|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 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 program that implements the Merge Sort algorithm. The function merge\_sort sorts a list in ascending order and includes documentation on its time and space complexit with user input.      **Observation :** The code correctly implements the Merge Sort algorithm, including the recursive division and merging of sorted halves the merge\_sort function includes a doc string that explains its purpose, arguments, return value, and correctly states the Time Complexity.the output shows the original list entered by the user ([2,4,1,3,6,5]) and the sorted list ([1,2,3,4,5,6]). This confirms that the merge\_sort function is working correctly for the given input.  **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 doc strings.   **Prompt :** Generate a Python program that implements the Binary Search algorithm. The function binary\_search finds the index of a target element in a sorted list, returning -1 if the element is not found. The function includes documentation on its time complexities with user input.      **Observation :** The binary\_search function correctly implements the binary search algorithm. It uses a while loop to repeatedly divide the search interval in half, adjusting the left and right pointers based on the comparison of the middle element with the target.The function has a comprehensive doc string that clearly explains.  **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:   1. Use AI to suggest the most efficient search and sort algorithms for this use case.   2. Implement the recommended algorithms in Python.   3. Justify the choice based on dataset size, update frequency, and performance requirements. * Expected Output:   1. A table mapping operation → recommended algorithm → justification.   2. Working Python functions for searching and sorting the inventory.   **Prompt :** Generate a Python program lets you manage a list of products. You can add new products, search for them by name or ID, and sort them by price or stock. It runs in the terminal and helps practice basic Python skills like classes, input, and sorting. Great for small inventory tracking or learning purposes with user input.      **Observation :** The program works well for managing a simple inventory through the terminal. It successfully adds, searches, and sorts products using user input. The use of a Product class makes the code organized and easy to understand. However, the inventory data is not saved permanently, so all data is lost when the program ends.  ✅ 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.   Top of Form | | | | | | Week6 - Monday |  |