

ASSIGNMENT-5.5

M.HARIKRISHNA

2203A52100

B-50

Task 1

Task Description #1 (Transparency in Algorithm Optimization)

Task: Use AI to generate two solutions for checking prime

numbers:

- Naive approach(basic)
- Optimized approach

Prompt:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

Expected Output:

- Code for both methods.
- Transparent explanation of time complexity.
- Comparison highlighting efficiency improvements.

Prompt:

Generate Python code for two prime number checking methods (naive and optimized) and explain how the optimized version improves performance.

Code:

```
import math
```

```
def is_prime_naive(n):
```

```
    if n <= 1:
```

```
        return False
```

```
    for i in range(2, n):
```

```
        if n % i == 0:
```

```

        return False
    return True

def is_prime_optimized(n):
    if n <= 1:
        return False    for i in range(2,
int(math.sqrt(n)) + 1):      if n % i ==
0:          return False    return True

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

print("Naive Method:", "Prime Number" if is_prime_naive(num) else "Not Prime Number")
print("Optimized Method:", "Prime Number" if is_prime_optimized(num) else "Not Prime
Number")

print("Naive Time Complexity: O(n)")
print("Optimized Time Complexity: O( $\sqrt{n}$ )")

```

Output:

\sqrt{n})'."/>

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive/Documents/AI CODING/ai-coding-5.5.py"
Enter a number: 53
Naive Method: Prime Number
Optimized Method: Prime Number
Naive Time Complexity: O(n)
Optimized Time Complexity: O( $\sqrt{n}$ )
PS C:\Users\prash>

```

Code Explanation:

This program checks whether a number is prime using two methods. The naive method checks all numbers from 2 to $n-1$, making it slower. The optimized method checks only up to the square root of n , reducing the number of iterations. The naive method has time complexity $O(n)$, while the optimized method has time complexity $O(\sqrt{n})$. The optimized method improves performance especially for large numbers.

Task 2

Task Description #2 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Instructions:

1. Ask AI to add clear comments explaining recursion.
2. Ask AI to explain base cases and recursive calls.

Expected Output:

- Well-commented recursive code.
- Clear explanation of how recursion works.
- Verification that explanation matches actual execution.

Prompt:

Generate recursive Fibonacci Python code with comments explaining recursion, base case, and recursive calls.

Code:

```
def fibonacci(n):  
    """Calculate the nth Fibonacci number  
    recursively."""    if n <= 0:        return 0    elif n == 1:  
    return 1    else:  
        return fibonacci(n - 1) + fibonacci(n - 2)  
  
n = int(input("Enter Fibonacci index: "))  
  
print(f"fibonacci({n}) =", fibonacci(n))
```

Output:

```
Optimized time complexity: O(n)  
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe"  
Enter Fibonacci index: 6  
Fibonacci(6) = 8  
PS C:\Users\prash> █
```

Code Explanation:

This program checks whether a number is prime using two methods. The naive method checks all numbers from 2 to $n-1$, making it slower. The optimized method checks only up to the square root of n , reducing the number of iterations. The naive method has time

complexity $O(n)$, while the optimized method has time complexity $O(\sqrt{n})$. The optimized method improves performance especially for large numbers.

Task 3

Task Description #3 (Transparency in Error Handling) Task:

Use AI to generate a Python program that reads a file and processes data.

Prompt:

“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

- Code with meaningful exception handling.
- Clear comments explaining each error scenario.
- Validation that explanations align with runtime behavior."

Prompt:

Generate Python file reading code with proper exception handling and explanations.

Code:

try:

 file_name = input("Enter file name: ")

 with open(file_name, "r") as file:

 data = file.read()

 print(data)

 except FileNotFoundError:

 print("File not found")

 except PermissionError:

 print("Permission

denied") except Exception as

 e: print("Error:", e)

Output:

```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive/Documents/AI CODING/ai-coding-5.5.py"
Enter file name: sandy.txt
File not found
PS C:\Users\prash> █
```

Code Explanation:

This program reads a file safely using exception handling. FileNotFoundError handles missing files. PermissionError handles restricted file access. The general Exception block handles unknown errors. This ensures program does not crash.

Task 4

Task Description #4 (Security in User Authentication)

Task: Use an AI tool to generate a Python-based login system.

Analyze: Check whether the AI uses secure password handling practices.

Expected Output:

- Identification of security flaws (plain-text passwords, weak validation).
- Revised version using password hashing and input validation.
- Short note on best practices for secure authentication.

Prompt:

Generate Python login system and analyze security flaws. Then provide improved version using password hashing and input validation.

Code:

```
import hashlib
```

```
stored_username = "admin" stored_password =
hashlib.sha256("admin123".encode()).hexdigest()
```

```
username = input("Enter username: ").strip()
password = input("Enter password: ").strip()
```

```
hashed_password = hashlib.sha256(password.encode()).hexdigest()

if username == stored_username and hashed_password == stored_password:
    print("Login Successful")
else:
    print("Invalid Login")
```

Output:

```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive/D
Enter username: admin
Enter password: admin123
Login Successful
PS C:\Users\prash> █
```

Code explanation:

The insecure version stores password in plain text. The secure version uses hashing, which converts password into encrypted format. Hashing improves security and protects user data.

Task 5

Task Description #5 (Privacy in Data Logging)

Task: Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp).

Analyze: Examine whether sensitive data is logged unnecessarily or insecurely.

Expected Output:

- Identified privacy risks in logging.
- Improved version with minimal, anonymized, or masked logging.
- Explanation of privacy-aware logging principles.

Prompt:

Generate Python logging script for user activity. Then improve it using privacy-aware logging techniques.

Code:

```
import datetime
```

```
username = input("Enter username: ")  
ip_address = input("Enter IP address: ")  
  
masked_username = username[0] + "***"  
masked_ip = ip_address[:-3] + "***"  
  
timestamp = datetime.datetime.now()  
  
print("Log:", masked_username, masked_ip, timestamp)
```

Output:

```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive - Microsoft 365/PycharmProjects/PrivacyAwareLogging/main.py"  
Enter username: sandeep  
Enter IP address: 192.168.1.25  
Log: s*** 192.168.1*** 2026-02-12 18:27:13.803858  
PS C:\Users\prash>
```

Code explanation:

Risky logging stores full sensitive data. Privacy-safe logging masks sensitive information. This protects user privacy and follows data protection standards.
