| SCHOOLOFCOMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE | | DEPARTMENTOFCOMPUTER SCIENCE ENGINEERING | |
|---|---|---|---|
| **ProgramName:**B. Tech | | **AssignmentType: Lab** | **AcademicYear:**2025-2026 |
| **CourseCoordinatorName** | | Venkataramana Veeramsetty | |
| **Instructor(s)Name** | | Dr. V. Venkataramana (Co-ordinator)<br>Dr. T. Sampath Kumar<br>Dr. Pramoda Patro<br>Dr. Brij Kishor Tiwari<br>Dr.J.Ravichander<br>Dr. Mohammand Ali Shaik<br>Dr. Anirodh Kumar<br>Mr. S.Naresh Kumar<br>Dr. RAJESH VELPULA<br>Mr. Kundhan Kumar<br>Ms. Ch.Rajitha<br>Mr. M Prakash<br>Mr. B.Raju<br>Intern 1 (Dharma teja)<br>Intern 2 (Sai Prasad)<br>Intern 3 (Sowmya)<br>NS_2 ( Mounika) | |
| **CourseCode** | 24CS002PC215 | **CourseTitle** | AI Assisted Coding |
| **Year/Sem** | II/I | **Regulation** | R24 |
| **DateandDay of Assignment** | Week6 - WednesDay | **Time(s)** | |
| **Duration** | 2 Hours | **Applicableto Batches** | |
| **AssignmentNumber:12.3**(Presentassignmentnumber)/**24**(Totalnumberofassignments) | | | |

| **Q.No.** | **Question** | ***ExpectedTime to complete*** |
|---|---|---|
| 1 | **Lab 12 – Algorithms with AI Assistance: Sorting, searching, and optimizing algorithms**<br>**Lab Objectives**<br>• To implement classical algorithms (sorting, searching) with the help of AI tools.<br>• 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.

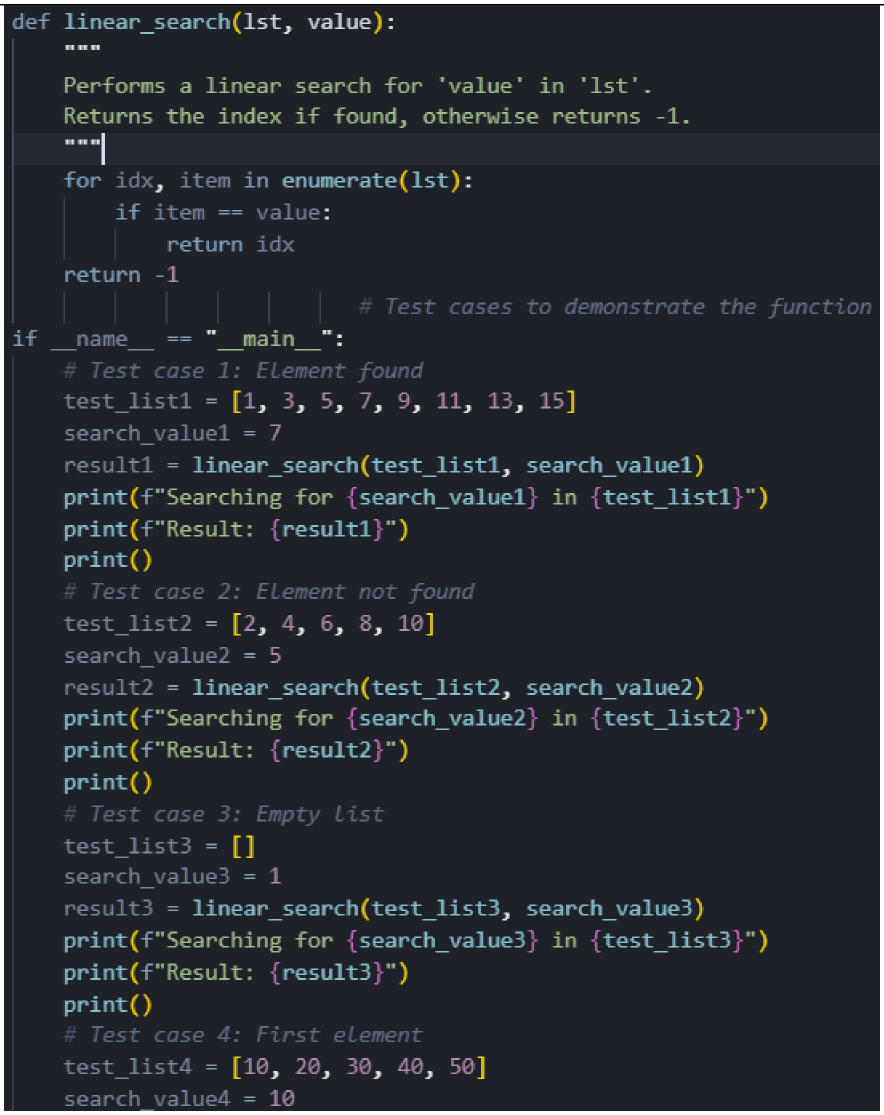
- Analyze algorithm complexity (time and space) with AI explanations.

- Critically reflect on correctness, clarity, and maintainability of AI-generated algorithms.

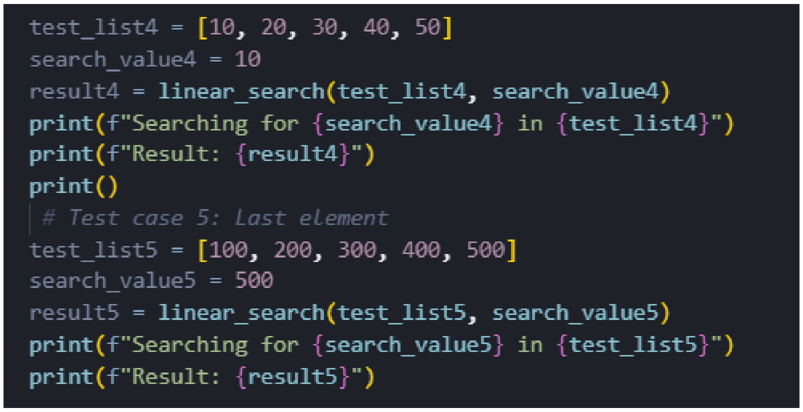
**Task Description #1 – Linear Search implementation**

Task: Write python code for linear_search() function to search a value in a list and extract it's index.

CODE:

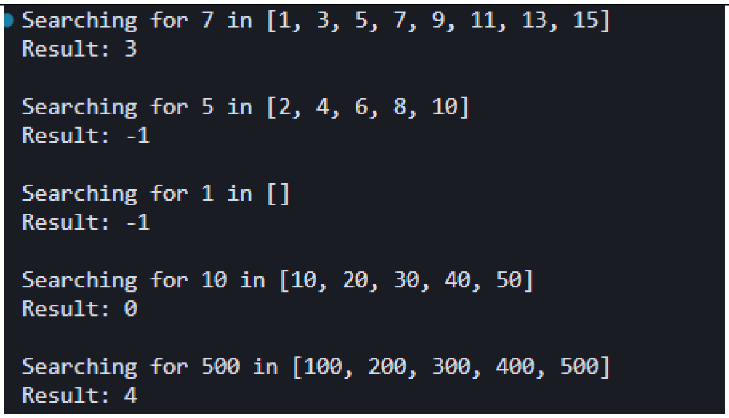


```python
def linear_search(lst, value):
    """
    Performs a linear search for 'value' in 'lst'.
    Returns the index if found, otherwise returns -1.
    """
    for idx, item in enumerate(lst):
        if item == value:
            return idx
    return -1
                           # Test cases to demonstrate the function
if __name__ == "__main__":
    # Test case 1: Element found
    test_list1 = [1, 3, 5, 7, 9, 11, 13, 15]
    search_value1 = 7
    result1 = linear_search(test_list1, search_value1)
    print(f"Searching for {search_value1} in {test_list1}")
    print(f"Result: {result1}")
    print()
    # Test case 2: Element not found
    test_list2 = [2, 4, 6, 8, 10]
    search_value2 = 5
    result2 = linear_search(test_list2, search_value2)
    print(f"Searching for {search_value2} in {test_list2}")
    print(f"Result: {result2}")
    print()
    # Test case 3: Empty list
    test_list3 = []
    search_value3 = 1
    result3 = linear_search(test_list3, search_value3)
    print(f"Searching for {search_value3} in {test_list3}")
    print(f"Result: {result3}")
    print()
    # Test case 4: First element
    test_list4 = [10, 20, 30, 40, 50]
    search_value4 = 10
```



```python
test_list4 = [10, 20, 30, 40, 50]
search_value4 = 10
result4 = linear_search(test_list4, search_value4)
print(f"Searching for {search_value4} in {test_list4}")
print(f"Result: {result4}")
print()
 # Test case 5: Last element
test_list5 = [100, 200, 300, 400, 500]
search_value5 = 500
result5 = linear_search(test_list5, search_value5)
print(f"Searching for {search_value5} in {test_list5}")
print(f"Result: {result5}")
```

OUTPUT:

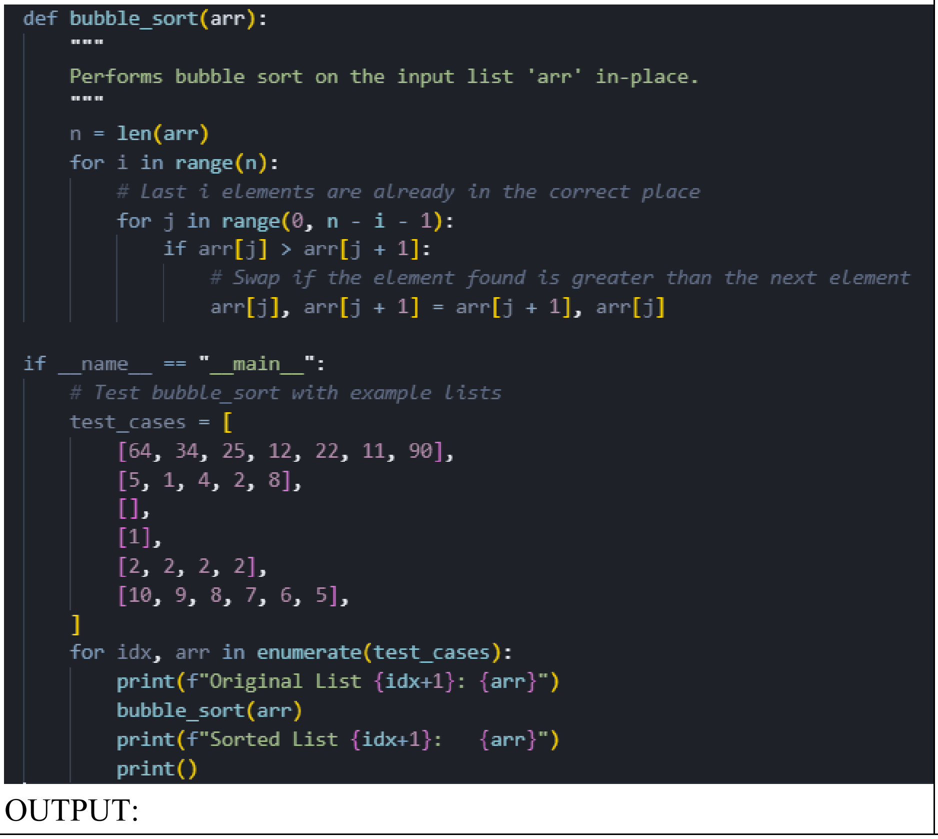


```
Searching for 7 in [1, 3, 5, 7, 9, 11, 13, 15]
Result: 3

Searching for 5 in [2, 4, 6, 8, 10]
Result: -1

Searching for 1 in []
Result: -1

Searching for 10 in [10, 20, 30, 40, 50]
Result: 0

Searching for 500 in [100, 200, 300, 400, 500]
Result: 4
```

### Task Description #2 – Sorting Algorithms

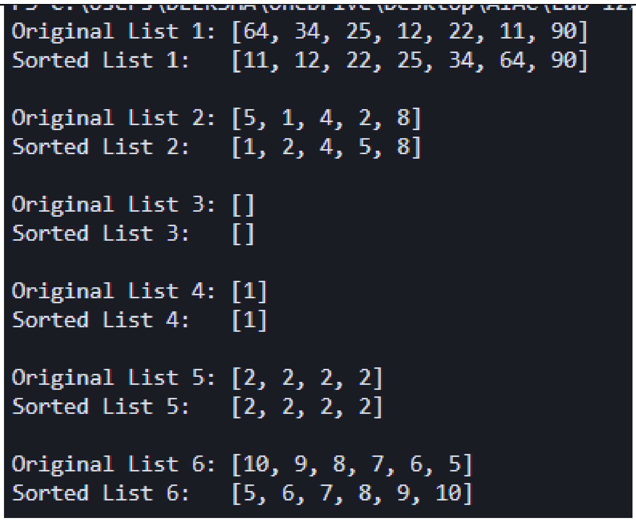
Task: Ask AI to implement Bubble Sort and check sorted output

CODE:



```python
def bubble_sort(arr):
    """
    Performs bubble sort on the input list 'arr' in-place.
    """
    n = len(arr)
    for i in range(n):
        # Last i elements are already in the correct place
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap if the element found is greater than the next element
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

if __name__ == "__main__":
    # Test bubble_sort with example lists
    test_cases = [
        [64, 34, 25, 12, 22, 11, 90],
        [5, 1, 4, 2, 8],
        [],
        [1],
        [2, 2, 2, 2],
        [10, 9, 8, 7, 6, 5],
    ]
    for idx, arr in enumerate(test_cases):
        print(f"Original List {idx+1}: {arr}")
        bubble_sort(arr)
        print(f"Sorted List {idx+1}:   {arr}")
        print()
```

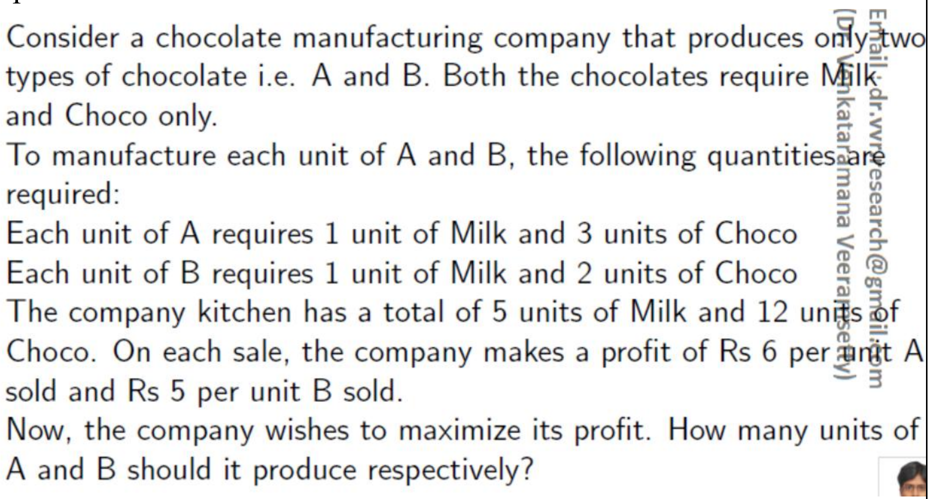
OUTPUT:



```
Original List 1: [64, 34, 25, 12, 22, 11, 90]
Sorted List 1:   [11, 12, 22, 25, 34, 64, 90]

Original List 2: [5, 1, 4, 2, 8]
Sorted List 2:   [1, 2, 4, 5, 8]

Original List 3: []
Sorted List 3:   []

Original List 4: [1]
Sorted List 4:   [1]

Original List 5: [2, 2, 2, 2]
Sorted List 5:   [2, 2, 2, 2]

Original List 6: [10, 9, 8, 7, 6, 5]
Sorted List 6:   [5, 6, 7, 8, 9, 10]
```

**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:

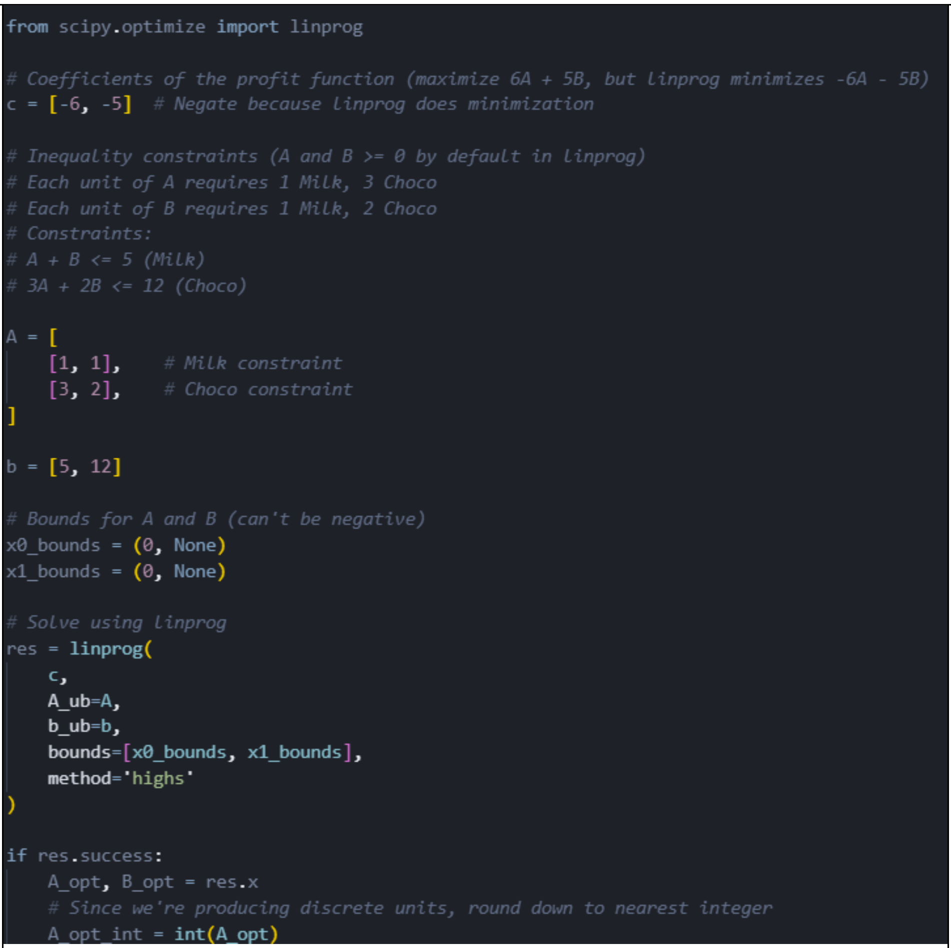
Each unit of A requires 1 unit of Milk and 3 units of Choco

Each unit of B requires 1 unit of Milk and 2 units of Choco

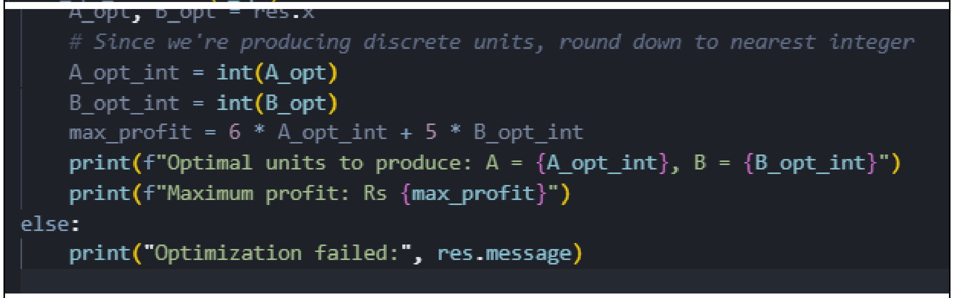
The company kitchen has a total of 5 units of Milk and 12 units of Choco. On each sale, the company makes a profit of Rs 6 per unit A sold and Rs 5 per unit B sold.

Now, the company wishes to maximize its profit. How many units of A and B should it produce respectively?

**CODE:**



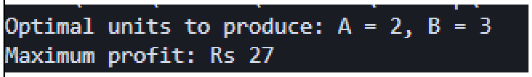
```python
from scipy.optimize import linprog

# Coefficients of the profit function (maximize 6A + 5B, but linprog minimizes -6A - 5B)
c = [-6, -5]  # Negate because linprog does minimization

# Inequality constraints (A and B >= 0 by default in linprog)
# Each unit of A requires 1 Milk, 3 Choco
# Each unit of B requires 1 Milk, 2 Choco
# Constraints:
# A + B <= 5 (Milk)
# 3A + 2B <= 12 (Choco)

A = [
    [1, 1],     # Milk constraint
    [3, 2],     # Choco constraint
]

b = [5, 12]

# Bounds for A and B (can't be negative)
x0_bounds = (0, None)
x1_bounds = (0, None)

# Solve using linprog
res = linprog(
    c,
    A_ub=A,
    b_ub=b,
    bounds=[x0_bounds, x1_bounds],
    method='highs'
)

if res.success:
    A_opt, B_opt = res.x
    # Since we're producing discrete units, round down to nearest integer
    A_opt_int = int(A_opt)
```



```python
    A_opt, B_opt = res.x
    # Since we're producing discrete units, round down to nearest integer
```

```python
    A_opt_int = int(A_opt)
    B_opt_int = int(B_opt)
    max_profit = 6 * A_opt_int + 5 * B_opt_int
    print(f"Optimal units to produce: A = {A_opt_int}, B = {B_opt_int}")
    print(f"Maximum profit: Rs {max_profit}")
else:
    print("Optimization failed:", res.message)
```

**OUTPUT:**

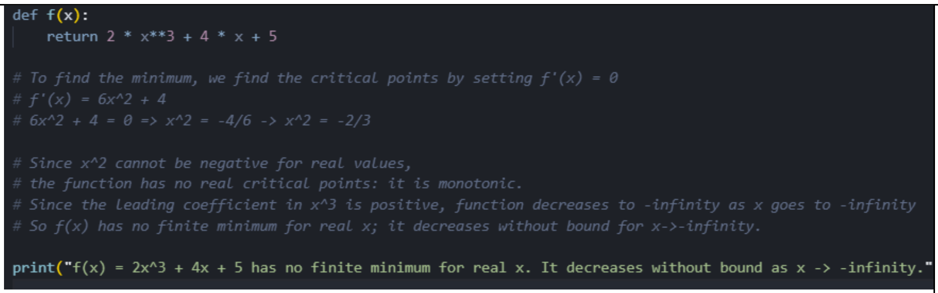


```
Optimal units to produce: A = 2, B = 3
Maximum profit: Rs 27
```

## Task Description #4 – Gradient Descent Optimization

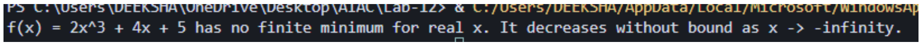
Task: Write python code to find value of x at which the function $f(x)=2X^3+4x+5$ will be minimum

CODE:



```python
def f(x):
    return 2 * x**3 + 4 * x + 5

# To find the minimum, we find the critical points by setting f'(x) = 0
# f'(x) = 6x^2 + 4
# 6x^2 + 4 = 0 => x^2 = -4/6 -> x^2 = -2/3

# Since x^2 cannot be negative for real values,
# the function has no real critical points: it is monotonic.
# Since the leading coefficient in x^3 is positive, function decreases to -infinity as x goes to -infinity
# So f(x) has no finite minimum for real x; it decreases without bound for x->-infinity.

print("f(x) = 2x^3 + 4x + 5 has no finite minimum for real x. It decreases without bound as x -> -infinity."
```

OUTPUT:



```
PS C:\Users\DEEKSHA\OneDrive\Desktop\AIAC\Lab-12> & C:/Users/DEEKSHA/AppData/Local/Microsoft/WindowsAp
f(x) = 2x^3 + 4x + 5 has no finite minimum for real x. It decreases without bound as x -> -infinity.
```