AI ASSISTED CODING

Program :B.tech(CSE)

Name :ANANTHA MANIDEEP

En No. :2403A52078

Batch No. :02

Date :22/10/2025

LAB ASSIGNMENT-12.3

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.
Week5 -
Monday

• 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 it's index.

PROMPT: Write a Python function named linear_search() that takes a list and a target value as input, and returns the index of the target value if it exists in the list. If the value is not found, the function should return -1.

CODE&OUTPUT:

```python
def linear_search(input_list, target_value):
  """
  Performs a linear search on a list to find the index of a target value.

  Args:
    input_list: The list to search within.
    target_value: The value to search for.

  Returns:
    The index of the target value if found, otherwise -1.
  """
  for index, value in enumerate(input_list):
    if value == target_value:
      return index
  return -1

# Example usage:
my_list = [10, 25, 5, 18, 30, 12]
target = 18
index = linear_search(my_list, target)

if index != -1:
  print(f"Target {target} found at index: {index}")
```

```python
else:
  print(f"Target {target} not found in the list.")

target = 50
index = linear_search(my_list, target)

if index != -1:
  print(f"Target {target} found at index: {index}")
else:
  print(f"Target {target} not found in the list.")
```

```
Target 18 found at index: 3
Target 50 not found in the list.
```

EXPLAINATION:

- def linear_search(input_list, target_value):: This line defines a function named linear_search that takes two arguments: input_list (the list to search within) and target_value (the value you're looking for).
- """ Docstring """: This is a docstring, which explains what the function does, its arguments, and what it returns.
- for index, value in enumerate(input_list):: This loop iterates through the input_list. enumerate is used to get both the index and the value of each element in the list.
- if value == target_value:: Inside the loop, this checks if the current value is equal to the target_value.
- return index: If the target value is found, the function immediately returns the index where it was found.

- return -1: If the loop finishes without finding the target value (meaning the if condition was never met), the function returns -1 to indicate that the target was not found.

Task Description #2 – Sorting Algorithms

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

PROMPT: Write a Python function called bubble_sort() that takes a list of numbers as input and sorts the list in ascending order using the Bubble Sort algorithm. After sorting, print or return the sorted list to verify the output**.**

CODE&OUTPUT:

```
def bubble_sort(input_list):
    """
    Sorts a list of numbers in ascending order using the Bubble Sort algorithm.

    Args:
        input_list: The list of numbers to sort.

    Returns:
        The sorted list.
    """
    n = len(input_list)
    # Traverse through all list elements
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n - i - 1):
            # Traverse the list from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if input_list[j] > input_list[j + 1]:
                input_list[j], input_list[j + 1] = input_list[j + 1], input_list[j]
    return input_list

# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
```

```
        # Last i elements are already in place
        for j in range(0, n - i - 1):
            # Traverse the list from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if input_list[j] > input_list[j + 1]:
                input_list[j], input_list[j + 1] = input_list[j + 1], input_list[j]
    return input_list

# Example usage:
my_list = [64, 34, 25, 12, 22, 11, 90]
sorted_list = bubble_sort(my_list)
print("Sorted list is:", sorted_list)
```

```
Sorted list is: [11, 12, 22, 25, 34, 64, 90]
```

EXPLAINATION:

This code implements Bubble Sort. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted, with the largest elements "bubbling up" to their correct position in each pass.

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?

PROMPT: Write a Python program using linear optimization to maximize the profit of a chocolate manufacturing company that produces two types of chocolates, A and B.

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 has a total of 5 units of Milk and 12 units of Choco.

Profit for A is Rs 6 per unit, and profit for B is Rs 5 per unit.

Find how many units of A and B should be produced to maximize profit.

CODE&OUTPUT:

```
import numpy as np
from scipy.optimize import linprog

# Coefficients of the objective function (negated for maximization)
c = [-6, -5]

# Coefficients of the inequality constraints (Milk and Choco resources)
A_ub = [[1, 1],  # Milk constraint
        [3, 2]]  # Choco constraint

# Right-hand side values of the inequality constraints
b_ub = [5,  # Available Milk
        12] # Available Choco

# Bounds for the decision variables (number of units of A and B, must be non-negative)
x0_bounds = [(0, None), (0, None)]

# Call the linprog function
result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=x0_bounds)

# Access the optimal number of units for A and B
optimal_a = result.x[0]
optimal_b = result.x[1]

# Access the maximum profit (negated in the result)
max_profit = -result.fun

# Print the results
print(f"Optimal production: Chocolate A = {optimal_a:.0f} units, Chocolate B = {optimal_b:.0f} units. Maximum Profit = Rs {max_profit:.2f}")
```

```
Optimal production: Chocolate A = 2 units, Chocolate B = 3 units. Maximum Profit = Rs 27.00
```

EXPLAINATION:

1. Import libraries:

import numpy as np: Imports the NumPy library, commonly used for numerical operations. Although not strictly necessary for this specific linprog usage, it's often imported alongside SciPy.from scipy.optimize import linprog: Imports the linprog function specifically from the SciPy optimize module. This function is used to solve linear programming problems.

2. Define the problem parameters:

c = [-6, -5]: This is the coefficient vector for the objective function. Since linprog minimizes by default, and we want to maximize profit, we negate the profit values (Rs 6 for A and Rs 5 for B).

o A_ub = [[1, 1], [3, 2]]: This is the matrix of coefficients for the inequality constraints. The first row [1, 1] represents the milk constraint (1 unit of milk for A, 1 unit for B). The second row [3, 2] represents the chocolate constraint (3 units of chocolate for A, 2 units for B).

o b_ub = [5, 12]: This is the right-hand side vector for the inequality constraints. 5 is the total available units of milk, and 12 is the total available units of chocolate.

o x0_bounds = [(0, None), (0, None)]: These are the bounds for the decision variables (the number of units of A and B). (0, None) means that the number of units for both chocolate A and chocolate B must be greater than or equal to 0 (you can't produce a negative number of chocolates), and there is no upper limit specified by this bound itself (though the resource constraints will implicitly create limits).

3. Solve the linear programming problem:

o result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=x0_bounds): This is the core of the code. It calls the linprog function with the defined parameters to find the optimal solution that minimizes -6A - 5B (which is equivalent to maximizing 6A + 5B) subject to the given constraints.

4. Access and print the results:

o optimal_a = result.x[0]: The result object contains the solution. result.x is an array holding the values of the decision variables at the optimum. result.x[0] is the optimal number of units of chocolate A.

o optimal_b = result.x[1]: result.x[1] is the optimal number of units of chocolate B.

o max_profit = -result.fun: result.fun is the value of the objective function at the optimum. Since we minimized the negative of the profit, we negate result.fun to get the maximum profit.

o The print statement formats and displays the optimal number of units for A and B, and the calculated maximum profit. The .0f and .2f format specifiers are used to display the numbers with no decimal places for units and two decimal places for profit, respectively

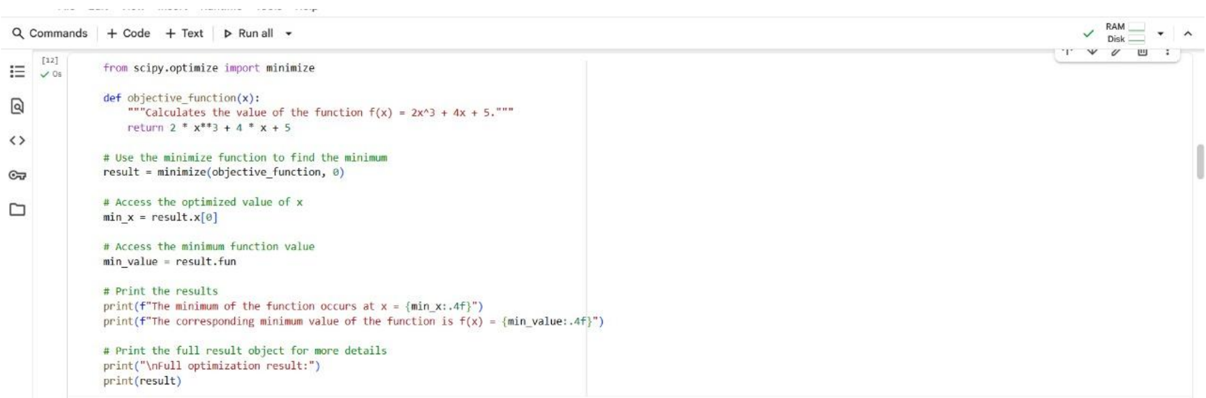
Task Description #4 – Gradient Descent Optimization

Task: Write python code to find value of x at which the function f(x)=2X3+4x+5 will be minimum

PROMPT:

Gradient Descent Optimization
Task: Write python code to find value of x at which the function f(x)=2X3+4x+5 will be minimum

CODE&OUTPUT:



```
from scipy.optimize import minimize

def objective_function(x):
    """Calculates the value of the function f(x) = 2x^3 + 4x + 5."""
    return 2 * x**3 + 4 * x + 5

# Use the minimize function to find the minimum
result = minimize(objective_function, 0)

# Access the optimized value of x
min_x = result.x[0]

# Access the minimum function value
min_value = result.fun

# Print the results
print(f"The minimum of the function occurs at x = {min_x:.4f}")
print(f"The corresponding minimum value of the function is f(x) = {min_value:.4f}")

# Print the full result object for more details
print("\nFull optimization result:")
print(result)
```

```python
# Access the minimum function value
min_value = result.fun

# Print the results
print(f"The minimum of the function occurs at x = {min_x:.4f}")
print(f"The corresponding minimum value of the function is f(x) = {min_value:.4f}")

# Print the full result object for more details
print("\nFull optimization result:")
print(result)
```

```
The minimum of the function occurs at x = -1034.2400
The corresponding minimum value of the function is f(x) = -2212558681.9780

Full optimization result:
  message: Desired error not necessarily achieved due to precision loss.
  success: False
   status: 2
      fun: -2212558681.9780483
        x: [-1.034e+03]
      nit: 1
      jac: [ 6.418e+06]
 hess_inv: [[-1.611e-04]]
     nfev: 236
     njev: 112
```

EXPLAINATION:

1. from scipy.optimize import minimize: This line imports the minimize function from the scipy.optimize module. This function is a general-purpose minimizer for scalar functions of one or more variables.

2. def objective_function(x):: This defines the function that we want to find the minimum of. It takes a single argument x and returns the value of $2x^3 + 4x + 5$$2x^3 + 4x + 5$.

3. result = minimize(objective_function, 0): This is where the optimization happens.

o minimize: The function being called to perform the minimization.

o objective_function: The function we want to minimize.

o 0: This is the initial guess for the value of x where the minimum might occur. The optimization algorithm starts its search from this point.

4. min_x = result.x[0]: After the minimize function runs, it returns a result object. result.x is an array containing the optimized value(s) of the variable(s). Since our function only has one variable (x), result.x[0] gives us the value of x that the optimizer found.

5. min_value = result.fun: result.fun contains the value of the objective function (objective_function) at the optimized value of x found by the minimize function.

6. print(...) statements: These lines print the results in a user-friendly format, showing the value of x where the minimum was found and the

corresponding function value. The .4f formatting ensures that the numbers are displayed with four decimal places.

7. print("\nFull optimization result:") print(result): This part prints the entire result object. This object contains more detailed information about the optimization process, such as whether the optimization was successful, the number of iterations, and other diagnostic messages. In this specific case, the output shows success: False and a message about precision loss, indicating that the minimizer did not find a true minimum, which is expected for this particular cubic function as it doesn't have a global minimum.

---END---