PROJECT REPORT

FOR
SECOND YEAR
COMPUTER SCIENCE
BATCH :- 2024-2028

DSA (CT- 159)

TASK MANAGEMENT SYSTEM

MEMBERS:-

1) M MUSTAFA (CT-24038)
2) SYED MUHAMMAD USMAN (CT-24031)
3) RAYYAN AHMED (CT-24032)

DISCIPLINE :- FSCS (COMPUTER SCIENCE)

SECTION :- A

SEMESTER :- FALL (2025-26)

1.Introduction:

The purpose of this project is to design and implement a **Task Scheduler** using the **Min-Heap data structure** in C++. In daily life, different tasks have different levels of importance. A system that automatically arranges these tasks based on priority can help users manage time more effectively. The Min-Heap ensures that the **task with the highest importance (lowest priority value)** always remains at the top.
This project provides a menu-driven interface that allows the user to add tasks, view tasks in sorted order, update tasks, and delete tasks while maintaining the heap structure.

---

2. Objectives:

The main objectives of this project include:

1. To understand and implement the concept of **Min-Heap**.
2. To manage and organize tasks based on their **priority levels**.
3. To apply concepts of **structs, vectors, recursion, and heap algorithms** in C++.
4. To develop a basic, user-friendly **task management application**.
5. To strengthen problem-solving skills by combining data structures with real-life scenarios.

---

3. Working of the System:

Each task contains the following fields:

- **Description**
- **Deadline**
- **Priority** (Smaller value = Higher importance)

Tasks are stored in a **vector**, which represents a Min-Heap. The root of the heap always contains the **most important task**.
The user interacts with the system using a simple menu. Every operation maintains the heap property using *heapifyUp* and *heapifyDown* functions.

---

4. Key Features:

1. Add Task
The user enters the description, deadline, and priority.
The new task is inserted into the heap and adjusted using **heapifyUp()**, so the Min-

Heap order remains correct.
2. View All Tasks
The tasks are displayed in sorted order by repeatedly removing the root of a copy of the heap.
This method ensures that tasks appear from **highest to lowest priority**.
3. Update Task
The system searches for a task by description.
The user can edit the description, deadline, and priority.
After modification, the heap is reorganized using both **heapifyUp** and **heapifyDown**.
4. Delete Task
The selected task is removed by replacing it with the last element in the heap.
The heap property is restored afterward.
5. Input Validation
The program handles incorrect input using cin.fail() and ensures smooth input functioning with cin.clear() and cin.ignore().

5.Algorithms Used:

a. Heapify Up (Bubble Up)
Used after insertion and sometimes after updating priority.
If a child node has a smaller priority value than its parent, the two nodes are swapped until the correct position is reached.
b. Heapify Down (Bubble Down)
Used after deletion or when a node's priority increases.
The node is compared with its left and right children, and swapped with the smaller child to maintain Min-Heap order.
c. Linear Search
findTaskIndex() performs a simple search to locate a task by description during update or delete operations.

---

6. Data Structures:

1. Struct
Used to store multiple fields of a task:
```
struct Task {
    int priority;
    string deadline;
    string description;
};
```
2. Vector
Serves as the underlying structure for the Min-Heap, allowing dynamic resizing and easy element access.

7. Flow of the Program:

1. Display menu
2. User selects an option
3. Perform operation (Add / View / Update / Delete)
4. Maintain heap structure
5. Return to the menu until user selects Exit

The flow is simple, interactive, and easy for users to follow.

---

8. Output

```
========== Task Scheduler ==========
1. Add Task
2. View Tasks
3. Update Task
4. Delete Task
5. Exit
Choose an option:
```

---

9. Conclusion:

This project successfully demonstrates how a **Min-Heap** can be used to manage tasks based on priority in an efficient and organized manner. By combining theoretical concepts with practical implementation, the system becomes a helpful tool for scheduling tasks. It also strengthens skills in:

- Data Structures
- Algorithm design
- Recursion
- User input handling
- C++ programming fundamentals

The project is modular, easy to extend, and reflects proper use of heap operations in a real-life scenario.