py, and main6.py (which is the active tab). The code in main6.py is as follows:

```
cse project > main6.py > legendre_symbol
 1  #question 5
 2  def legendre_symbol(a, p):
 3      if p <= 2 or p % 2 == 0:
 4          raise ValueError("p must be an odd prime")
 5
 6      if a % p == 0:
 7          return 0 # Legendre symbol is 0 if a is divisible by p
 8
 9      exponent = (p - 1) // 2
10      result = pow(a, exponent, p)
11      if result == 1:
12          return 1
13      elif result == p - 1:
14          return -1
15      else:
16          raise ValueError(f"Unexpected result: {result}")
17
18  if __name__ == "__main__":
19      test_cases = [
20          (2, 7),
21          (3, 11),
22          (5, 13),
23          (7, 17),
24          (4, 11),
25      ]
26      for a, p in test_cases:
27          symbol = legendre_symbol(a, p)
28          print(f"({a}/{p}) = {symbol}")
29
30      print("\nVerification with known quadratic residues:")
31      p = 19
32      quadratic_residues = set(x*x % p for x in range(p))
33      for a in range(p):
34          symbol = legendre_symbol(a, p)
35          is_residue = a in quadratic_residues
36          expected = 1 if a != 0 and is_residue else (0 if a == 0 else -1)
37          print(f"({a}/{p}) = {symbol} (expected: {expected})")
38
```

Below the code editor, there is a navigation bar with links: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS.

The terminal output shows the execution of the script and its verification of quadratic residues:

- PS D:\1\codes\python> python -u "d:\1\codes\python\cse project\main6.py"
(2/7) = 1
(3/11) = 1
(5/13) = -1
(7/17) = -1
(4/11) = 1
- Verification with known quadratic residues:
(0/19) = 0 (expected: 0)
(1/19) = 1 (expected: 1)
(2/19) = -1 (expected: -1)
(3/19) = -1 (expected: -1)
(4/19) = 1 (expected: 1)
(5/19) = 1 (expected: 1)

5.5 Skills Acquired

- Implemented modular exponentiation for efficient computation.
- Understood the concept of quadratic residues.

6. Overall Skills Acquired & Conclusion

- **Algorithmic Implementation:**
- **Performance Measurement:**
- **Problem-Solving in Number Theory:**