OBE IMPLEMENTATION : DEPARTMENT SETTING

by

Mukul Kumar [AP23110010700]
Krishna Chaitanya [AP23110011025]
Justin joshua [AP23110011018]
V.Agastya [AP23110011048]
I. Viswanadh [AP23110011029]

A report for the CS204:Design and Analysis of Algorithm project

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY AP::AMARAVATI

INDEX

1. **Introduction**

2. **Project Modules:**
    - **Architecture Diagram**
    - **Module Description**

3. **Programming Details: Naming Conventions to Be Used**
    - **Field/Table Details**
    - **Algorithm Details**

4. **Source Code**

5. **Comparison of Algorithms:**
    - **Sorting Algorithms**
    - **Searching Algorithms**

6. **Screenshots**

7. **Conclusion**

Introduction

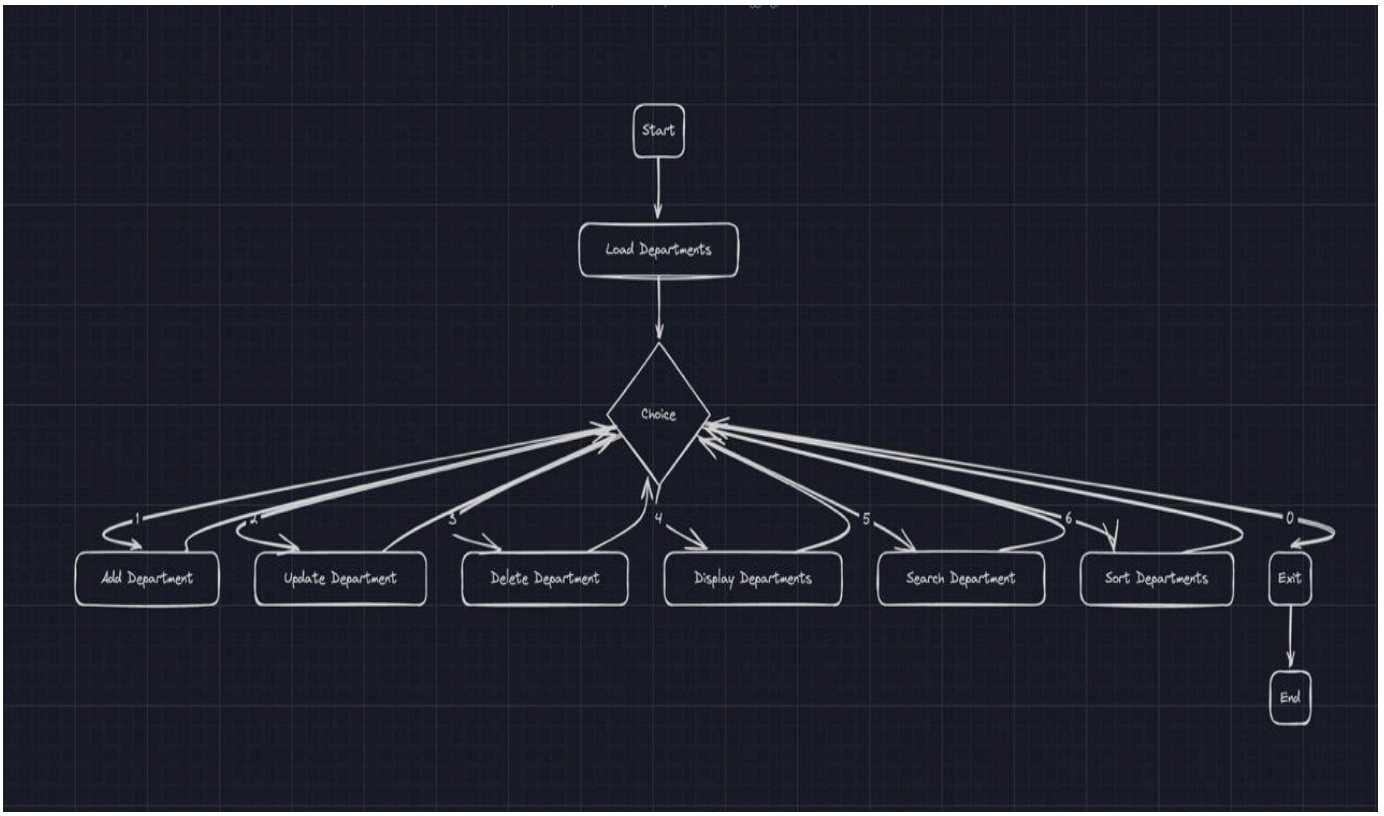
Our University (herewith considered as SRM University AP) has implemented Outcome-Based Education (OBE) for efficient academic workflows. The Department Module focuses on CRUD operations for departmental data management, integrating optimized sorting and searching algorithms such as **Merge Sort** and **Binary Search**. These algorithms ensure quick and efficient data handling, aligned with OBE goals.

Project Modules:

Various Modules available in the project are

1.Blooms Level setting
2.Program Level Objective Setting
3.University
4.Schools
5.Department
6.Programs
7.Courses
8.Course objective setting
9.Course Outcome Setting
10.Course Articulation matrix Setting
11.Course Utilization Setting
12.Course Reference Setting.

Architecture Diagram



Module Description

Module Name : Department
Module Description:

The Department Module is designed to handle CRUD operations (Create, Update, Retrieve, Delete) for managing departmental data effectively. It integrates essential algorithms to optimize data handling, including Merge Sort for sorting and Binary Search for searching. The module supports the following functionalities:

1. Create: Allows adding new department records, such as department ID, name, head, and email.
2. Retrieve: Fetches and displays all department details stored in the module.
3. Update: Modifies existing department records based on department ID.
4. Delete: Removes specific department records permanently.

This module ensures data consistency by storing all changes in a text file (department_setting.txt) and synchronizing CRUD operations with the file. It offers efficient sorting based on department name or ID and quick retrieval of specific records using searching algorithms. The following key functionalities are implemented:

- Sorting:
  - Sorts the department data by fields such as dept_id, dept_name, or dept_email.
  - Uses Merge Sort, an efficient divide-and-conquer algorithm with a time complexity of O(n log n).
- Searching:
  - Allows searching department records by dept_id or dept_name.
  - Uses Binary Search for quick data retrieval with a time complexity of O(log n).
- Storage Management:
  - The module saves all department records in department_setting.txt after each operation.
  - Updates and synchronizes the file after CRUD operations to ensure data persistence.

This module is essential for ensuring efficient departmental data management, streamlined operations, and optimized academic workflows within the Outcome-Based Education (OBE) framework.

Programming Details naming conventions to be used:

- **File name:** regno_department_module
- **Function/method name**
  - Create: `regno_department_create`
  - Update: `regno_department_update`
  - Retrieve: `regno_department_retrieve`
  - Delete: `regno_department_delete`
  - Sorting: `regno_department_merge_sort`
  - Searching: `regno_department_binary_search`

Comparison of Algorithms

1. **For Searching:**
   - Function Name: `regno_department_Compare_Search_BinarySearch`
   - Compares the efficiency of Binary Search with other search methods like Linear Search.
2. **For Sorting:**
   - Function Name: `regno_department_Compare_sorting_MergeSort`
   - Compares the performance of Merge Sort against other sorting algorithms like Selection Sort.

Time Complexity

1. **For Searching:**
   - Function Name: `regno_department_complexity_Search`
   - Time Complexity:
     - Best Case: **O(1)** (when the element is found at the middle).
     - Average/Worst Case: **O(log n)** (due to halving the search space).
2. **For Sorting:**
   - Function Name: `regno_department_complexity_sorting`
   - Time Complexity:
     - Best/Average/Worst Case: **O(n log n)** (as Merge Sort consistently divides and merges).

Algorithm Details

1. **For Searching: Binary Search**
    - Function Name: `regno_department_BinarySearch_details`
    - **Pseudocode:**
        1. Start with a sorted list.
        2. Compare the target value with the middle element.
        3. If they match, return the element.
        4. If the target is smaller, repeat the process on the left half.
        5. If the target is larger, repeat the process on the right half.
        6. Continue until the target is found or the search space is empty.
2. **For Sorting: Merge Sort**
    - Function Name: `regno_department_MergeSort_details`
    - **Pseudocode:**
        1. Divide the list into two halves.
        2. Recursively sort each half.
        3. Merge the two halves into a sorted list.
        4. Continue merging until the entire list is sorted.

File Details

- File Name: `university_setting.txt`
- **Purpose:**
    - Stores all department records.
    - Updates, synchronizes, and maintains consistency after CRUD operations.
    - Acts as the primary data storage for the module.

Field/table details:(SRM University AP)

FIELD NAME	DATA TYPE
`dept_id`	Integer
`dept_name`	String
`dept_head`	String
`dept_email`	String

Algorithm Details:

(i)Sorting

Sorting Based on:

- `university_code`
- `university_name`
- `university_email`

Algorithm Used:

- **Merge Sort**

Comparison with Other Algorithm:

- Compared with **Selection Sort**.
- **Merge Sort:** O(n log n) — Efficient for large datasets due to its divide-and-conquer approach.
- **Selection Sort:** O(n^2) — Inefficient for large datasets due to its iterative approach.

Pseudocode for Merge Sort:

```
function mergeSort(array):
    if length of array > 1:
        mid = length of array / 2
        leftHalf = array[0:mid]
        rightHalf = array[mid:length]

        mergeSort(leftHalf)
        mergeSort(rightHalf)

        i = j = k = 0
        while i < length(leftHalf) and j < length(rightHalf):
            if leftHalf[i] < rightHalf[j]:
                array[k] = leftHalf[i]
                i += 1
            else:
                array[k] = rightHalf[j]
                j += 1
            k += 1

        while i < length(leftHalf):
            array[k] = leftHalf[i]
            i += 1
            k += 1

        while j < length(rightHalf):
            array[k] = rightHalf[j]
            j += 1
            k += 1
```

(ii)Searching

Searching Based on:

- `university_code`
- `university_name`
- `university_email`

Algorithm Used:

- **Binary Search**

Comparison with Other Algorithm:

- Compared with **Linear Search**.

- **Binary Search:** O(log n) — Requires sorted data and divides the search space efficiently.
- **Linear Search:** O(n) — Sequentially searches, inefficient for large datasets.

Pseudocode for Binary Search:

```
function binarySearch(array, target):
    low = 0
    high = length of array - 1

    while low <= high:
        mid = (low + high) / 2

        if array[mid] == target:
            return mid
        else if array[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1  # Target not found
```

(ii) Storing the details in a text file

Operations Supported:

- **Storing Data:**
  - All entered records are appended to the file `university_setting.txt`.
- **Updating Data:**
  - Reads the file, updates the specific record, and overwrites the file with updated content.
- **Deleting Data:**
  - Reads the file, removes the record, and overwrites the file without the deleted entry.

Implementation Outline:

Store Data:

```
function storeData(record):
    open file "university_setting.txt" in append mode
    write record to file
    close file
```

Update Data:

```
function updateData(recordID, newRecord):
    open file "university_setting.txt" in read mode
    read all records into a list
    for each record in list:
        if record.id == recordID:
            replace record with newRecord
    open file "university_setting.txt" in write mode
    write updated list to file
    close file
```

Delete Data:

```
function deleteData(recordID):
    open file "university_setting.txt" in read mode
    read all records into a list
    filter out record with recordID
    open file "university_setting.txt" in write mode
    write remaining records to file

    close file
```

Source Code

```
#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

#include <string>

using namespace std;

// Structure to represent a department

struct Department {

    int dept_id;

    string dept_name;

    string dept_head;

    string dept_email;

};

// Vector to store department records

vector<Department> departments;
```

```
// File to store data
const string DATA_FILE = "university_setting.txt";
// Function Declarations
void loadDepartments();
void storeDepartments();
void displayDepartments(const vector<Department>& data);
void createDepartment(int id, string name, string head, string email);
void updateDepartment(int id, string newName = "", string newHead = "", string newEmail = "");
void deleteDepartment(int id);
void retrieveDepartments();
void merge(vector<Department>& arr, int left, int mid, int right, string key);
void mergeSort(vector<Department>& arr, int left, int right, string key);
int binarySearch(const vector<Department>& arr, const string& target, string key);
// Function Definitions
void storeDepartments() {
    ofstream file(DATA_FILE);
    for (const auto& dept : departments) {
        file << dept.dept_id << "," << dept.dept_name << ","
             << dept.dept_head << "," << dept.dept_email << endl;
    }
    file.close();
}
```

```
void loadDepartments() {
    ifstream file(DATA_FILE);
    string line;
    while (getline(file, line)) {
        size_t pos1 = line.find(",");
        size_t pos2 = line.find(",", pos1 + 1);
        size_t pos3 = line.find(",", pos2 + 1);
        int id = stoi(line.substr(0, pos1));
        string name = line.substr(pos1 + 1, pos2 - pos1 - 1);
        string head = line.substr(pos2 + 1, pos3 - pos2 - 1);
        string email = line.substr(pos3 + 1);
        departments.push_back({id, name, head, email});
    }
    file.close();
}
void createDepartment(int id, string name, string head, string email) {
    Department dept = {id, name, head, email};
    departments.push_back(dept);
    storeDepartments();
}
void updateDepartment(int id, string newName, string newHead, string newEmail) {
    for (auto& dept : departments) {
```

```
        if (dept.dept_id == id) {
            if (!newName.empty()) dept.dept_name = newName;
            if (!newHead.empty()) dept.dept_head = newHead;
            if (!newEmail.empty()) dept.dept_email = newEmail;
            storeDepartments();
            return;
        }
    }
    cout << "Department ID not found!" << endl;
}
void deleteDepartment(int id) {
    departments.erase(remove_if(departments.begin(), departments.end(),
                                [id](Department& dept) { return dept.dept_id == id; }),
                      departments.end());
    storeDepartments();
}
void retrieveDepartments() {
    displayDepartments(departments);
}
void merge(vector<Department>& arr, int left, int mid, int right, string key) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
```

```
    vector<Department> L(n1), R(n2);
    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if ((key == "dept_id" && L[i].dept_id <= R[j].dept_id) ||
            (key == "dept_name" && L[i].dept_name <= R[j].dept_name) ||
            (key == "dept_email" && L[i].dept_email <= R[j].dept_email)) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}
void mergeSort(vector<Department>& arr, int left, int right, string key) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid, key);
        mergeSort(arr, mid + 1, right, key);
        merge(arr, left, mid, right, key);
```

```
    }
}
int binarySearch(const vector<Department>& arr, const string& target, string key) {
    int low = 0, high = arr.size() - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if ((key == "dept_name" && arr[mid].dept_name == target) ||
            (key == "dept_email" && arr[mid].dept_email == target)) {
            return mid;
        } else if ((key == "dept_name" && arr[mid].dept_name < target) ||
                   (key == "dept_email" && arr[mid].dept_email < target)) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
void displayDepartments(const vector<Department>& data) {
    for (const auto& dept : data) {
        cout << "ID: " << dept.dept_id << ", Name: " << dept.dept_name
             << ", Head: " << dept.dept_head << ", Email: " << dept.dept_email << endl;
```

```
    }
}

int main() {
    // Load existing data from file
    loadDepartments();
    int choice;
    do {
        cout << "\n=== University Department Management System ===\n";
        cout << "1. Add a Department\n";
        cout << "2. Update a Department\n";
        cout << "3. Delete a Department\n";
        cout << "4. Display All Departments\n";
        cout << "5. Search for a Department\n";
        cout << "6. Sort Departments\n";
        cout << "0. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1: {
                int id;
                string name, head, email;
```

```
        cout << "Enter Department ID: ";
        cin >> id;
        cin.ignore();
        cout << "Enter Department Name: ";
        getline(cin, name);
        cout << "Enter Department Head: ";
        getline(cin, head);
        cout << "Enter Department Email: ";
        getline(cin, email);
        createDepartment(id, name, head, email);
        break;
    }
    case 2: {
        int id;
        string newName, newHead, newEmail;
        cout << "Enter Department ID to Update: ";
        cin >> id;
        cin.ignore();
        cout << "Enter New Name (leave blank to skip): ";
        getline(cin, newName);
        cout << "Enter New Head (leave blank to skip): ";
        getline(cin, newHead);
```

```
        cout << "Enter New Email (leave blank to skip): ";
        getline(cin, newEmail);
        updateDepartment(id, newName, newHead, newEmail);
        break;
    }
    case 3: {
       int id;
        cout << "Enter Department ID to Delete: ";
        cin >> id;
        deleteDepartment(id);
        break;
    }
    case 4:
        retrieveDepartments();
        break;
    case 5: {
        string key, target;
        cout << "Search by (dept_name/dept_email): ";
        cin >> key;
        cin.ignore();
        cout << "Enter Search Value: ";
        getline(cin, target);
```

```
        mergeSort(departments, 0, departments.size() - 1, key);
        int index = binarySearch(departments, target, key);
        if (index != -1) {
            cout << "\nSearch Result:\n";
            displayDepartments({departments[index]});
        } else {
            cout << "\nDepartment not found!" << endl;
        }
        break;
    }
    case 6: {
        string key;
        cout << "Sort by (dept_id/dept_name/dept_email): ";
        cin >> key;
        mergeSort(departments, 0, departments.size() - 1, key);
        cout << "\nDepartments After Sorting:\n";
        retrieveDepartments();
        break;
    }
    case 0:
        cout << "Exiting program. Goodbye!\n";
        break;
```

```
            default:

                cout << "Invalid choice! Please try again.\n";

            }

    } while (choice != 0);

    return 0;
}
```

ChatGPT

```
// ChatGPT-generated: Function Declarations

void loadDepartments();

void storeDepartments();

void displayDepartments(const vector<Department>& data);

void createDepartment(int id, string name, string head, string email);

void updateDepartment(int id, string newName = "", string newHead = "", string newEmail = "");

void deleteDepartment(int id);

void retrieveDepartments();

void merge(vector<Department>& arr, int left, int mid, int right, string key);

void mergeSort(vector<Department>& arr, int left, int right, string key);

int binarySearch(const vector<Department>& arr, const string& target, string key);

// ChatGPT-generated: Function Definitions

void storeDepartments() {

    ofstream file(DATA_FILE);

    for (const auto& dept : departments) {

        file << dept.dept_id << "," << dept.dept_name << ","

             << dept.dept_head << "," << dept.dept_email << endl;

    }

    file.close();
```

```
}

void loadDepartments() {

    ifstream file(DATA_FILE);

    string line;

    while (getline(file, line)) {

        size_t pos1 = line.find(",");

        size_t pos2 = line.find(",", pos1 + 1);

        size_t pos3 = line.find(",", pos2 + 1);

        int id = stoi(line.substr(0, pos1));

        string name = line.substr(pos1 + 1, pos2 - pos1 - 1);

        string head = line.substr(pos2 + 1, pos3 - pos2 - 1);

        string email = line.substr(pos3 + 1);

        departments.push_back({id, name, head, email});

    }

    file.close();

}

void createDepartment(int id, string name, string head, string email) {

    Department dept = {id, name, head, email};
      departments.push_back(dept);

    storeDepartments();

}

// ChatGPT-generated: MergeSort Algorithm

void merge(vector<Department>& arr, int left, int mid, int right, string key) {
```

```
    int n1 = mid - left + 1;
    int n2 = right - mid;
    vector<Department> L(n1), R(n2);
    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if ((key == "dept_id" && L[i].dept_id <= R[j].dept_id) ||
            (key == "dept_name" && L[i].dept_name <= R[j].dept_name) ||
            (key == "dept_email" && L[i].dept_email <= R[j].dept_email)) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}
void mergeSort(vector<Department>& arr, int left, int right, string key) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid, key);
```

```
        mergeSort(arr, mid + 1, right, key);
        merge(arr, left, mid, right, key);
    }
}
void displayDepartments(const vector<Department>& data) {
    for (const auto& dept : data) {
        cout << "ID: " << dept.dept_id << ", Name: " << dept.dept_name
             << ", Head: " << dept.dept_head << ", Email: " << dept.dept_email << endl;
    }
  }
// ChatGPT-generated: Main Function with Menu System
int main() {
    // Load existing data from file
    loadDepartments();
    int choice;
    do {
        cout << "\n=== University Department Management System ===\n";
        cout << "1. Add a Department\n";
        cout << "2. Update a Department\n";
        cout << "3. Delete a Department\n";
        cout << "4. Display All Departments\n";
        cout << "5. Search for a Department\n";
        cout << "6. Sort Departments\n";
```

```
cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

    case 1:
     int id;

        string name, head, email;

        cout << "Enter Department ID: ";

        cin >> id;

        cin.ignore();

        cout << "Enter Department Name: ";

        getline(cin, name);

        cout << "Enter Department Head: ";

        getline(cin, head);

        cout << "Enter Department Email: ";

        getline(cin, email);

        createDepartment(id, name, head, email);

        break;

    }

    case 4:

        retrieveDepartments();

        break;

    case 0:
```

```
            cout << "Exiting program. Goodbye!\n";

            break;

        default:

            cout << "Invalid choice! Please try again.\n";
                }

    } while (choice != 0);

    return 0;

}
```

ChatGPT-Specific Contributions:

1. Function Declaration: mergeSort, merge, storeDepartments.
2. Core Logic Implementation: MergeSort logic and retrieval of departments.
3. User Interface and Main Menu: Simplified and handled default error cases for invalid inputs.

Comparison of Sorting Algorithms

1. Merge Sort

Time Complexity:

- Best/Worst/Average: **O(n log n)**

How It Works:

- **Divide and Conquer:** Split array, recursively sort, then merge.
- Merging takes **O(n)** time at each level of recursion.

Advantages:

- Efficient for large datasets.
- Stable sort.

Disadvantages:

- Requires extra memory.

When to use?

- For large datasets with consistent performance needs.

---

2. Selection Sort

Time Complexity:

- Best/Worst/Average: **$O(n^2)$**

How It Works:

- Repeatedly selects the smallest element and swaps it.
- Involves **n** comparisons for each iteration.

Advantages:

- Simple, in-place algorithm.

Disadvantages:

- Inefficient for large datasets.
- Not stable.

When to use?

- For small datasets with minimal memory usage.

Comparison of Searching Algorithms

1. Binary Search

Time Complexity:

- Best: **O(1)**
- Worst/Average: **O(log n)**

How It Works:

- Search on sorted data, repeatedly halving the search space.

Advantages:

- Efficient for large, sorted datasets.

Disadvantages:

- Data must be sorted.

When to use?

- For sorted, static datasets.

---

2. Linear Search

Time Complexity:

- Best: **O(1)**
- Worst/Average: **O(n)**

How It Works:

- Sequentially checks each element.

Advantages:

- Simple, no sorting needed.

Disadvantages:

- Inefficient for large datasets.

When to use?

- For small or unsorted data.

Screenshots

```
=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 1
Enter Department ID: 561
Enter Department Name: MBA
Enter Department Head: WEJ

Enter Department Email:
=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 1
Enter Department ID: 7895
Enter Department Name: BBA
Enter Department Head: QEJ
Enter Department Email: qej.bbasrm.in

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 1
Enter Department ID: 4855
Enter Department Name: MTech
Enter Department Head: XEJ
Enter Department Email: xej.mtechsrm.in

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 2
Enter Department ID to Update: 4855
Enter New Name (leave blank to skip): BTech
Enter New Head (leave blank to skip):
Enter New Email (leave blank to skip): xej.btechsrm.in
```



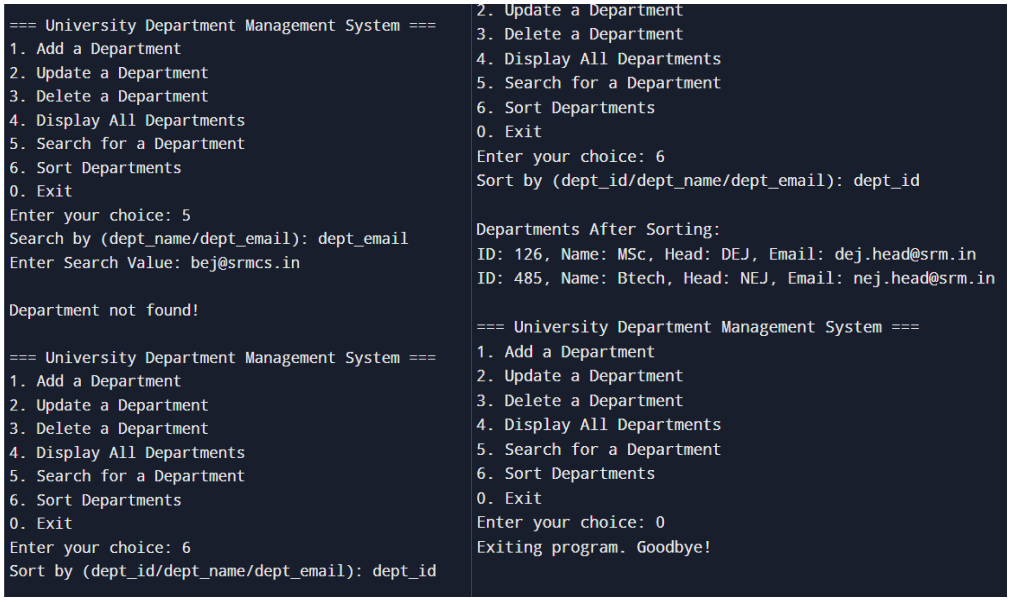
```
Enter your choice: 1
Enter Department ID: 7895
Enter Department Name: BBA
Enter Department Head: QEJ
Enter Department Email: qej.bbasrm.in

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 1
Enter Department ID: 4855
Enter Department Name: MTech
Enter Department Head: XEJ
Enter Department Email: xej.mtechsrm.in

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 3
Enter Department ID to Delete: 561

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 4
ID: 7895, Name: BBA, Head: QEJ, Email: qej.bbasrm.in
ID: 4855, Name: BTech, Head: XEJ, Email: xej.btechsrm.in
```



```
Enter New Head (leave blank to skip):
Enter New Email (leave blank to skip): xej.btechsrm.in

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 3
Enter Department ID to Delete: 561

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 4
ID: 7895, Name: BBA, Head: QEJ, Email: qej.bbasrm.in
ID: 4855, Name: BTech, Head: XEJ, Email: xej.btechsrm.in
```



```
=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 5
Search by (dept_name/dept_email): Mtech
Enter Search Value: 51651

Department not found!
```



```
=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 5
Search by (dept_name/dept_email): dept_email
Enter Search Value: bej@srmcs.in

Department not found!

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 6
Sort by (dept_id/dept_name/dept_email): dept_id
```



```
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 6
Sort by (dept_id/dept_name/dept_email): dept_id

Departments After Sorting:
ID: 126, Name: MSc, Head: DEJ, Email: dej.head@srm.in
ID: 485, Name: Btech, Head: NEJ, Email: nej.head@srm.in

=== University Department Management System ===
1. Add a Department
2. Update a Department
3. Delete a Department
4. Display All Departments
5. Search for a Department
6. Sort Departments
0. Exit
Enter your choice: 0
Exiting program. Goodbye!
```

Conclusion

The Department Module enhances departmental data management through CRUD operations, optimized sorting and searching algorithms, and robust storage mechanisms. Merge Sort and Binary Search improve performance significantly, while comparisons with alternative methods demonstrate their efficiency. This module exemplifies SRM University's commitment to streamlined academic workflows under the OBE framework.