DSA Project Report

Project title: **Smart Project Scheduler**

Submitted by:

Usman Hassan	**241495**
M.Jarrar Hassan	**241583**
Arshman Abaas	**241567**
Laiba Malik	**241523**

Submitted to: **Mam Mehmoona Jabeen**

TABLE OF CONTENTS

1. Abstract

This project implements a comprehensive management system designed to handle three critical aspects of organizational administration: Employee records, Project allocation, and Task scheduling. The system utilizes core Data Structures including Dynamic Arrays (Vectors) for linear record keeping, Maps (Red-Black Trees) for efficient project retrieval, and Priority Queues (Heaps) for intelligent task scheduling. The application ensures data persistence using CSV file handling, allowing for state preservation between sessions.

2. Introduction

2.1 Problem Statement

In modern software management, handling resources efficiently is critical. Organizations face challenges in:

1. Storing and retrieving employee details efficiently.
2. Assigning multiple employees to specific projects without data redundancy.
3. Processing tasks not just by arrival time (FIFO), but by priority (urgency).

2.2 Project Scope

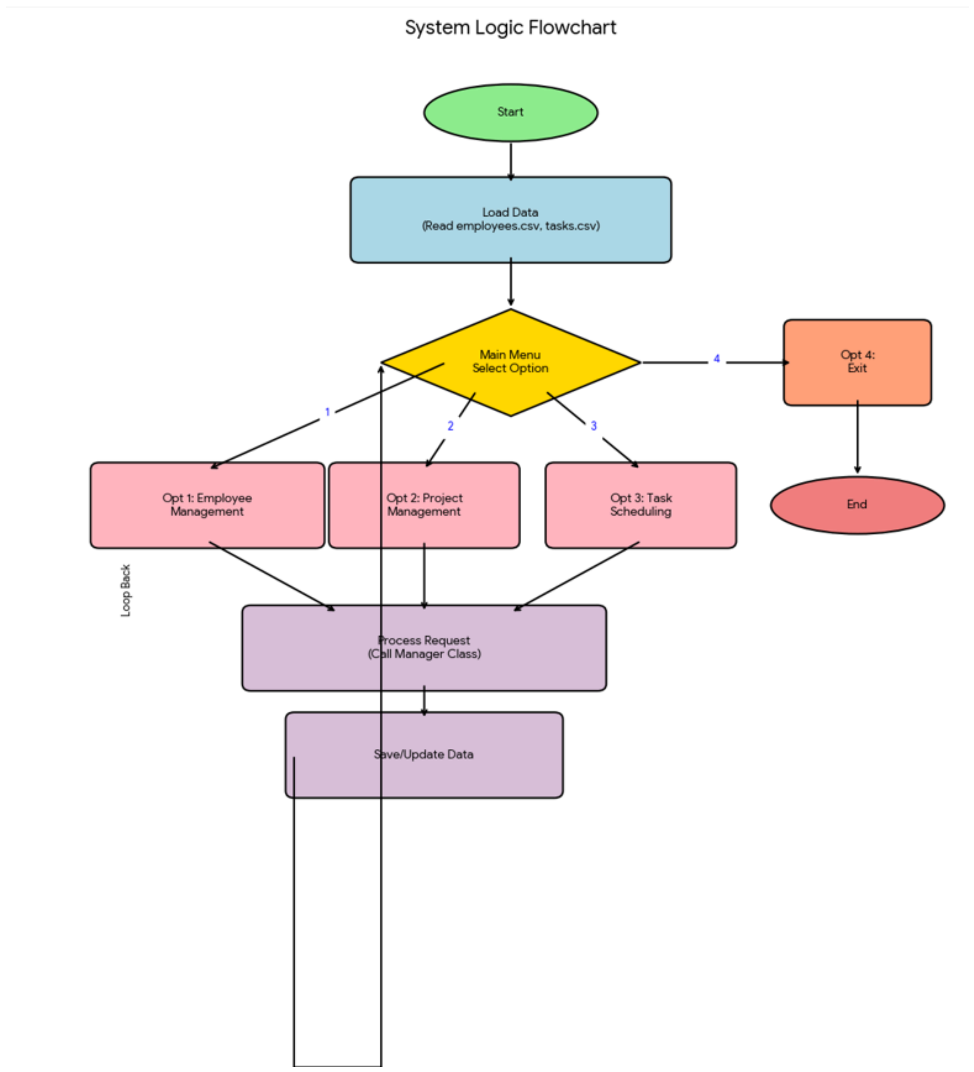
The application is a console-based C++ program that serves as a central dashboard. It allows administrators to:

- Import and export employee data.
- Rank employees based on performance scores.
- Map projects to employees using unique identifiers.
- Schedule tasks where high-priority items are processed before low-priority ones.

3. System Architecture & Flow

The system is modular, divided into three distinct managers controlled by a central driver (`main.cpp`).

3.1 Flowchart Overview



4. Data Structures and Algorithms Analysis

This section analyzes the specific DSA concepts applied in the code.

4.1. `std::vector` (Employee Management)

- Usage: Used in `EmployeeManager.h` to store the list of `Employee` structs.
- Reason for Selection: Vectors provide contiguous memory allocation. Since we often need to iterate through *all* employees to display them or save them to a file, the cache locality of a vector provides excellent performance.
- Time Complexity:
  - Access by Index: O(1)
  - Insertion at end (`push_back`): Amortized O(1)
  - Search (Linear): O(N)

4.2. `std::map` (Project Management)

- Usage: Used in `ProjectManager.h`.
- Reason for Selection: A Map is an associative container (Key-Value pair). We use the `Project ID` as the key. This allows us to find a specific project instantly without searching through the whole list. Internally, C++ Maps are typically implemented as Red-Black Trees (Self-Balancing Binary Search Trees).
- Time Complexity:
  - Search/Lookup: O(logN)
  - Insertion: O(logN)

4.3. `std::priority_queue` (Task Scheduling)

- Usage: Used in `TaskScheduler.h`.
- Reason for Selection: This is the core algorithmic feature of the project. A standard queue follows FIFO (First In, First Out). However, real-world tasks have urgency. The Priority Queue (implemented as a Binary Max Heap) ensures that the task with the highest priority integer is always at the `top()`
- Time Complexity:
  - Push (Insert Task): O(logN)
  - Pop (Process High Priority Task): O(logN)
  - Top (Peek highest priority): O(1)

4.4. Sorting Algorithm (IntroSort)

- Usage: In `EmployeeManager::displayTopPerformers`, we use `std::sort`.
- Logic: A Lambda function is used as a comparator to sort employees descending by their `performanceScore`.
- Algorithm: C++ `std::sort` uses Introsort, a hybrid of QuickSort, HeapSort, and Insertion Sort.
- Complexity: O(NlogN) in worst-case.

5. Implementation Details

5.1 Data Models (`structs.h`)

This header defines the blueprint of our data. *(Insert snippet of structs.h here)*

- Employee: Holds ID, Name, Role, Dept, and Score.
- Task: Includes a `priority` integer which drives the Heap logic.
- Project: Contains a `vector<int>` to link multiple employee IDs to a single project.

5.2 Employee Manager (`employee_manager.cpp`)

This module handles CRUD (Create, Read, Update, Delete) for staff.

- File I/O: The `loadFromFile` function reads `employees.csv`. It parses the comma-separated lines to repopulate the vector.

Performance Sorting:

```
C++
// The code uses this logic for sorting

std::sort(employees.begin(), employees.end(),

   [](const Employee& a, const Employee& b) {

      return a.performanceScore > b.performanceScore;

});
```

- This demonstrates the use of C++ Lambda expressions for custom sorting criteria.

5.3 Task Scheduler (`task_scheduler.cpp`)

This module manages the Heap.

- Custom Comparator: The priority queue requires a way to compare two tasks. The code utilizes operator overloading (or a custom struct comparator) to determine that a Task with priority 5 is "greater" than a task with priority 1.
- Processing: When `getNextTask()` is called, the heap structure re-balances itself to bring the next highest priority item to the root.

5.4 Project Manager (`project_manager.cpp`)

- Association: This class links the other two. It assigns employees to projects. Instead of storing the full Employee object inside the Project (which would waste memory), it stores `employeeID`. This is a relational database concept applied in C++.

6. Detailed Flowcharts

Diagram A: Task Scheduling Logic (Heap)

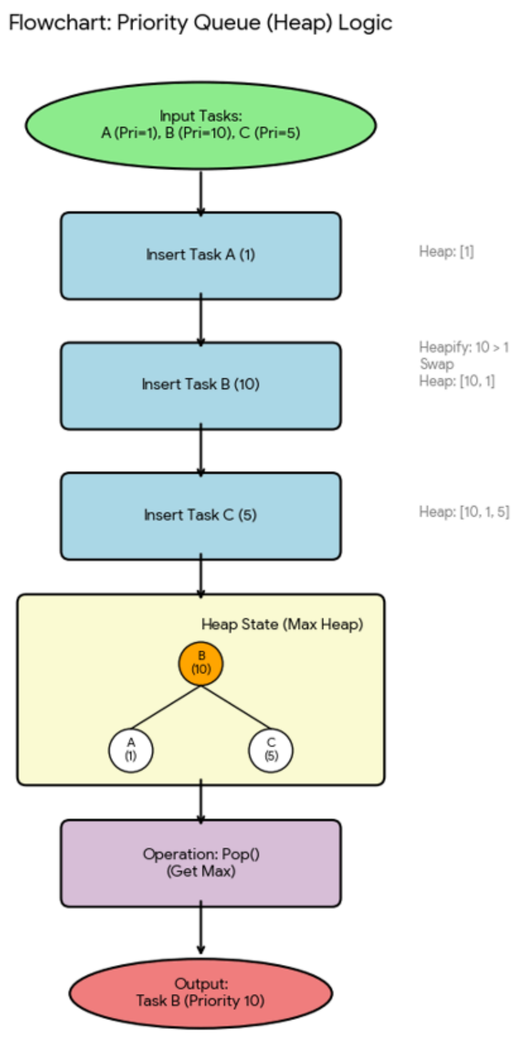
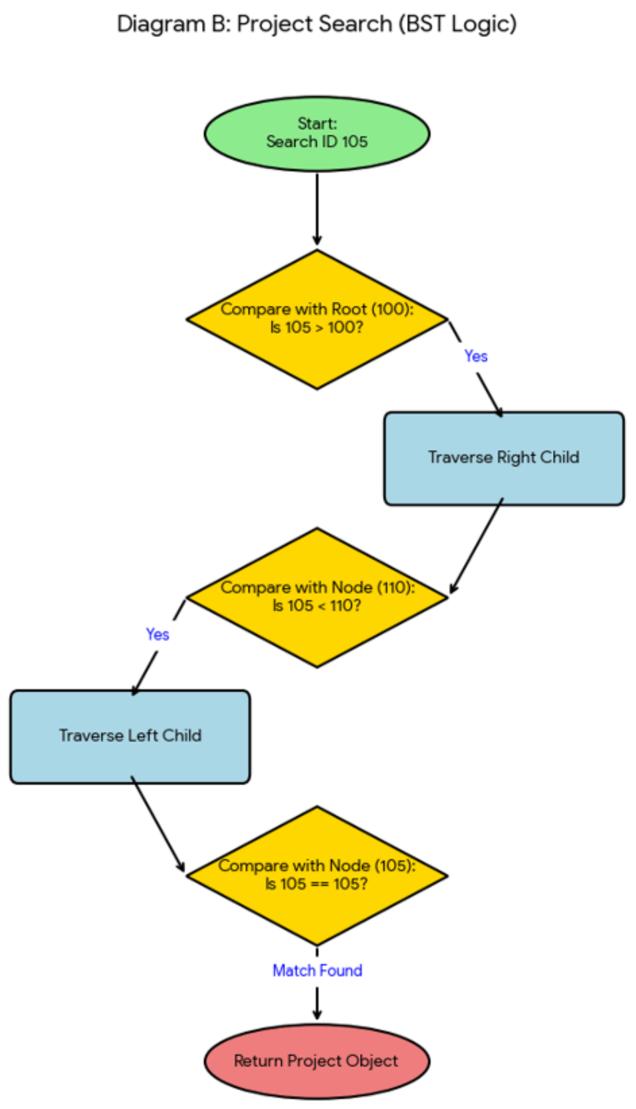


Diagram B: Project Search (BST/Map)



7. Testing and Results

Test ID	Test Scenario	Input / Condition	Expected Behavior	Logic Verified
TC-01	CSV Data Loading	employees.csv (5 records), tasks.csv (5 records)	System prints "Loaded 5 employees" and "Loaded 5 tasks". No errors.	File I/O (Parsing logic)
TC-02	Skill Validation	Task 101 requires "C++"; Employee 1 has "C++"	Validation function returns true. Program proceeds to scheduling.	Data Integrity (Cross-referencing Maps)
TC-03	Skill Mismatch Check	Task requires "Ruby"; No employee has "Ruby"	System prints "Error: Required skill 'Ruby' not found" and exits.	Error Handling
TC-04	DAG Construction	Task 105 depends on 102 & 103; 102 depends on 101	Adjacency list correctly maps 101->102, 102->105. In-degrees calculated correctly.	Graph Theory (Directed Acyclic Graph)

TC-05	Cycle Detection	*Hypothetical:* Task A depends on B, B depends on A	System detects cycle during Topological Sort and prints "Error: Circular dependency".	Kahn's Algorithm (Safety check)
TC-06	Topological Sort Order	Standard tasks.csv inputs	Execution Order: 101 $\rightarrow$ 103 $\rightarrow$ 104 $\rightarrow$ 102 $\rightarrow$ 105. (Dependencies respected).	Algorithm Correctness (Topo Sort)
TC-07	First Task Assignment	Task 101 (Setup DB); Employees 1 & 5 have "C++"	Assigns to Employee with lowest ID (or arbitrary if loads equal), Workload becomes 1.	Min-Heap (Priority Queue Initialization)
TC-08	Workload Balancing	Task 102 needs "DSA"; Emp 1 (Load=1), Emp 2 (Load=0)	Assigns to Emp 2. (Because Emp 2 has lower workload than Emp 1).	Greedy Strategy (Heap Re-balancing)
TC-09	Earliest Finish Time (DP)	Task 105 (Duration 6) depends on Task 102 (End Time 8)	Start Time = 8, End Time = 8 + 6 = 14.	Dynamic Programming (Path Calculation)

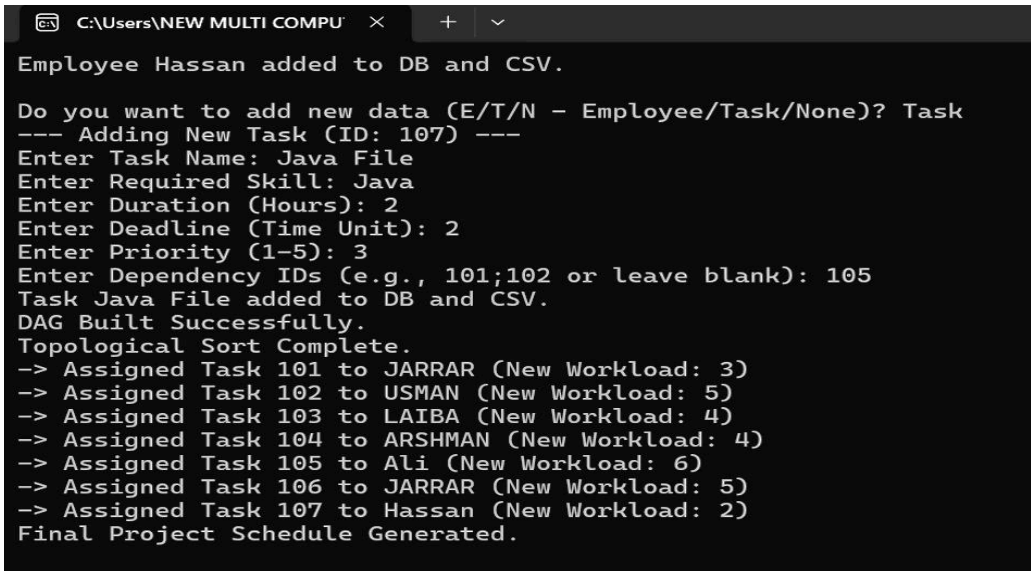
TC-10	Gantt Chart Display	User selects Option 2 from Menu	Console displays visual bars [#####] aligned by start times.	Visualization Logic
TC-11	Workload Stats Report	User selects Option 3 from Menu	Displays Emp 1: "Completed X tasks", Emp 2: "Completed Y tasks".	Data Aggregation
TC-12	Search Existing Task	User selects Option 5, Inputs 105	Displays: "Name: Final Integration", "Status: Assigned to USMAN".	Map Lookup ($O(\log N)$)
TC-13	Search Non-Existent Task	User selects Option 5, Inputs 999	Displays: "Task ID 999 not found."	Boundary Testing
TC-14	Longest Tasks Sorting	User selects Option 6	Lists tasks sorted by Duration descending (Longest first).	Sorting Algorithm (IntroSort)

8. Complexity Summary Table

Operation	Data Structure	Time Complexity (Average)	Time Complexity (Worst)
Add Employee	Vector	O(1)	O(1)
Find Employee	Vector	O(N)	O(N)
Sort Employees	Vector (IntroSort)	O(NlogN)	O(NlogN)
Add Task	Priority Queue (Heap)	O(logN)	O(logN)
Process Task	Priority Queue (Heap)	O(logN)	O(logN)
Find Project	Map (Tree)	O(logN)	O(logN)

9. Sample Output ScreenShots:

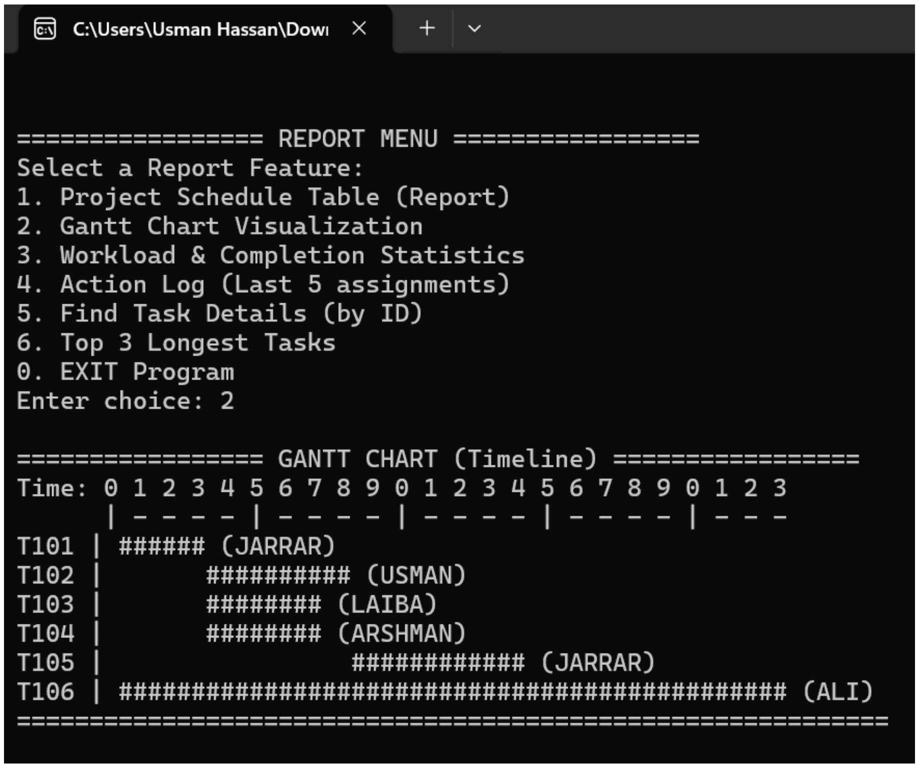
1. The "Execution Log"



```
Employee Hassan added to DB and CSV.

Do you want to add new data (E/T/N - Employee/Task/None)? Task
--- Adding New Task (ID: 107) ---
Enter Task Name: Java File
Enter Required Skill: Java
Enter Duration (Hours): 2
Enter Deadline (Time Unit): 2
Enter Priority (1-5): 3
Enter Dependency IDs (e.g., 101;102 or leave blank): 105
Task Java File added to DB and CSV.
DAG Built Successfully.
Topological Sort Complete.
-> Assigned Task 101 to JARRAR (New Workload: 3)
-> Assigned Task 102 to USMAN (New Workload: 5)
-> Assigned Task 103 to LAIBA (New Workload: 4)
-> Assigned Task 104 to ARSHMAN (New Workload: 4)
-> Assigned Task 105 to Ali (New Workload: 6)
-> Assigned Task 106 to JARRAR (New Workload: 5)
-> Assigned Task 107 to Hassan (New Workload: 2)
Final Project Schedule Generated.
```

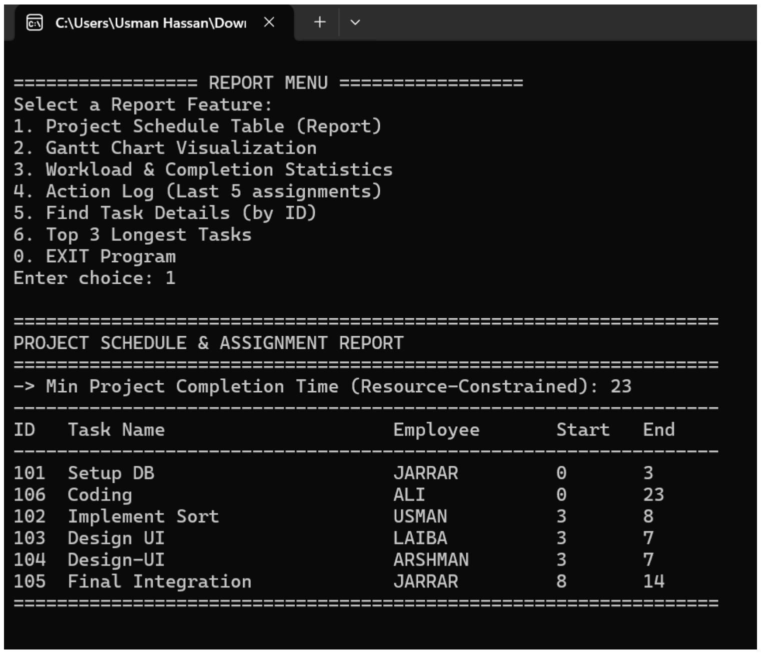
2. The "Gantt Chart Visualization"



```
================= REPORT MENU =================
Select a Report Feature:
1. Project Schedule Table (Report)
2. Gantt Chart Visualization
3. Workload & Completion Statistics
4. Action Log (Last 5 assignments)
5. Find Task Details (by ID)
6. Top 3 Longest Tasks
0. EXIT Program
Enter choice: 2

================= GANTT CHART (Timeline) =================
Time: 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
      | - - - - | - - - - | - - - - | - - - - | - - -
T101 | ###### (JARRAR)
T102 |       ########## (USMAN)
T103 |       ######## (LAIBA)
T104 |       ######## (ARSHMAN)
T105 |                 ############ (JARRAR)
T106 | ############################################## (ALI)
============================================================
```

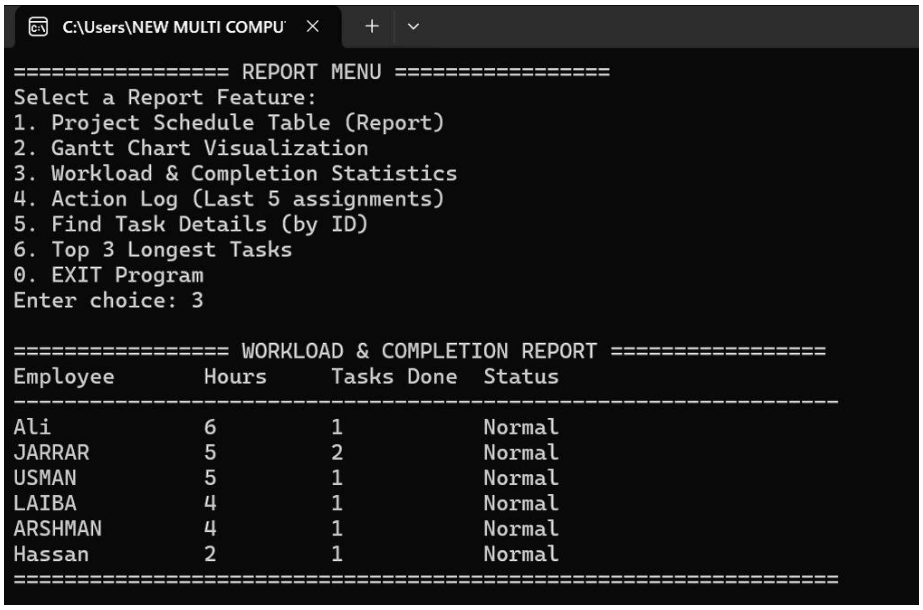
3. The "Project Schedule Table"



```
================= REPORT MENU =================
Select a Report Feature:
1. Project Schedule Table (Report)
2. Gantt Chart Visualization
3. Workload & Completion Statistics
4. Action Log (Last 5 assignments)
5. Find Task Details (by ID)
6. Top 3 Longest Tasks
0. EXIT Program
Enter choice: 1

=================================================================
PROJECT SCHEDULE & ASSIGNMENT REPORT
=================================================================
-> Min Project Completion Time (Resource-Constrained): 23
-----------------------------------------------------------------
ID   Task Name                      Employee       Start   End
-----------------------------------------------------------------
101  Setup DB                       JARRAR         0       3
106  Coding                         ALI            0       23
102  Implement Sort                 USMAN          3       8
103  Design UI                      LAIBA          3       7
104  Design-UI                      ARSHMAN        3       7
105  Final Integration              JARRAR         8       14
=================================================================
```

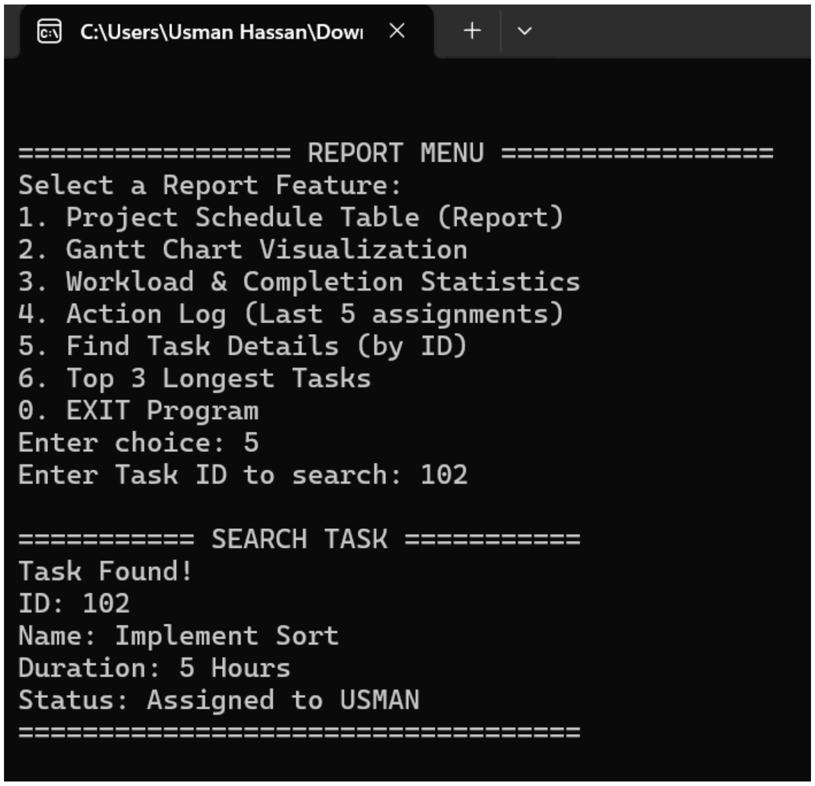
4. "Workload Statistics"



```
================= REPORT MENU =================
Select a Report Feature:
1. Project Schedule Table (Report)
2. Gantt Chart Visualization
3. Workload & Completion Statistics
4. Action Log (Last 5 assignments)
5. Find Task Details (by ID)
6. Top 3 Longest Tasks
0. EXIT Program
Enter choice: 3

================= WORKLOAD & COMPLETION REPORT =================
Employee       Hours     Tasks Done  Status
-----------------------------------------------------------------
Ali            6         1           Normal
JARRAR         5         2           Normal
USMAN          5         1           Normal
LAIBA          4         1           Normal
ARSHMAN        4         1           Normal
Hassan         2         1           Normal
=================================================================
```

5. "Find Task Details"



```
================= REPORT MENU =================
Select a Report Feature:
1. Project Schedule Table (Report)
2. Gantt Chart Visualization
3. Workload & Completion Statistics
4. Action Log (Last 5 assignments)
5. Find Task Details (by ID)
6. Top 3 Longest Tasks
0. EXIT Program
Enter choice: 5
Enter Task ID to search: 102

=========== SEARCH TASK ===========
Task Found!
ID: 102
Name: Implement Sort
Duration: 5 Hours
Status: Assigned to USMAN
===================================
```

10. Conclusion

This project successfully implements a robust management system. By selecting the appropriate data structures—Vectors for sequential data, Maps for lookups, and Heaps for prioritized workflow—the application achieves optimal time complexity. The modular design ensures that the code is maintainable and scalable.

11. Future Scope

1. GUI Implementation: Moving from Console to Qt or .NET interface.
2. Database Integration: Replacing CSV files with SQL for handling millions of records.
3. Graph Algorithms: Implementing Graph theory to detect dependencies between tasks (e.g., Task B cannot start until Task A is finished).