SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: M. Tech/MCA/MSC		**Assignment Type: Lab**	**AcademicYear:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Course Code		**Course Title**	AI Assisted Problem Solving Using Python
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week5- Tuesday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:12.3(Present assignment number) /**24**(Total number of assignments)			

Q. No.	Question	*Expected Time to complete*
1	**Lab 12 – Algorithms with AI Assistance: Sorting, searching, and optimizing algorithms** **Lab Objectives** • To implement classical algorithms (sorting, searching) with the help of AI tools. • To analyze AI suggestions for efficiency and correctness. • To explore AI-assisted optimizations of existing algorithms. • To compare naive vs. optimized approaches generated by AI. **Learning Outcomes** After completing this lab, students will be able to: 🎬 Implement sorting and searching algorithms using AI suggestions. 🎬 Compare AI-generated algorithm variants in terms of readability and efficiency. 🎬 Use AI to optimize brute-force algorithms into more efficient ones. 🎬 Analyze algorithm complexity (time and space) with AI explanations. 🎬 Critically reflect on correctness, clarity, and maintainability of AI-generated algorithms. **Task Description #1 – Linear Search implementation** Task: Write python code for linear_search () function to search a value in a list and extract its index. CODE:	Week5- Tuesday

```python
def linear_search(arr, target):
    """
    Searches for the target in the list and returns its index if found, else -
    """
    for index, value in enumerate(arr):
        if value == target:
            return index
    return -1
# Main program
if __name__ == "__main__":
    try:
        # Get list input from user
        user_input = input("Enter list elements separated by spaces: ")
        arr = user_input.split()

        # Get target value to search
        target = input("Enter the value to search for: ")

        # Perform linear search
        index = linear_search(arr, target)

        # Display result
        if index != -1:
            print(f"Value '{target}' found at index {index}.")
        else:
            print(f"Value '{target}' not found in the list.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

OUTPUT:

```
pp assigenment 12/12.1.py"
Enter list elements separated by spaces: 1 2 3 4
Enter the value to search for: 2
Value '2' found at index 1.
```

Task Description #2 – Sorting Algorithms

Task: Ask AI to implement Bubble Sort and check sorted output

CODE:

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Track if any swaps happen in this pass
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap if elements are in wrong order
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break  # List is already sorted
    return arr

# Main program
if __name__ == "__main__":
    try:
        # Get input from user
        user_input = input("Enter numbers separated by spaces: ")
        numbers = list(map(int, user_input.strip().split()))

        # Sort the list
        sorted_numbers = bubble_sort(numbers)

        # Display the result
        print("Sorted list:", sorted_numbers)
    except ValueError:
        print("Please enter valid integers separated by spaces.")
```

OUTPUT:

```
Enter numbers separated by spaces: 3 2 4  6 7
Sorted list: [2, 3, 4, 6, 7, 8]
```

Task Description #3 – Optimization

Task: Write python code to solve below case study using linear optimization

Consider a chocolate manufacturing company that produces only two types of chocolate i.e. A and B. Both the chocolates require Milk and Choco only.
To manufacture each unit of A and B, the following quantities are required:
Each unit of A requires 1 unit of Milk and 3 units of Choco
Each unit of B requires 1 unit of Milk and 2 units of Choco
The company kitchen has a total of 5 units of Milk and 12 units of Choco. On each sale, the company makes a profit of Rs 6 per unit A sold and Rs 5 per unit B sold.
Now, the company wishes to maximize its profit. How many units of A and B should it produce respectively?

Email: dr.vvrresearch@gmail.com
(Dr.V.Venkataramana Veeramsetty)

CODE:

```
from pulp import LpMaximize, LpProblem, LpVariable, value

# Define the optimization problem
model = LpProblem("Chocolate_Profit_Maximization", LpMaximize)

# Define decision variables
x = LpVariable("Chocolate_A", lowBound=0, cat='Integer')
y = LpVariable("Chocolate_B", lowBound=0, cat='Integer')

# Add constraints
model += x + y <= 5          # Milk constraint
model += 3*x + 2*y <= 12     # Choco constraint

# Define objective function
model += 6*x + 5*y           # Profit function

# Solve the problem
model.solve()

# Output results
print("Optimal Production Plan:")
print(f"Produce {int(x.value())} units of Chocolate A")
print(f"Produce {int(y.value())} units of Chocolate B")
print(f"Maximum Profit: Rs {int(value(model.objective))}")
```

OUTPUT:

```
Optimal Production Plan:
Produce 2 units of Chocolate A
Produce 3 units of Chocolate B
Maximum Profit: Rs 27
PS C:\Users\91630\OneDrive\Desktop\aipp assigenment 12
```

Task Description #4 – Gradient Descent Optimization

Task: Write python code to find value of x at which the function $f(x)=2X^3+4x+5$ will be minimum

CODE:

```python
12.4.py > ...
def f(x):
    return 2 * x**3 + 4 * x + 5

def df(x):
    # Derivative of f(x): f'(x) = 6x^2 + 4
    return 6 * x**2 + 4

def gradient_descent(x0, learning_rate, iterations):
    x = x0
    for i in range(iterations):
        grad = df(x)
        x = x - learning_rate * grad
        print(f"Iteration {i+1}: x = {x:.6f}, f(x) = {f(x):.6f}")
    return x

# Main program
if __name__ == "__main__":
    try:
        x0 = float(input("Enter initial guess for x: "))
        learning_rate = float(input("Enter learning rate (e.g., 0.01): "
        iterations = int(input("Enter number of iterations: "))

        result = gradient_descent(x0, learning_rate, iterations)
        print(f"\nEstimated minimum at x = {result:.6f}, f(x) = {f(resul
    except Exception as e:
        print(f"Error: {e}")
```

OUTPUT:

```
Enter initial guess for x: 2
Enter learning rate (e.g., 0.01): 0.01
Enter number of iterations: 2
Iteration 1: x = 1.720000, f(x) = 22.056896
Iteration 2: x = 1.502496, f(x) = 17.793736
```