SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty		
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week6 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:12.1(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:** • Apply AI-assisted programming to implement and optimize sorting and searching algorithms. • Compare different algorithms in terms of efficiency and use	Week6 - Monday

cases.

- Understand how AI tools can suggest optimized code and complexity improvements.

Task Description #1 (Sorting – Merge Sort Implementation)

- Task: Use AI to generate a Python program that implements the Merge Sort algorithm.
- Instructions:
  - Prompt AI to create a function merge_sort(arr) that sorts a list in ascending order.
  - Ask AI to include time complexity and space complexity in the function docstring.
  - Verify the generated code with test cases.
- Expected Output:
  - A functional Python script implementing Merge Sort with proper documentation.

Prompt: Generate a Python program that implements the Merge Sort algorithm. The program should prompt the user to enter a list of numbers, separated by spaces. The program should then sort the list in ascending order and print the sorted list. Also, include a docstring for the merge_sort function that explains its time and space complexity.

```
def merge_sort(arr):
    """Sorts a list using the Merge Sort algorithm.
    Args:
        arr: The list to be sorted.
    Returns:
        The sorted list.
    Time Complexity: O(n log n) - The algorithm divides the list in half repeatedly (log n divisions) and merging takes linear time (n).
    Space Complexity: O(n) - Additional space is required to store the temporary subarrays during the merge process.
    """
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    return merge(left_half, right_half)

def merge(left, right):
    """Merges two sorted lists into a single sorted list."""
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result

# Get input from the user
input_str = input("Enter a list of numbers separated by spaces: ")
try:
    numbers = [int(x) for x in input_str.split()]
    sorted_numbers = merge_sort(numbers)
    print("Sorted list:", sorted_numbers)
except ValueError:
    print("Invalid input. Please enter numbers separated by spaces.")
```

```
Enter a list of numbers separated by spaces: 1 2 6  8 4 5
Sorted list: [1, 2, 4, 5, 6, 8]
```

Task Description #2 (Searching – Binary Search with AI Optimization)

- Task: Use AI to create a binary search function that finds a target element in a sorted list.
- Instructions:
  - Prompt AI to create a function binary_search(arr, target) returning the index of the target or -1 if not found.
  - Include docstrings explaining best, average, and worst-case complexities.
  - Test with various inputs.
- Expected Output:
  - Python code implementing binary search with AI-generated comments and docstrings.

Prompt: Generate a Python program with a binary_search function that finds a target number in a sorted list. The program should ask the user to input a series of numbers (separated by spaces) for the list and then ask for the target number to search for. The binary_search function should return the index of the target if found, and -1 otherwise. Include a docstring for the function explaining its best, average, and worst-case time complexities.

```
def binary_search(arr, target):
    """Finds the index of a target number in a sorted list using Binary Search.

    Args:
        arr: The sorted list to search within.
        target: The number to search for.

    Returns:
        The index of the target if found, otherwise -1.

    Time Complexity:
        Best Case: O(1) - The target is found at the middle of the list in the first comparison.
        Average Case: O(log n) - The list is repeatedly halved until the target is found or the search space is empty.
        Worst Case: O(log n) - The target is at either end of the list or not present, requiring the maximum number of comparisons.
    """
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1

# Get input from the user for the sorted list
input_list_str = input("Enter a sorted list of numbers separated by spaces: ")
try:
    sorted_numbers = [int(x) for x in input_list_str.split()]
```

```
        # Get input from the user for the target number
        target_str = input("Enter the target number to search for: ")
        target_number = int(target_str)

        index = binary_search(sorted_numbers, target_number)

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

    except ValueError:
        print("Invalid input. Please enter numbers separated by spaces for the list and a valid number for the target.")
```

```
Enter a sorted list of numbers separated by spaces: 1 2 4 7 8
Enter the target number to search for: 4
Target number 4 found at index 2.
```

Task Description #3 (Real-Time Application – Inventory Management System)

- Scenario: A retail store's inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:
    1. Quickly search for a product by ID or name.
    2. Sort products by price or quantity for stock analysis.
- Task:
    - Use AI to suggest the most efficient search and sort algorithms for this use case.
    - Implement the recommended algorithms in Python.
    - Justify the choice based on dataset size, update frequency, and performance requirements.
- Expected Output:
    - A table mapping operation → recommended algorithm → justification.
    - Working Python functions for searching and sorting the inventory.

Prompt: Genarate a Python program for a basic inventory system. The program should ask the user to input a list of products, each with a product ID, name, and price. Then, give the user two options: search for a product by its ID or sort the products by either price or ID. Make sure the code includes comments explaining which algorithms are best for searching and sorting a large dataset and why.

```python
class Product:
    """Represents a product with a product ID, name, and price."""

    def __init__(self, product_id, name, price):
        """Initializes a new Product instance.

        Args:
            product_id: The unique identifier for the product.
            name: The name of the product.
            price: The price of the product.
        """
        self.product_id = product_id
        self.name = name
        self.price = price

    def __str__(self):
        """Returns a string representation of the product."""
        return f"Product ID: {self.product_id}, Name: {self.name}, Price: ${self.price:.2f}"

def search_product_by_id(inventory, target_id):
    """Searches for a product by its ID in the inventory list.

    Args:
        inventory: A list of Product objects.
        target_id: The product ID to search for.

    Returns:
        The Product object if found, otherwise None.

    Note: For a large dataset, a more efficient approach like using a hash table
    (dictionary in Python) for O(1) average time complexity or binary search
    (if the list is sorted by ID) for O(log n) time complexity would be better
    than this linear search (O(n)).
    """
    for product in inventory:
        if product.product_id == target_id:
            return product
    return None
```

```python
def sort_products_by_price(inventory):
    """Sorts the inventory list in place by product price.

    Args:
        inventory: A list of Product objects.

    Note: For large datasets, efficient sorting algorithms like Merge Sort or
    Quick Sort offer better time complexity (O(n log n) on average) compared
    to simpler algorithms. Python's list.sort() typically uses Timsort, which
    is efficient for various data types and often performs well.
    """
    # Sort the inventory list in place using the 'price' attribute as the key
    inventory.sort(key=lambda product: product.price)

def sort_products_by_id(inventory):
    """Sorts the inventory list in place by product ID.

    Args:
        inventory: A list of Product objects.

    Note: For large datasets, efficient sorting algorithms like Merge Sort or
    Quick Sort offer better time complexity (O(n log n) on average).
    Python's list.sort() typically uses Timsort, which is efficient for various
    data types and often performs well.
    """
    # Sort the inventory list in place using the 'product_id' attribute as the key
    inventory.sort(key=lambda product: product.product_id)

inventory = []
```

```
print("Enter product details:")
while True:
    try:
        product_id_str = input("Enter product ID: ")
        product_id = int(product_id_str)

        name = input("Enter product name: ")

        price_str = input("Enter product price: ")
        price = float(price_str)

        product = Product(product_id, name, price)
        inventory.append(product)

        add_another = input("Add another product? (yes/no): ").lower()
        if add_another in ['no', 'n']:
            break

    except ValueError:
        print("Invalid input. Please enter valid numbers for product ID and price.")

print(f"Added {len(inventory)} products to the inventory.")

while True:
    print("\nInventory System Options:")
    print("1. Search by product ID")
    print("2. Sort by price")
    print("3. Sort by ID")
    print("4. Exit")

    choice = input("Enter your choice: ")
```

```
if choice == '1':
    try:
        target_id_str = input("Enter the product ID to search for: ")
        target_id = int(target_id_str)
        found_product = search_product_by_id(inventory, target_id)
        if found_product:
            print("Product Found:", found_product)
        else:
            print(f"Product with ID {target_id} not found.")
    except ValueError:
        print("Invalid input. Please enter a valid integer for the product ID.")
elif choice == '2':
    sort_products_by_price(inventory)
    print("Inventory sorted by price:")
    for product in inventory:
        print(product)
elif choice == '3':
    sort_products_by_id(inventory)
    print("Inventory sorted by ID:")
    for product in inventory:
        print(product)
elif choice == '4':
    print("Exiting Inventory System.")
    break
else:
    print("Invalid choice. Please enter a number between 1 and 4.")
```

```
Enter product details:
Enter product ID: 4
Enter product name: laptop
Enter product price: 75000
Add another product? (yes/no): yes
Enter product ID: 2
Enter product name: mobile
Enter product price: 20000
Add another product? (yes/no): yes
Enter product ID: 1
Enter product name: mouse
Enter product price: 1500
Add another product? (yes/no): yes
Enter product ID: 3
Enter product name: tablet
Enter product price: 45000
Add another product? (yes/no): no
Added 4 products to the inventory.

Inventory System Options:
1. Search by product ID
2. Sort by price
3. Sort by ID
4. Exit
Enter your choice: 1
Enter the product ID to search for: 3
Product Found: Product ID: 3, Name: tablet, Price: $45000.00

Inventory System Options:
1. Search by product ID
2. Sort by price
3. Sort by ID
4. Exit
Enter your choice: 2
Inventory sorted by price:
Product ID: 1, Name: mouse, Price: $1500.00
Product ID: 2, Name: mobile, Price: $20000.00
Product ID: 3, Name: tablet, Price: $45000.00
Product ID: 4, Name: laptop, Price: $75000.00

  Inventory System Options:
  1. Search by product ID
  2. Sort by price
  3. Sort by ID
  4. Exit
  Enter your choice: 3
  Inventory sorted by ID:
  Product ID: 1, Name: mouse, Price: $1500.00
  Product ID: 2, Name: mobile, Price: $20000.00
  Product ID: 3, Name: tablet, Price: $45000.00
  Product ID: 4, Name: laptop, Price: $75000.00

  Inventory System Options:
  1. Search by product ID
  2. Sort by price
  3. Sort by ID
  4. Exit
  Enter your choice: 4
  Exiting Inventory System.
```

✅ Deliverables (For All Tasks)

1.AI-generated prompts for code and test case generation.

2.At least 3 assert test cases for each task.

3.AI-generated initial code and execution screenshots.

4.Analysis of whether code passes all tests.

5.Improved final version with inline comments and explanation.

6.Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.
