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:

create a Python function that sorts a list in ascending order using the Merge Sort algorithm.
Include a docstring with time and space complexities and allow the user to input a list of numbers to sort

CODE:

```
[1]
13s
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
    return arr
nums = list(map(int, input("Enter numbers to sort (space separated): ").split()))
print("Sorted List:", merge_sort(nums))
```

OUTPUT:

```
Enter numbers to sort (space separated): 1 3 9 6 4 5 7 2 8
Sorted List: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

CODE EXPLATION:

1. User enters a list of integers separated by spaces.
2. The list is divided recursively into halves.
3. Each half is sorted using recursive calls.
4. Sorted halves are merged into a final sorted list.
5. Temporary arrays cause space complexity.

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:

write a function binary_search(arr, target) to find a user-specified target in a sorted list include best average and worst-case allow the user to input the sorted list and the target value

CODE:

```
def binary_search(arr, target):
    low, high = 0, 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
arr = list(map(int, input("Enter a sorted list of numbers (space separated): ").split()))
target = int(input("Enter the number to search for: "))
index = binary_search(arr, target)
if index != -1:
    print(f"Target {target} found at index {index}")
else:
    print(f"Target {target} not found in the list")
```

```
Enter a sorted list of numbers (space separated): 2 3 6 4 3 2
Enter the number to search for: 3
Target 3 found at index 1
```

CODE EXPLATION:

1. User provides a sorted list and a target value.
2. The list is divided repeatedly to locate the target.
3. Low and high pointers define the search range.
4. Complexity is log n for average/worst cases.
5. Uses additional space

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:

create a Python inventory management systemthe user can Add products (ID, name, price, quantity) search for a product by ID Sort the inventory by price or quantity Suggest the best search and sort algorithms for large datasets

CODE:

```python
def add_product():
    pid = int(input("Enter product ID: "))
    name = input("Enter product name: ")
    price = float(input("Enter product price: "))
    quantity = int(input("Enter product quantity: "))
    inventory.append({"id": pid, "name": name, "price": price, "quantity": quantity})
def search_by_id(product_id):
    for product in inventory:
        if product["id"] == product_id:
            return product
    return None
def sort_by_price():
    return sorted(inventory, key=lambda x: x["price"])
def sort_by_quantity():
    return sorted(inventory, key=lambda x: x["quantity"], reverse=True)
inventory = []
while True:
    print("\nInventory Menu")
    print("1. Add Product")
    print("2. Search by ID")
    print("3. Sort by Price")
    print("4. Sort by Quantity")
    print("5. Exit")
    choice = input("Enter choice: ")
    if choice == "1":
        add_product()
    elif choice == "2":
        pid = int(input("Enter product ID to search: "))
        result = search_by_id(pid)
        print(result if result else "Product not found.")
    elif choice == "3":
        print("Sorted by Price:", sort_by_price())
    elif choice == "4":
        print("Sorted by Quantity:", sort_by_quantity())
    elif choice == "5":
        break
    else:
        print("Invalid choice!")
```

OUTPUT:

```
1. Add Product
2. Search by ID
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice: 1
Enter product ID: 101
Enter product name: laptop
Enter product price: 50000
Enter product quantity: 3

Inventory Menu
1. Add Product
2. Search by ID
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice: 2
Enter product ID to search: 101
{'id': 101, 'name': 'laptop', 'price': 50000.0, 'quantity': 3}

Inventory Menu
1. Add Product
2. Search by ID
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice: 3
Sorted by Price: [{'id': 101, 'name': 'laptop', 'price': 50000.0, 'quantity': 3}]

Inventory Menu
1. Add Product
2. Search by ID
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice: 3
Sorted by Price: [{'id': 101, 'name': 'laptop', 'price': 50000.0, 'quantity': 3}]

Inventory Menu
1. Add Product
2. Search by ID
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice: exit
Invalid choice!

Inventory Menu
1. Add Product
2. Search by ID
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice: 5
```

CODE EXPLATION:

1. User can add multiple products with details like ID, name, price, and quantity
2. Search by ID uses a simple linear search for clarity HashMap recommended for large datasets
3. Sorting uses Python for efficiency (merge sort hybrid).
4. Sorting can be done by price or quantity
5. Menu-driven interface allows repeated operations

✅ 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.	