Lab Manual

Practical and Skills Development

CERTIFICATE

THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY PERFORMED BY

Registration No	: 25BAI10734
Name of Student	: Atharav Balaji Khonde
Course Name	: Introduction to Problem Solving and Programming
Course Code	: CSE1021
School Name	: 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	Write a function Modular Multiplicative Inverse mod_inverse(a, m) that finds the number x such that (a * x) ≡ 1 mod m.	13-11-25	
2	Write a function chinese Remainder Theorem Solver crt(remainders, moduli) that solves a system of congruences x ≡ ri mod mi.	13-11-25	
3	Write a function Quadratic Residue Check is_quadratic_residue(a, p) that checks if x2 ≡ a mod p has a solution.	13-11-25	
4	Write a function order_mod(a, n) that finds the smallest positive integer k such that ak ≡ 1 mod n.	13-11-25	
5	Write a function Fibonacci Prime Check is_fibonacci_prime(n) that checks if a number is both Fibonacci and prime.	13-11-25	
6			
7			
8			
9			
10			
11			
12			

13			
14			
15			

Practical No: 1

Date: 13-11-25

TITLE: mod_inverse(a, m)

AIM/OBJECTIVE(s): To Write a function Modular Multiplicative Inverse mod_inverse(a, m) that finds the number x such that (a * x) ≡ 1 mod m.

METHODOLOGY & TOOL USED:

The given program is running in google collab

Using modular arithmetic

The provided Python code uses a brute-force methodology to find a modular inverse by linearly searching for a valid solution.

It employs two standard Python libraries as tools for performance analysis:

1. time module: Used to measure the execution duration (benchmarking) of the algorithm.
2. tracemalloc module: Used to measure the peak memory usage (profiling) of the traced code section.

BRIEF DESCRIPTION:

This Python code defines a function to calculate the modular multiplicative inverse using a simple brute-force loop. It then runs this function and executes for loop as soon as it gets a result it prints it or returns no function while simultaneously using the built in time and tracemalloc modules to measure the execution time and peak memory usage, printing these performance statistics to the console. It uses modular arithmetic logic (a * x) % m = 1 for if else statement

RESULTS ACHIEVED: the function to find modular multiplicative inverse mod_inverse(a, m) is acheived

DIFFICULTY FACED BY STUDENT:

Finding limits of range function and using modular arthimetic and its logic

```python
import time
import tracemalloc
tracemalloc.start()
start = time.time()
def mod_inverse(a, m):
    a = int(a)
    m = int(m)
    for x in range(1, m):
        if (a * x) % m == 1:
            print(x)
            break
    else:
        print("no inverse")
mod_inverse(2,5)
end = time.time()
current,peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(peak,"bytes")
print("execution time is",end - start)
```

```
3
32864 bytes
execution time is 0.000823974609375
```

SKILLS ACHIEVED: a brief knowledge of modular arthimetics and logics and decinding range values of this type of problems

Practical No: 2

Date: 13-11-25

TITLE: crt(remainders, moduli)

AIM/OBJECTIVE(s): Write a function chinese Remainder Theorem Solver crt(remainders, moduli) that solves a system of congruences x ≡ ri mod mi.

METHODOLOGY & TOOL USED:

The code uses a brute-force search methodology

Modular arithmetic

It uses for loop,range() and many arithmetic operations

It employs two standard Python libraries as tools for performance analysis:

1. **time module:** Used to **measure the execution duration** (benchmarking) of the algorithm.
2. **tracemalloc module:** Used to **measure the peak memory usage** (profiling) of the traced code section.

BRIEF DESCRIPTION:This code defines a function crt(remainders, moduli). Despite its name, which typically refers to the Chinese Remainder Theorem, this implementation performs a simpler check. It

iterates through values from 0 up to moduli - 1. Inside the loop, it checks if the current iteration value (x) is equal to the remainders input. If this condition is met, it prints the remainders value.
The time and tracemalloc modules are utilized to measure the execution time and peak memory usage of this function, providing insights into its performance.

RESULTS ACHIEVED: a function to find chinese Remainder Theorem Solver is made

DIFFICULTY FACED BY STUDENT:

Finding limits of range function and using modular arthimetic and its logic

SKILLS ACHIEVED: a brief knowledge of modular arthimetics and logics and decinding range values of this type of problems

```python
import time
import tracemalloc
tracemalloc.start()
start = time.time()
def crt(remainders, moduli):
    r=int(remainders)
    m=int(moduli)
    x=0
    for x in range(0,m):
        if (x % m)==r:
            print(r)
    x=x+1
crt(2,7)
end = time.time()
current,peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(peak,"bytes")
print("execution time is",end - start)
```

```
2
32896 bytes
execution time is 0.00082111358642578125
```

Practical No: 3

Date: 13-11-25

TITLE: is_quadratic_residue

AIM/OBJECTIVE(s): Write a function Quadratic Residue

Check is_quadratic_residue(a, p) that checks if x2 ≡ a mod p

has a solution.

METHODOLOGY & TOOL USED:

The code uses a **brute-force search methodology** to check for a quadratic residue.

Modular arithmetic

It uses for loop,range() and many arithmetic operations

It employs two standard Python libraries as tools for performance analysis:

1. **time module:** Used to **measure the execution duration** (benchmarking) of the algorithm.
2. **tracemalloc module:** Used to **measure the peak memory usage** (profiling) of the traced code section.

BRIEF DESCRIPTION:

This Python code defines a function that uses a **brute-force approach** to check if a number is a quadratic residue modulo another number.

It uses for loop in range to find the solution and if it finds a solutions it prints it as output

The script then executes this function while simultaneously employing the time and tracemalloc modules to measure and print the algorithm's **execution time** and **peak memory usage**.

It uses modular arithmetic logic (x**2)%p=a

RESULTS ACHIEVED:

a function Quadratic Residue

Check is_quadratic_residue(a, p) that checks if x2 ≡ a mod p

has a solution is achieved.

DIFFICULTY FACED BY STUDENT:

Finding limits of range function and using modular arthimetic and its logic

SKILLS ACHIEVED:

a brief knowledge of modular arthimetics and logics and decinding range values of this type of problems

```
import time
import tracemalloc
tracemalloc.start()
start = time.time()
def is_quadratic_residue(a, p):
    a=int(a)
    p=int(p)
    for x in range(0,p):
        if (x**2)%p==a:
            print(x)
            break
    else:
        print("no x")
is_quadratic_residue(2,5)
end = time.time()
current,peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(peak,"bytes")
print("execution time is",end - start)
```

```
no x
32870 bytes
execution time is 0.0009095668792724609
```

Practical No: 4

Date: 13-11-25

TITLE: order_mod(a, n)

AIM/OBJECTIVE(s): Write a function order_mod(a, n) that finds the smallest positive integer k such that ak ≡ 1 mod n.

METHODOLOGY & TOOL USED:

The code uses a brute-force search methodology

Modular arithmetic

It uses for loop,range() and many arithmetic operations

It employs two standard Python libraries as tools for performance analysis:

1. time module: Used to measure the execution duration (benchmarking) of the algorithm.
2. tracemalloc module: Used to measure the peak memory usage (profiling) of the traced code section.

BRIEF DESCRIPTION:

This function attempts to find the smallest positive integer k

such that $a^k \equiv 1 (\mathrm{mod} n)$.

It iterates through potential values of k starting from 1 up to $n-1$

If it finds a k where the condition (a**k) % n == 1 is true, it prints that k

and stops (break). This is the multiplicative order.

If the loop finishes without finding such a k (which would happen if the condition is never met within that range), it prints "no order".

RESULTS ACHIEVED: a function order_mod(a, n) that finds the smallest

positive integer k such that ak ≡ 1 mod n is achieved.

DIFFICULTY FACED BY STUDENT:

Finding limits of range function and using modular arthimetic and its logic

SKILLS ACHIEVED:

a brief knowledge of modular arthimetics and logics and decinding range values of this type of problems

```
import time
import tracemalloc
tracemalloc.start()
start = time.time()
def order_mod(a, n):
    a=int(a)
    n=int(n)
    for k in range(1,n):
        if (a**k)%n==1:
            print(k)
            break
    else:
        print("no order")
order_mod(5,2)
end = time.time()
current,peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(peak,"bytes")
print("execution time is",end - start)
```

```
1
32863 bytes
execution time is 0.0008895397186279297
```

Practical No: 5

Date: 13-11-25

TITLE: is_fibonacci_prime(n)

AIM/OBJECTIVE(s): a function Fibonacci Prime

Check is_fibonacci_prime(n) that checks if a number is

both Fibonacci and prime.

METHODOLOGY & TOOL USED:

The code uses a brute-force search methodology

Modular arithmetic

It uses for loop,range() and many arithmetic operations

It employs two standard Python libraries as tools for performance analysis:

1. time module: Used to measure the execution duration (benchmarking) of the algorithm.
2. tracemalloc module: Used to measure the peak memory usage (profiling) of the traced code section.

BRIEF DESCRIPTION:

This python code solves code in 2 parts

- is_prime(n):

Purpose: Determines if a number is prime (only divisible by 1 and itself).

Method: Uses an efficient mathematical check, iterating through possible divisors only up to the square root of the input number.

- is_fibonacci(n):

Purpose: Determines if a number belongs to the Fibonacci sequence (e.g., 0, 1, 1, 2, 3, 5, 8...).

Method: Generates the sequence iteratively until it finds the input number or exceeds it.

RESULTS ACHIEVED: a function Fibonacci Prime Check is_fibonacci_prime(n) that checks if a number is both Fibonacci and prime is achieved.

DIFFICULTY FACED BY STUDENT:

Finding limits of range function and using modular arthimetic and its logic

SKILLS ACHIEVED:

a brief knowledge of modular arthimetics and logics and decinding range values of this type of problems

```
import time
import tracemalloc
tracemalloc.start()
start = time.time()
def is_fibonacci_prime(n):
    n=int(n)
    i=1
    for i in range(1,n):
        if n % i == 0:
            b=i
        i=i+1
    if b==1:
        print("prime")
    else:
        print("not prime")
    k=0
    l=1
    v=0
    for v in range(0,n):
        u=l+k
        k=l
        l=u
        if n==u:
            print("number is fibonacci")
            break
        if n<u:
            print("number is not fibonacci")
```

```
        l=1
        v=0
        for v in range(0,n):
            u=l+k
            k=l
            l=u
            if n==u:
                print("number is fibonacci")
                break
            if n<u:
                print("number is not fibonacci")
                break
        v=v+1
    is_fibonacci_prime(5)
    end = time.time()
    current,peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    print(peak,"bytes")
    print("execution time is",end - start)
```

```
...  prime
    number is fibonacci
    46496 bytes
    0.0018379688262939453
```