

.AI ASSISTED CODING.

LAB ASSIGNMENT -12.3

Program	: B-TECH
Specialization	: AIML
Name of Student	: AKULA MANOJ
Enrollment-No	: 2403A52031
Batch No	: 02
Date	: 22-10-2025

- 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

TASK-1: Write Python code for the `linear_search()` function to search a value in a list and extract its index.

PROMPT: Write Python code for the `linear_search()` function to search a value in a list and extract its index. It should give the index of Repeated values in the list also.

CODE GENERATED:

```
◆ Gemini

def linear_search(data, value):
    """
    Searches for all occurrences of a value in a list and returns their indices.

    Args:
        data: The list to search within.
        value: The value to search for.

    Returns:
        A list of indices where the value is found. Returns an empty list if not found.
    """
    indices = []
    for index, item in enumerate(data):
        if item == value:
            indices.append(index)
    return indices

# Get input from the user for the list
list_input = input("Enter the list elements separated by spaces: ")
my_list = [int(x) for x in list_input.split()] # Assuming the input are integers

# Get input from the user for the value to search
search_value = int(input("Enter the value to search for: "))

# Example usage:
indices = linear_search(my_list, search_value)

if indices:
    print(f"Value {search_value} found at indices: {indices}")
else:
    print(f"Value {search_value} not found in the list")
```

OUTPUT:

```
➔ Enter the list elements separated by spaces: 1 2 3 4 5 4
Enter the value to search for: 4
Value 4 found at indices: [3, 5]
```

TASK DESCRIPTION-2

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

PROMPT:

Generate an implementation of Bubble Sort and check the sorted output,take the input from the user.

GENERATED CODE:

```
def bubble_sort(data):
    """
    Sorts a list using the Bubble Sort algorithm.

    Args:
        data: The list to be sorted.

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

# Get input from the user for the list
list_input = input("Enter the list elements separated by spaces: ")
my_list = [int(x) for x in list_input.split()] # Assuming the input are integers

# Sort the list using bubble sort
sorted_list = bubble_sort(my_list)

# Display the sorted list
print("Sorted list:", sorted_list)
```

OUTPUT:

```
➔ Enter the list elements separated by spaces: 3 2 1 3 4 3 5
Sorted list: [1, 2, 3, 3, 3, 4, 5]
```

TASK DESCRIPTION-3

Task :Task: Write Python code to solve below case study below 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 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 chocolate. 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?

GENERATED CODE:

```
# Install PuLP if you haven't already
%pip install pulp

import pulp

# Define the problem
prob = pulp.LpProblem("Chocolate Manufacturing", pulp.LpMaximize)

# Get input from the user for resources and profit
try:
    milk_available = int(input("Enter the total units of Milk available: "))
    choco_available = int(input("Enter the total units of Chocolate available: "))
    profit_A = int(input("Enter the profit per unit of Chocolate A: "))
    profit_B = int(input("Enter the profit per unit of Chocolate B: "))
except ValueError:
    print("Invalid input. Please enter integer values for resources and profit.")
    exit()

# Define the decision variables
# x represents the number of units of chocolate A
# y represents the number of units of chocolate B
x = pulp.LpVariable("A", lowBound=0, cat='Integer')
y = pulp.LpVariable("B", lowBound=0, cat='Integer')

# Define the objective function
prob += profit_A * x + profit_B * y, "Total Profit"

# Define the constraints
# Milk constraint
prob += 1 * x + 1 * y <= milk_available, "Milk Constraint"
# Chocolate constraint
prob += 3 * x + 2 * y <= choco_available, "Chocolate Constraint"

# Solve the problem
prob.solve()

# Display the results
print("\n--- Results ---")
print("Status:", pulp.LpStatus[prob.status])
print(f"Optimal number of units of Chocolate A to produce: {x.varValue}")
print(f"Optimal number of units of Chocolate B to produce: {y.varValue}")
print(f"Maximum Total Profit: {pulp.value(prob.objective)}")
```

OUTPUT:

```
Collecting pulp
  Downloading pulp-3.3.0-py3-none-any.whl.metadata (8.4 kB)
  Downloading pulp-3.3.0-py3-none-any.whl (16.4 MB)
    ----- 16.4/16.4 MB 82.1 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-3.3.0
/usr/local/lib/python3.12/dist-packages/pulp/pulp.py:1489: UserWarning: Spaces are not permitted in the name. Converted to '_'
  warnings.warn("Spaces are not permitted in the name. Converted to '_'")
Enter the total units of Milk available: 23
Enter the total units of Chocolate available: 23
Enter the profit per unit of Chocolate A: 43
Enter the profit per unit of Chocolate B: 34

--- Results ---
Status: Optimal
Optimal number of units of Chocolate A to produce: 1.0
Optimal number of units of Chocolate B to produce: 10.0
Maximum Total Profit: 383.0
```

TASK DESCRIPTION-4

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

PROMPT:

GENERATED CODE:

```
import sympy

# Define the variable and the function
x = sympy.Symbol('x')
f = 2*x**3 + 4*x + 5

# Find the derivative of the function
f_prime = sympy.diff(f, x)

# Find the critical points by setting the derivative to zero
critical_points = sympy.solve(f_prime, x)

# Print the derivative and critical points
print("The derivative of the function is:", f_prime)
print("The critical points are:", critical_points)

# Analyze the second derivative to determine if it's a minimum or maximum
f_double_prime = sympy.diff(f_prime, x)
print("The second derivative of the function is:", f_double_prime)

# Evaluate the second derivative at each critical point if it's a real number
print("\nAnalyzing critical points:")
found_real_critical_points = False
for point in critical_points:
    if point.is_real:
        found_real_critical_points = True
        second_deriv_value = f_double_prime.subs(x, point)
        print(f"Second derivative at x = {point}: {second_deriv_value}")
        if second_deriv_value > 0:
            print(f"This is a local minimum at x = {point}")
        elif second_deriv_value < 0:
            print(f"This is a local maximum at x = {point}")
        else:
            print(f"The second derivative test is inconclusive at x = {point}")

if not found_real_critical_points:
    print("\nNo real critical points found. The function is always increasing.")
    print("For this cubic function with a positive leading coefficient, there is no local minimum or maximum.")
    print("The function approaches negative infinity as x approaches negative infinity.")
```

OUTPUT:

```
The derivative of the function is: 6*x**2 + 4
The critical points are: [-sqrt(6)*I/3, sqrt(6)*I/3]
The second derivative of the function is: 12*x

Analyzing critical points:

No real critical points found. The function is always increasing.
For this cubic function with a positive leading coefficient, there is no local minimum or maximum.
The function approaches negative infinity as x approaches negative infinity.
```

THANK YOU

