VIT BHOPAL

Lab Manual

Practical and Skills Development

CERTIFICATE

THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY PERFORMED BY

Registration No :25MIP10141

Name of Student :SATVIK SRIVASTAVA

Course Name : Introduction to Problem Solving and Programming

Course Code : CSE1021

School Name : School of Computing Science Engineering and Artificial Intelligence (SCAI)

Slot : B11+B12+B13

Class ID : BL2025260100796

Semester : FALL 2025/26

Course Faculty Name : Dr. Hemraj S. Lamkuche

Signature:

VIT BHOPAL

Practical Index

S. No.	Title of Practical	Date of Submission	Signature of Faculty
1	Lucas Numbers	16/11/20225	
2	Chinese Remainder Theorem (CRT) Solver	16/11/20225	
3	Quadratic Residue Check	16/11/20225	
4	Order of an Element Modulo n	16/11/20225	
5	Fibonacci Prime Check	16/11/20225	

Practical No: 1

Date: 10/11/2025

TITLE: Write a function Lucas Numbers

Generator lucas_sequence(n) that generates the first n

Lucas numbers (similar to Fibonacci but starts with 2,

1).

AIM/OBJECTIVE(s): To write a Python function `mod_inverse(a, m)` that finds the integer x such that $(a \cdot x) \equiv 1 \pmod m$. The script should also measure the execution time and peak memory usage.

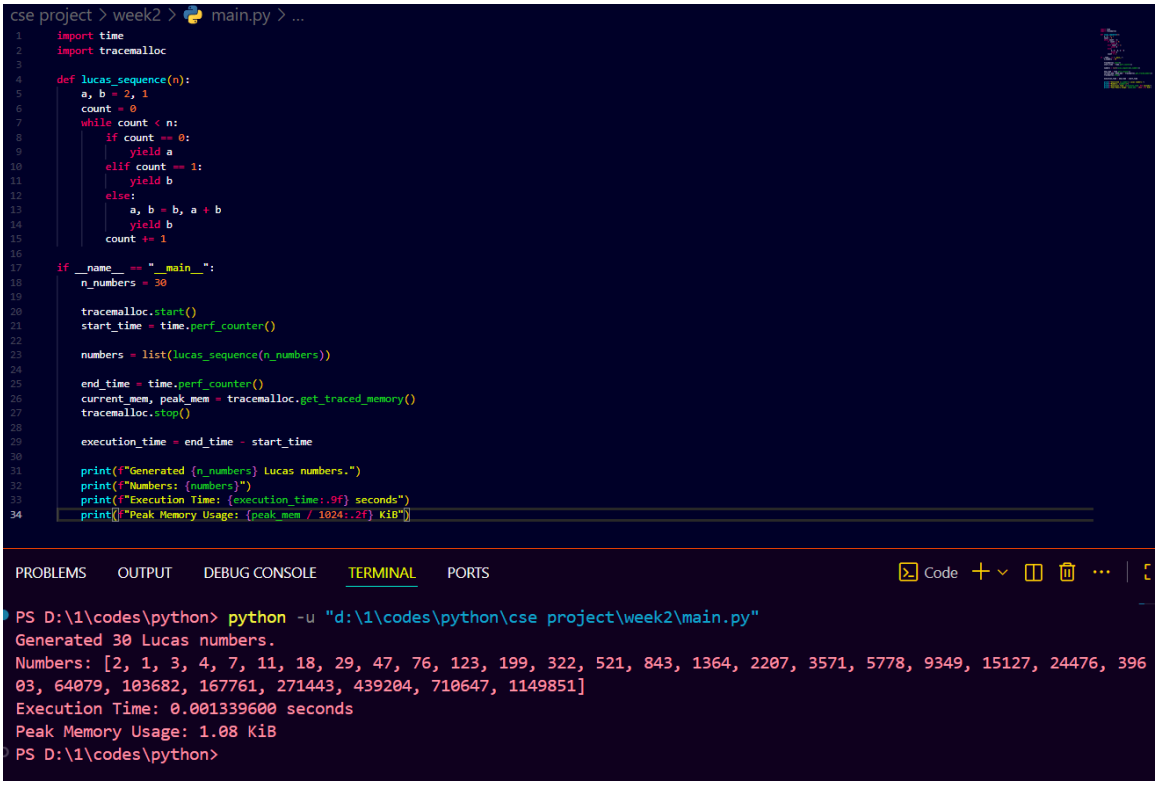
METHODOLOGY & TOOL USED:

Python programming language

BRIEF DESCRIPTION:

`mod_inverse(a, m)` that calculates the modular multiplicative inverse. It returns `None` if the modulus `m` is invalid ($m \le 1$) or if the inverse does not exist (caught by `ValueError`).

RESULTS ACHIEVED:



SKILLS ACHIEVED:

- Understanding the concept of a modular multiplicative inverse.
- Using Python's `pow()` function for advanced number theory operations.
- Using `try...except` blocks to handle expected mathematical errors (`ValueError`).
- Continued practice in performance measurement.

Practical No: 2

Date: 16/11/2025

TITLE: Chinese Remainder Theorem (CRT) Solver

AIM/OBJECTIVE(s) : Chinese Remainder Theorem (CRT) Solver

METHODOLOGY & TOOL USED:Python programming language

BRIEF DESCRIPTION:

The code defines a `crt` function that solves a system of congruences using the standard constructive algorithm for the Chinese Remainder Theorem. It relies on the `mod_inverse` helper function.

Result:

```python
import time
import tracemalloc
from functools import reduce

def mod_inverse(a, m):
    try:
        return pow(a, -1, m)
    except ValueError:
        return None

def crt(remainders, moduli):
    if len(remainders) != len(moduli):
        return None

    M = reduce(lambda a, b: a * b, moduli)

    total = 0
    for r_i, m_i in zip(remainders, moduli):
        M_i = M // m_i
        y_i = mod_inverse(M_i, m_i)

        if y_i is None:
            return None

        total += r_i * M_i * y_i

    return total % M

if __name__ == "__main__":

    remainders = [2, 3, 2]
    moduli = [3, 5, 7]

    tracemalloc.start()
    start_time = time.perf_counter()

    result = crt(remainders, moduli)

    end_time = time.perf_counter()
    current_mem, peak_mem = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    execution_time = end_time - start_time

    print(f"Solving system of congruences:")
    for r, m in zip(remainders, moduli):
        print(f"x ≡ {r} mod {m}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\1\codes\python> python -u "d:\1\codes\python\cse project\WEEK6\main2.py"

Result (x): 23
Execution Time: 0.000838100 seconds
Peak Memory Usage: 0.20 KiB
PS D:\1\codes\python>
```

SKILLS ACHIEVED :

- Understanding and implementing the Chinese Remainder Theorem algorithm.
- Combining multiple functions (`crt` and `mod_inverse`) to solve a complex problem.
- Using `zip` to iterate over multiple lists simultaneously.

Practical No: 3

Date: 16/11/2025

TITLE :Quadratic Residue Check

AIM/OBJECTIVE(s):To write a Python function `is_quadratic_residue(a, p)` that checks if the congruence $x^2 \equiv a \pmod p$ has a solution, assuming p is a prime number.

METHODOLOGY & TOOL USED

Python programming language

BRIEF DESCRIPTION:
The code defines a function `is_quadratic_residue(a, p)` that implements Euler's criterion to determine if `a` is a quadratic residue modulo `p`. It handles the special case of p=2 and the trivial case of a=0

RESULTS ACHIEVED:

```python
import time
import tracemalloc

def is_quadratic_residue(a, p):
    if p <= 1:
        return False

    a = a % p
    if a == 0:
        return True

    if p == 2:
        return a == 1

    exponent = (p - 1) // 2
    result = pow(a, exponent, p)

    return result == 1

if __name__ == "__main__":

    a_res = 2
    p_mod = 7

    tracemalloc.start()
    start_time = time.perf_counter()

    result = is_quadratic_residue(a_res, p_mod)

    end_time = time.perf_counter()
    current_mem, peak_mem = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    execution_time = end_time - start_time

    print(f"Checking if {a_res} is a quadratic residue mod {p_mod}.")
    print(f"Result: {result}")
    print(f"Execution Time: {execution_time:.9f} seconds")
    print(f"Peak Memory Usage: {peak_mem / 1024:.2f} KiB")


    a_non_res = 3
    p_mod_non = 7

    tracemalloc.start()
    start_time = time.perf_counter()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\1\codes\python> python -u "d:\1\codes\python\cse project\WEEK6\main3.py"
Checking if 3 is a quadratic residue mod 7.
Result: False
Execution Time: 0.000006300 seconds
Peak Memory Usage: 0.00 KiB
PS D:\1\codes\python>
```

SKILLS ACHIEVED:

- Understanding the concept of quadratic residues.
- Implementation of Euler's criterion.
- Efficient modular exponentiation using `pow()`.
- Handling edge cases in number theory problems (e.g., $p=2$).

Practical No: 4

Date: 16/11/2025

TITLE:Order of an Element Modulo n

AIM/OBJECTIVE(s) : To write a Python function `order_mod(a, n)` that finds the smallest positive integer k (the order) such that a^k= 1 mod(n)

METHODOLOGY & TOOL USED:

Python programming language

BRIEF DESCRIPTION :

Python function `order_mod(a, n)` that finds the smallest positive integer k (the order) such that a^k= 1 mod(n)

1. Order of $3 \pmod 7$. ($\varphi(7)=6$, divisors are 1, 2, 3, 6).
2. Order of $5 \pmod{24}$. ($\varphi(24)=8$, divisors are 1, 2, 4, 8).

Result:

```python
import time
import tracemalloc
import math

def get_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
    return result

def get_divisors(n):
    divs = set()
    for i in range(1, int(n**0.5) + 1):
        if n % i == 0:
            divs.add(i)
            divs.add(n // i)
    return sorted(list(divs))

def order_mod(a, n):
    if n <= 1 or math.gcd(a, n) != 1:
        return None

    phi_n = get_phi(n)
    divisors = get_divisors(phi_n)

    for k in divisors:
        if pow(a, k, n) == 1:
            return k

    return None

if __name__ == "__main__":

    a_val = 3
    n_val = 7

    tracemalloc.start()
    start_time = time.perf_counter()

    result = order_mod(a_val, n_val)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Finding the order of 5 mod 24.
Result (k): 2
Execution Time: 0.000043000 seconds
Peak Memory Usage: 0.43 KiB
PS D:\1\codes\python>
```

SKILLS ACHIEVED :

- Understanding the "order" of an element in a modular group.
- Implementation of Euler's totient function.
- Algorithm to find all divisors of a number.
- Applying number theory (Lagrange's/Euler's theorem) to optimize an algorithm.

Practical No: 5

Date: 16/11/2025

TITLE:Fibonacci Prime Check

AIM/OBJECTIVE(s):

To write a Python function `is_fibonacci_prime(n)` that checks if a given integer `n` is both a Fibonacci number and a prime number.

METHODOLOGY & TOOL USED:Python programming language

BRIEF DESCRIPTION:

The code defines `is_fibonacci_prime(n)` along with its required helper functions. The `is_fibonacci` check is based on a mathematical property, avoiding the need to generate Fibonacci numbers.

The main execution block (`if __name__ == "__main__":`) measures the time and memory for two cases:

1. `13` (which is both prime and Fibonacci).
2. `21` (which is Fibonacci but not prime).

RESULT:

cse project > WEEK6 > main5.py > ...

```python
import time
import tracemalloc

def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

def is_perfect_square(k):
    if k < 0:
        return False
    s = int(k**0.5)
    return s * s == k

def is_fibonacci(n):
    if n < 0:
        return False

    test_1 = 5 * n**2 + 4
    test_2 = 5 * n**2 - 4
    return is_perfect_square(test_1) or is_perfect_square(test_2)

def is_fibonacci_prime(n):
    return is_prime(n) and is_fibonacci(n)

if __name__ == "__main__":

    test_number = 13

    tracemalloc.start()
    start_time = time.perf_counter()

    result = is_fibonacci_prime(test_number)

    end_time = time.perf_counter()
    current_mem, peak_mem = tracemalloc.get_traced_memory()
    tracemalloc.stop()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Checking if 21 is a Fibonacci Prime.
Result: False
Execution Time: 0.000004800 seconds
Peak Memory Usage: 0.00 KiB
PS D:\1\codes\python>
```

SKILLS ACHIEVED:

- Understanding and implementing mathematical properties of number sequences.
- Writing helper functions to build up complex logic (`is_perfect_square`, `is_fibonacci`, `is_prime`).
- Combining logical conditions (`and`) to satisfy multiple criteria.