|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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) | | | | | | | | | |
| **NAME: P.HEMAN ROLL NO: 2403A510F5**  **BATCH: 06 DATE: 22-09-2025** | | | | | | | | | |
|  | **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 simple Python program that implements the Merge Sort algorithm. Create a function merge\_sort(arr) that sorts a list in ascending order. Include time and space complexity in the function docstring. Add a few test cases to verify that the function works correctly."  CODE:    OUTPUT:      OBSERVATION:   1. The **test cases pass**, showing that the merge\_sort function correctly sorts arrays of different sizes and types (including empty and single-element lists). 2. The function works **recursively**, splitting the list and merging sorted halves. 3. The **time complexity** is O(n log n) and **space complexity** is O(n), as noted in the docstring. 4. The original arrays are not modified; the function returns **new sorted lists**   **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 that implements a binary search function. The function should be named binary\_search(arr, target) and return the index of the target in a sorted list, or -1 if not found. Include a test\_binary\_search() function that runs multiple test cases to verify correctness. Also, allow the user to input a sorted list and a target number to search interactively. Ensure the script handles invalid input gracefully."  CODE:    OUTPUT:    OBSERVATION:  Here’s a **simple observation** for your binary search script:   1. The **test cases all passed**, showing that binary\_search correctly finds the index of existing elements and returns -1 for missing elements. 2. The **interactive input works**, allowing the user to enter a sorted list and target number. 3. The function handles **invalid input** and **empty lists** gracefully. 4. The search is **fast (O(log n))** for large sorted lists.   This shows the program works correctly for both automated tests and user input.  **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:  "Create a Python program to manage a store inventory. Each product has an ID, name, price, and quantity. Implement the following features:   1. Search a product by ID using binary search (assume inventory is sorted by ID). 2. Search a product by name using a dictionary lookup. 3. Sort the inventory by price or quantity. Demonstrate all functions with example products and print the results."   CODE:    OUTPUT:    OBSERVATION:   1. **Binary search by ID** correctly finds the product with id=103 (Orange). 2. **Dictionary lookup by name** successfully finds the product "Mango". 3. **Sorting by price** returns products in ascending order of price. 4. **Sorting by quantity (descending)** correctly orders products with the highest stock first. 5. The program works efficiently for small inventories and demonstrates fast search and sort operations.   This shows that both search and sort functions are working as intended.  ✅ 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 |  |