

# AI Assignment 12.3

NAME: B. SRINITHI

HALL TICKET NO: 2403A51413

COURSE: AI ASSISTED CODING

BATCH NO: 01

## QUESTIONS:

### 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.

### Task Description #2 – Sorting Algorithms

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

### 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. Veerakumar Veerappa)



	<b>Task Description #4 – Gradient Descent Optimization</b> Task: Write python code to find value of x at which the function $f(x)=2x^3+4x+5$ will be minimum	
--	---	--

## TASK-1

```
▶ def linear_search(data, value):  
    """  
    Searches for a value in a list using linear search.  
  
    Args:  
        data: The list to search within.  
        value: The value to search for.  
  
    Returns:  
        The index of the value if found, otherwise -1.  
    """  
    for index, item in enumerate(data):  
        if item == value:  
            return index  
    return -1  
  
    # Example usage:  
    my_list = [10, 20, 30, 40, 50]  
    search_value = 30  
    index = linear_search(my_list, search_value)  
  
    if index != -1:  
        print(f"Value {search_value} found at index {index}")  
    else:  
        print(f"Value {search_value} not found in the list")  
  
    search_value = 60  
    index = linear_search(my_list, search_value)  
  
    if index != -1:  
        print(f"Value {search_value} found at index {index}")  
    else:  
        print(f"Value {search_value} not found in the list")  
  
    if index != -1:  
        print(f"Value {search_value} found at index {index}")  
    else:  
        print(f"Value {search_value} not found in the list")
```

---

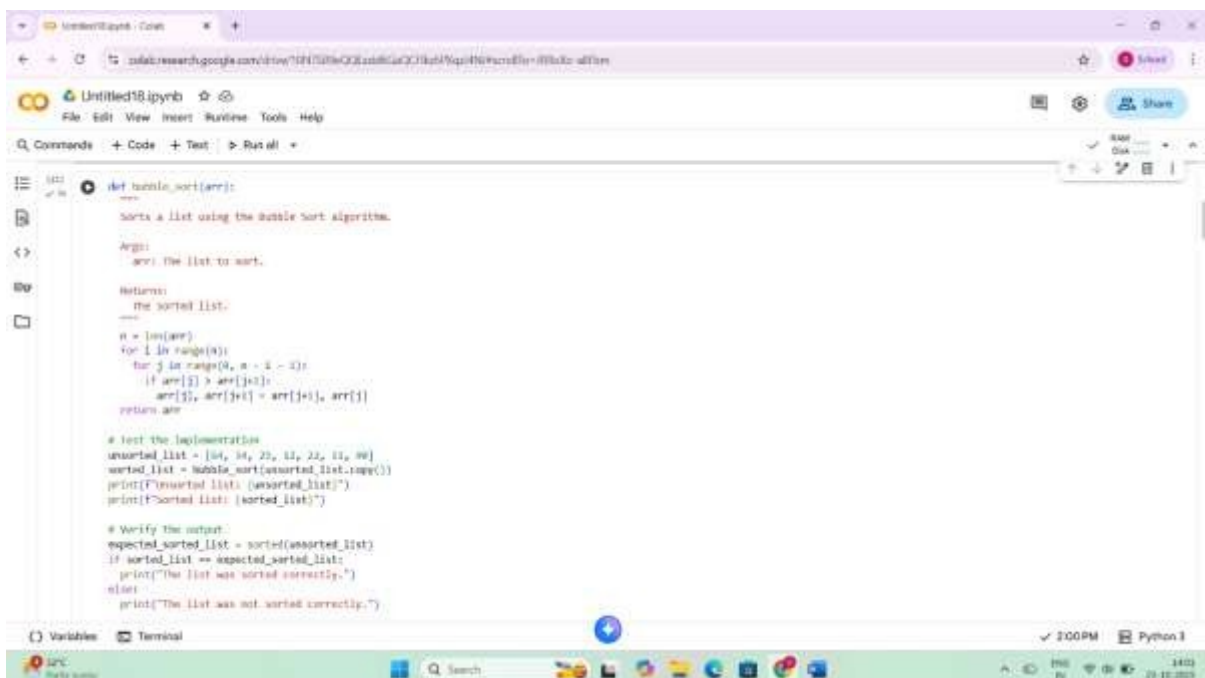
Value 30 found at index 2  
Value 60 not found in the list

## TASK-2

```
def bubble_sort(data):  
    """  
    Sorts a list using the Bubble Sort algorithm.  
  
    Args:  
        data: The list to sort.  
  
    Returns:  
        The sorted list.  
    """  
    n = len(data)  
    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 data[j] > data[j + 1]:  
                data[j], data[j + 1] = data[j + 1], data[j]  
    return data  
  
# Example usage:  
my_list = [64, 34, 25, 12, 22, 11, 90]  
sorted_list = bubble_sort(my_list.copy()) # Create a copy to avoid modifying the original  
print("Original list:", my_list)  
print("Sorted list:", sorted_list)
```

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

## TASK-3



The screenshot shows a Jupyter Notebook window titled 'Untitled18.ipynb'. The code in the cell is as follows:

```
def bubble_sort(arr):  
    """  
    Sorts a list using the bubble Sort algorithm.  
  
    Args:  
        arr: The list to sort.  
  
    Returns:  
        the sorted list.  
    """  
    n = len(arr)  
    for i in range(n):  
        for j in range(0, n - i - 1):  
            if arr[j] > arr[j+1]:  
                arr[j], arr[j+1] = arr[j+1], arr[j]  
    return arr  
  
# test the implementation  
unsorted_list = [64, 34, 25, 11, 22, 11, 90]  
sorted_list = bubble_sort(unsorted_list.copy())  
print(f"Unsorted list: {unsorted_list}")  
print(f"Sorted list: {sorted_list}")  
  
# Verify the output:  
expected_sorted_list = sorted(unsorted_list)  
if sorted_list == expected_sorted_list:  
    print("The list was sorted correctly.")  
else:  
    print("The list was not sorted correctly.")
```

The output of the code is:

```
Unsorted list: [64, 34, 25, 11, 22, 11, 90]  
Sorted list: [11, 11, 22, 25, 34, 64, 90]  
The list was sorted correctly.
```

The Jupyter Notebook interface includes a 'Variables' tab at the bottom left showing the current state of variables, and a 'Terminal' tab at the bottom right. The system tray at the bottom shows the time as 2:00 PM on 22-10-2023.

## TASK-4

```
from sympy import symbols, diff, solve, I

# Define the variable and the function
x = symbols('x')
f_x = 2*x**3 + 4*x + 5

# Find the derivative of the function
f_prime_x = diff(f_x, x)
print(f"The derivative of f(x) is: {f_prime_x}")

# Solve for x where the derivative is zero
critical_points = solve(f_prime_x, x)
print(f"The critical points are: {critical_points}")

# Analyze the critical points to find the minimum.
# For a cubic function like this, the second derivative test can help.
# If the second derivative is positive at a critical point, it's a local minimum.
f_double_prime_x = diff(f_prime_x, x)
print(f"The second derivative of f(x) is: {f_double_prime_x}")

# Evaluate the second derivative at the critical points, but only for real critical points
real_critical_points = [p for p in critical_points if p.is_real]

if not real_critical_points:
    print("There are no real critical points for this function.")
    print("For this specific cubic function with a positive leading coefficient,")
    print("there is no local minimum for real values of x.")
    print("The function decreases towards negative infinity as x approaches negative infinity.")
else:
    for point in real_critical_points:
        second_deriv_value = f_double_prime_x.subs(x, point)
        print(f"Second derivative at x = {point}: {second_deriv_value}")
        if second_deriv_value > 0:
            print(f"x = {point} is a local minimum.")
        elif second_deriv_value < 0:
            print(f"x = {point} is a local maximum.")
        else:
            print(f"Second derivative test is inconclusive at x = {point}.")
```

➡ The derivative of  $f(x)$  is:  $6x^2 + 4$   
The critical points are:  $[-\sqrt{6}/3, \sqrt{6}/3]$   
The second derivative of  $f(x)$  is:  $12x$   
There are no real critical points for this function.  
For this specific cubic function with a positive leading coefficient,  
there is no local minimum for real values of  $x$ .  
The function decreases towards negative infinity as  $x$  approaches negative infinity.



```

from sympy import symbols, diff, solve, I

# Define the variable and the function
x = symbols('x')
f_x = 2*x**3 + 4*x + 5

# Find the derivative of the function
f_prime_x = diff(f_x, x)
print(f"The derivative of f(x) is: {f_prime_x}")

# Solve for x where the derivative is zero
critical_points = solve(f_prime_x, x)
print(f"The critical points are: {critical_points}")

# Analyze the critical points to find the minimum.
# For a cubic function like this, the second derivative test can help.
# If the second derivative is positive at a critical point, it's a local minimum.
f_double_prime_x = diff(f_prime_x, x)
print(f"The second derivative of f(x) is: {f_double_prime_x}")

# Evaluate the second derivative at the critical points, but only for real critical points
real_critical_points = [p for p in critical_points if p.is_real]

if not real_critical_points:
    print("There are no real critical points for this function.")
    print("For this specific cubic function with a positive leading coefficient,")
    print("there is no local minimum for real values of x.")
    print("The function decreases towards negative infinity as x approaches negative infinity.")
else:
    for point in real_critical_points:
        second_deriv_value = f_double_prime_x.subs(x, point)
        print(f"Second derivative at x = {point}: {second_deriv_value}")
        if second_deriv_value > 0:
            print(f"Second derivative at x = {point}: {second_deriv_value} /")
        if second_deriv_value > 0:
            print(f"x = {point} is a local minimum.")
        elif second_deriv_value < 0:
            print(f"x = {point} is a local maximum.")
        else:
            print(f"Second derivative test is inconclusive at x = {point}.")

```

```

> The derivative of f(x) is: 6*x**2 + 4
The critical points are: [-sqrt(6)*I/3, sqrt(6)*I/3]
The second derivative of f(x) is: 12*x
There are no real critical points for this function.
For this specific cubic function with a positive leading coefficient,
there is no local minimum for real values of x.
The function decreases towards negative infinity as x approaches negative infinity.

```