**DSA PROJECT BRIEF**

**NAME**: USMAN AHMED JAN-4744

MAZHAR ALI-4743

KHALID BADSHAZOI-4596

HUZAIFA-4593

**Submitted To:** [MR.SHAKEEL]  
**Department:** SOFTWARE ENGINEEERING

**Project Title:** University Timetable Generator

**Course:** Data Structures and Algorithms (DSA)  
**Date:** 23-June -2025

**Implemented Operations & Their Descriptions**

| **Operation** | **Description** |
| --- | --- |
| addCourse() | Adds a new course to the course list. |
| inputCourses() | Takes user input for multiple courses (code, name, teacher, hours). |
| listCourses() | Displays all currently added courses. |
| generateSchedule() | Allocates time slots for each course without teacher/time conflicts. |
| displaySchedule() | Shows the generated timetable on the console. |
| saveToCSV() | Saves the timetable to a .csv file in a readable format. |
| clearCourses() | Clears all data from the system (courses, schedules, flags). |
| showCredits() | Displays author/project credits. |
| printMenu() | Prints main menu options on the screen. |

**Data Structures Used**

* **vector<Course>** – Stores the list of all added courses.
* **vector<ScheduledCourse>** – Stores the finalized timetable (with time slots).
* **map<string, set<string>>** – Tracks each teacher’s occupied slots to avoid conflicts.
* **bool slotTaken[5][6]** – Marks time slots across all days to avoid double booking.

**Example Flow of Operations**

1. User chooses **Add Course** → Enters details → Data stored in vector.
2. User chooses **Generate Schedule** → Algorithm checks empty slots, assigns.
3. User chooses **Display Schedule** → Table format timetable shown.
4. User chooses **Save to CSV** → Writes the schedule to timetable.csv.
5. User can also **Clear**, **View Credits**, or **Exit**.

### ****Class Diagram****

lua

CopyEdit

+----------------+

| Timetable |

+----------------+

| - courses | → vector<Course>

| - schedule | → vector<ScheduledCourse>

| - teacherBusy | → map<string, set<string>>

+----------------+

| +addCourse() |

| +inputCourses() |

| +listCourses() |

| +generateSchedule() |

| +displaySchedule() |

| +saveToCSV() |

| +clearCourses() |

+----------------+

▲

|

|

+-----------+ +---------------------+

| Course | | ScheduledCourse |

+-----------+ +---------------------+

| +code | | +course : Course |

| +name | | +day : string |

| +teacher | | +time : string |

| +hoursPerWeek |

+-----------+ +---------------------+

**Comparison Of Data Structure**

### ****1.**** vector ****vs**** array

| **Feature** | **vector (Used)** | **array** |
| --- | --- | --- |
| Size | Dynamic (can grow/shrink at runtime) | Fixed size |
| Flexibility | Easy to insert/remove elements | Manual handling needed |
| Use Case | Stores course and schedule lists | Couldn’t handle unknown number of courses |
| Why Used? | Courses are user-defined, dynamic | vector provides better flexibility |

**vector<Course>** — stores list of all courses  
 **vector<ScheduledCourse>** — stores all generated slots

### ****2.**** map<string, set<string>> ****vs**** unordered\_map ****or**** list

| **Feature** | **map (Used)** | **unordered\_map** | **list** |
| --- | --- | --- | --- |
| Ordered keys | Yes (alphabetically sorted) | No | N/A |
| Access Speed | O(log n) | O(1) average | O(n) |
| Duplicate Keys | No | No | N/A |
| Why Used? | Tracks unique slots per teacher | Good readability | Not suitable here |

**map<string, set<string>> teacherBusy** — tracks each teacher’s occupied time slots  
**set** avoids duplicate slots per teacher

### ****3.**** bool[][] ****vs**** set<string>

| **Feature** | **bool slotTaken[5][6] (Used)** | **set<string> alternative** |
| --- | --- | --- |
| Access Time | O(1) constant access | O(log n) |
| Memory Usage | Low (only 5×6 = 30 cells) | Higher |
| Clarity | Easy to understand as time table grid | Less visual |
| Why Used? | Timetable grid needs fast slot checking | Efficient for small static layout |

2D boolean array was perfect for checking **conflicts** across days & times.

### Summary of Choices

| **Data Structure** | **Why It Was Chosen** |
| --- | --- |
| vector | For dynamic, expandable lists (courses, scheduled slots) |
| map<string, set> | To map teachers to their used time slots efficiently |
| bool[5][6] | Fast lookup for time slot availability (grid-based logic) |
| 1. **Searching Algorithm Analysis**Where Searching is Relevant:  * When checking if a **teacher** is already booked in a time slot. * When checking if a **time slot** is already taken in the timetable grid. * When potentially adding **search by course name/code** (future feature).  Algorithm Used:✅ **Set-based Search** (set<string>)  * Used to track time slots assigned to each teacher. * teacherBusy[teacher].count(slot) checks if slot is taken.  | **Feature** | **Value** | | --- | --- | | Data Structure | set<string> | | Time Complexity | **O(log n)** | | Justification | Efficient for frequent inserts + lookups without duplicates |  **2. Sorting Algorithm Analysis**Where Sorting Can Be Used (Optional Enhancements):  * Sorting courses alphabetically by code or name before scheduling. * Sorting scheduled timetable entries by day and time before display.  📘 Algorithm Used:✅ std::sort() (from <algorithm>) – Optional Example (not in base code but recommended):  cpp  CopyEdit  sort(schedule.begin(), schedule.end(), [](ScheduledCourse a, ScheduledCourse b) {  return a.day < b.day || (a.day == b.day && a.time < b.time);  });   | **Feature** | **Value** | | --- | --- | | Data Structure | vector<ScheduledCourse> | | Algorithm | IntroSort (Quick + Heap + Insertion) | | Time Complexity | **O(n log n)** | | Use Case | Sorting final timetable for display | | Benefit | Improves readability | |  |