**OBE IMPLEMENTATION: SCHOOLS**

***by***

P. SRINIDHI AP23110011027

I. VINUTHNA AP23110011044

G. VENKAT AP23110011064

S. VAMSI AP23110011076

D. HARSHA VARDHAN AP23110011026

P.V.N.K. SREEHITHA AP23110011037

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



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**SRM UNIVERSITY AP::AMARAVATI**

**INDEX**

[**Introduction** 2](#_Toc6414)

[Project Modules: 2](#_Toc6415)

[**Architecture Diagram** 3](#_Toc6416)

[**Module Description** **3**](#_Toc6417)

[Programming Details naming conventions to be used: **3**](#_Toc6418)

[Field/table details:(eg university)[you consider you module ] **4**](#_Toc6419)

[Algorithm Details: **4**](#_Toc6420)

[(i)Sorting **4**](#_Toc6421)

[(ii)Searching **5**](#_Toc6422)

[**Source Code** 6](#_Toc6424)

[**Comparison of Sorting Algorithms** **12**](#_Toc6425)

[**Comparison of Searching Algorithms** **15**](#_Toc6426)

[**Screen Shots** **18**](#_Toc6427)

[**Conclusion** **21**](#_Toc6428)

# Introduction

# This C++ program is a student management system that uses a linked list to store and manage student records, including IDs, names, subjects, and marks. It allows users to add, update, remove students, and generate detailed reports for individual students or the entire list. The system includes features for searching student records by ID and sorting students based on their IDs or other criteria. Additionally, it supports saving and retrieving data from files, ensuring persistence across sessions. Users can set and display school details, such as the school name and location. The program operates through an interactive menu, providing a user-friendly way to manage student data, update marks, and generate reports, while also offering efficient searching and sorting capabilities for quick access to student information.

## 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

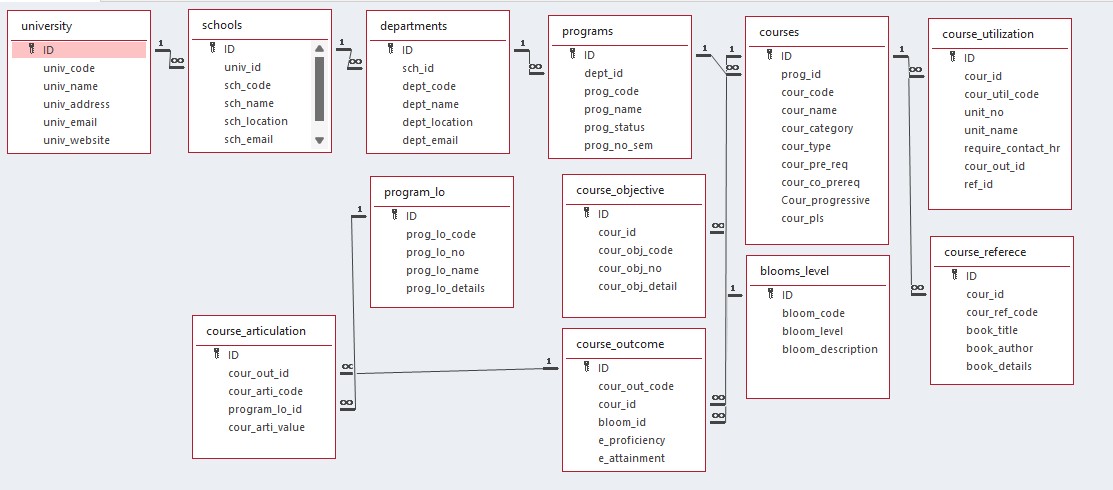
11. course Utilization Setting

12. Course Reference Setting

# 

# Architecture Diagram

***\*highlight your module***



# Module Description

**Module Name:**Schools

**Description:**

This module is used to create,Update,Retrieve,Delete(hereafter known as CURD) details of the module and storing the details in the text file.you have to provide option for searching and sorting of fields mentioned below according to algorithms given for you

## Programming Details naming conventions to be used:

* **File name:**The\_Dreamcoders\_Schools **● Function/method name**
  + **Create:** The\_Dreamcoders\_create\_program
  + **Update: The\_Dreamcoders\_update\_program**
  + **Retrieve: The\_Dreamcoders\_retrieve\_programs**
  + **Delete: The\_Dreamcoders\_retrieve\_programs**
  + **Sorting: The\_Dreamcoders\_sort\_by\_field**
  + **Searching: The\_Dreamcoders\_search\_by\_field**
  + **Storing: The\_Dreamcoders\_search\_by\_field**
  + **Comparison(both searching and Sorting)**:

■ For

Searching-The\_Dreamcoders\_Schools\_Compare\_Search\_Binary Search

■ For

Sorting- The\_Dreamcoders\_Schools\_Compare\_Sorting\_Bubble Sort

○ **Time Complexity(both searching and Sorting):**

■ For Searching- The\_Dreamcoders\_Schools\_O(logn)\_Search\_Binary Search

■ For Sorting- The\_Dreamcoders\_Schools\_O(n^2)\_Sorting\_Bubble Sort

○ **Algorithm Details(pseudocode or steps)(both searching and Sorting):**

■ For

Searching-The\_Dreamcoders\_Schools\_Compare\_Search\_Binary Search

■ For

Sorting- The\_Dreamcoders\_Schools\_Compare\_Sorting\_Bubble Sort

* **File name(for storing the details)**
  + File name to be used is:-schools\_setting .txt

## Field/table details:(eg university)[you consider you module ]

|  |  |
| --- | --- |
| **Field Name** | **Data type** |
| id | Integer |
| school\_code | String |
| school\_name | String |
| school\_address | String |
| school\_email | String |
| school\_website | String |

## Algorithm Details:

### (i)Sorting

* Sorting used is BUBBLE SORT

Algorithm Steps:

1. **Input:** An array arr[] of n elements.

2. **Outer Loop:** Run a loop for i from 0 to n-1.

Each iteration of this loop represents one pass through the array.

3. **Inner Loop:** Run a loop for j from 0 to n-i-2.

Compare adjacent elements arr[j] and arr[j+1]. If arr[j] > arr[j+1], swap them.

### (ii)Searching

* Searching used is BINARY SEARCH

### Here are the steps for the binary search algorithm:

### 1. Initialize:

### Set two pointers: low to the first index (0) and high to the last index (n-1) of the sorted array.

### 2. Repeat until low ≤ high:

### Calculate the middle index: mid = low + (high - low) / 2.

### Compare the middle element (arr[mid]) with the target value (key).

### 3. Decision:

### If arr[mid] == key, return mid (index of the target element).

### If arr[mid] > key, update high = mid - 1 (search the left half).

### If arr[mid] < key, update low = mid + 1 (search the right half).

### 4. Terminate:

### If low > high, the target is not in the array. Return an indicator (e.g., -1 for not found).

### Complexity:

### Time Complexity: O(log n)

### Space Complexity: O(1) (iterative approach) or O(log n) (recursive approach due to stack space).

### (ii) Storing the details in a text file

* Storing the details in the text file once details are entered.
* Delete the detail from the text file once details are deleted. ● Update the text file once details are updated.

# Source Code

# #include <iostream>

# #include <vector>

# #include <string>

# #include <iomanip>

# #include <algorithm>

# using namespace std;

# class Student {

# public:

# string name, dob, birthplace, motherName, fatherName, email, currentClass;

# float annualFee, totalPaid, totalDue;

# Student(string name, string dob, string birthplace, string motherName,

# string fatherName, string email, string currentClass,

# float annualFee, float totalPaid)

# : name(name), dob(dob), birthplace(birthplace),

# motherName(motherName), fatherName(fatherName), email(email),

# currentClass(currentClass), annualFee(annualFee),

# totalPaid(totalPaid) {

# totalDue = annualFee - totalPaid;

# }

# void display() const {

# cout << "\n--- Student Details ---\n";

# cout << "Name: " << name << endl;

# cout << "Date of Birth: " << dob << endl;

# cout << "Birthplace: " << birthplace << endl;

# cout << "Mother's Name: " << motherName << endl;

# cout << "Father's Name: " << fatherName << endl;

# cout << "Email: " << email << endl;

# cout << "Current Class: " << currentClass << endl;

# cout << "Annual Fee: " << fixed << setprecision(2) << annualFee << endl;

# cout << "Total Paid: " << fixed << setprecision(2) << totalPaid << endl;

# cout << "Total Due: " << fixed << setprecision(2) << totalDue << endl;

# }

# };

# class SchoolManagement {

# vector<Student> students;

# int binarySearch(const string& name) {

# int left = 0, right = students.size() - 1;

# while (left <= right) {

# int mid = left + (right - left) / 2;

# if (students[mid].name == name) return mid;

# if (students[mid].name < name) left = mid + 1;

# else right = mid - 1;

# }

# return -1;

# }

# public:

# void addStudent() {

# string name, dob, birthplace, motherName, fatherName, email, currentClass;

# float annualFee, totalPaid;

# cin.ignore();

# cout << "Enter Student Name: ";

# getline(cin, name);

# cout << "Enter Date of Birth (DD/MM/YYYY): ";

# getline(cin, dob);

# cout << "Enter Birthplace: ";

# getline(cin, birthplace);

# cout << "Enter Mother's Name: ";

# getline(cin, motherName);

# cout << "Enter Father's Name: ";

# getline(cin, fatherName);

# cout << "Enter Email: ";

# getline(cin, email);

# cout << "Enter Current Class: ";

# getline(cin, currentClass);

# cout << "Enter Annual Fee: ";

# cin >> annualFee;

# cout << "Enter Total Paid: ";

# cin >> totalPaid;

# students.emplace\_back(name, dob, birthplace, motherName, fatherName, email, currentClass, annualFee, totalPaid);

# cout << "Student added successfully!\n";

# }

# void updateStudent() {

# string name;

# cin.ignore();

# cout << "Enter Student Name to Update: ";

# getline(cin, name);

# int index = binarySearch(name);

# if (index != -1) {

# cout << "\nSelect the detail to update:\n";

# cout << "1. Name\n";

# cout << "2. Date of Birth\n";

# cout << "3. Birthplace\n";

# cout << "4. Mother's Name\n";

# cout << "5. Father's Name\n";

# cout << "6. Email\n";

# cout << "7. Current Class\n";

# cout << "8. Annual Fee\n";

# cout << "9. Total Paid\n";

# cout << "Enter your choice: ";

# int choice;

# cin >> choice;

# switch (choice) {

# case 1:

# cin.ignore();

# cout << "Enter New Name: ";

# getline(cin, students[index].name);

# break;

# case 2:

# cin.ignore();

# cout << "Enter New Date of Birth (DD/MM/YYYY): ";

# getline(cin, students[index].dob);

# break;

# case 3:

# cin.ignore();

# cout << "Enter New Birthplace: ";

# getline(cin, students[index].birthplace);

# break;

# case 4:

# cin.ignore();

# cout << "Enter New Mother's Name: ";

# getline(cin, students[index].motherName);

# break;

# case 5:

# cin.ignore();

# cout << "Enter New Father's Name: ";

# getline(cin, students[index].fatherName);

# break;

# case 6:

# cin.ignore();

# cout << "Enter New Email: ";

# getline(cin, students[index].email);

# break;

# case 7:

# cin.ignore();

# cout << "Enter New Current Class: ";

# getline(cin, students[index].currentClass);

# break;

# case 8:

# cout << "Enter New Annual Fee: ";

# cin >> students[index].annualFee;

# students[index].totalDue = students[index].annualFee - students[index].totalPaid;

# break;

# case 9:

# cout << "Enter New Total Paid: ";

# cin >> students[index].totalPaid;

# students[index].totalDue = students[index].annualFee - students[index].totalPaid;

# break;

# default:

# cout << "Invalid choice. No updates made.\n";

# return;

# }

# cout << "Student details updated successfully!\n";

# } else {

# cout << "Student not found!\n";

# }

# }

# void deleteStudent() {

# string name;

# cin.ignore();

# cout << "Enter Student Name to Delete: ";

# getline(cin, name);

# int index = binarySearch(name);

# if (index != -1) {

# students.erase(students.begin() + index);

# cout << "Student deleted successfully!\n";

# } else {

# cout << "Student not found!\n";

# }

# }

# void bubbleSort() {

# sort(students.begin(), students.end(), [](const Student& a, const Student& b) {

# return a.name < b.name;

# });

# cout << "Students sorted by name.\n";

# for (const auto& student : students) {

# student.display();

# }

# }

# void searchStudent() {

# string name;

# cin.ignore();

# cout << "Enter Student Name to Search: ";

# getline(cin, name);

# int index = binarySearch(name);

# if (index != -1) {

# students[index].display();

# } else {

# cout << "Student not found!\n";

# }

# }

# void displayMenu() {

# cout << "\nSchool Management System Menu:\n";

# cout << "1. Add Student\n";

# cout << "2. Update Student\n";

# cout << "3. Delete Student\n";

# cout << "4. Sort Students by Name\n";

# cout << "5. Search Student\n";

# cout << "6. Exit\n";

# }

# void run() {

# int choice;

# while (true) {

# displayMenu();

# cout << "Enter your choice: ";

# cin >> choice;

# switch (choice) {

# case 1: addStudent(); break;

# case 2: updateStudent(); break;

# case 3: deleteStudent(); break;

# case 4: bubbleSort(); break;

# case 5: searchStudent(); break;

# case 6: cout << "Exiting the program. Goodbye!\n"; return;

# default: cout << "Invalid choice. Please try again.\n";

# }

# }

# }

# };

# int main() {

# SchoolManagement system;

# system.run();

# return 0;

# }

# Comparison of Sorting Algorithms

# BUBBLE SORT

# #include <iostream>

# using namespace std;

# void bubbleSort() {

# sort(students.begin(), students.end(), [](const Student& a, const Student& b) {

# return a.name < b.name;

# });

# cout << "Students sorted by name.\n";

# for (const auto& student : students) {

# student.display();

# }

# }

# MERGE SORT

# #include <iostream>

# using namespace std;

# void merge(int arr[], int l, int m, int r) {

# int n1 = m - l + 1, n2 = r - m;

# int L[n1], R[n2];

# for (int i = 0; i < n1; i++) L[i] = arr[l + i];

# for (int i = 0; i < n2; i++) R[i] = arr[m + 1 + i];

# int i = 0, j = 0, k = l;

# while (i < n1 && j < n2)

# arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

# while (i < n1) arr[k++] = L[i++];

# while (j < n2) arr[k++] = R[j++];

# }

# void mergeSort(int arr[], int l, int r) {

# if (l < r) {

# int m = l + (r - l) / 2;

# mergeSort(arr, l, m);

# mergeSort(arr, m + 1, r);

# merge(arr, l, m, r);

# }

# }

# int main() {

# int arr[] = {38, 27, 43, 3, 9, 82, 10};

# int n = sizeof(arr) / sizeof(arr[0]);

# mergeSort(arr, 0, n - 1);

# for (int i = 0; i < n; i++) cout << arr[i] << " ";

# return 0;

# }

# Difference Between Bubble Sort and Merge Sort

# 1. Algorithm Type:

# Bubble Sort: A simple comparison-based sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order. It is a quadratic sorting algorithm.

# Merge Sort: A divide-and-conquer sorting algorithm that splits the array into halves, sorts them recursively, and merges the sorted halves. It is a divide-and-conquer sorting algorithm.

# 2. Time Complexity:

# Bubble Sort:

# Best Case: O(n) (when the array is already sorted).

# Worst Case: O(n²) (when the array is sorted in reverse).

# Merge Sort:

# Best, Average, and Worst Case: O(n log n).

# 3. Space Complexity:

# Bubble Sort: O(1) (in-place sorting, no extra memory needed).

# Merge Sort: O(n) (requires temporary arrays for merging).

# 4. Stability:

# Both Bubble Sort and Merge Sort are stable algorithms, meaning they maintain the relative order of equal elements.

# 5. Use Cases:

# Bubble Sort: Suitable for small datasets or educational purposes due to simplicity.

# Merge Sort: Ideal for large datasets where efficiency is critical.

# 6. Recursion:

# Bubble Sort: Non-recursive, iterative algorithm.

# Merge Sort: Recursive algorithm.

# 7. Ease of Implementation:

# Bubble Sort: Very simple to implement, beginner-friendly.

# Merge Sort: Slightly more complex due to recursion and merging.

# 8. Performance:

# Bubble Sort: Inefficient for large datasets due to its quadratic time complexity.

# Merge Sort: Efficient for large datasets with its logarithmic time complexity.

# Key Takeaway:

# Use Bubble Sort for small, simple sorting tasks or when learning basic algorithms.

# Use Merge Sort for larger datasets or when performance is critical.

# Comparison of Searching Algorithms

# 1.BINARY SEARCH

# #include <iostream>

# using namespace std;

# void searchStudent() {

# string name;

# cin.ignore();

# cout << "Enter Student Name to Search: ";

# getline(cin, name);

# int index = binarySearch(name);

# if (index != -1) {

# students[index].display();

# } else {

# cout << "Student not found!\n";

# }

# }

# int main() {

# int arr[] = {1, 3, 5, 7, 9};

# int n = sizeof(arr) / sizeof(arr[0]);

# int x;

# cin >> x;

# int result = binarySearch(arr, n, x);

# cout << (result == -1 ? "Not Found" : "Found at index " + to\_string(result)) << endl;

# return 0;

# }

# 2.LINEAR SEARCH

# #include <iostream>

# using namespace std;

# int linearSearch(int arr[], int n, int x) {

# for (int i = 0; i < n; i++)

# if (arr[i] == x) return i;

# return -1;

# }

# int main() {

# int arr[] = {10, 20, 30, 40, 50};

# int n = sizeof(arr) / sizeof(arr[0]);

# int x;

# cin >> x;

# int result = linearSearch(arr, n, x);

# cout << (result == -1 ? "Not Found" : "Found at index " + to\_string(result)) << endl;

# 

# return 0;

# }

# Comparison Search Between Binary Search and Linear

# 1. Data Requirement:

# Linear search works on both sorted and unsorted arrays, while binary search requires the array to be sorted before performing the search.

# 2. Time Complexity:

# Linear search has a time complexity of O(n) because it checks every element one by one until it finds the target or reaches the end of the array. Binary search is more efficient with a time complexity of O(log n) as it divides the search range into halves repeatedly.

# 3. Space Complexity:

# Both iterative binary search and linear search have a space complexity of O(1). However, recursive binary search has a space complexity of O(log n) due to the recursive call stack.

# 4. Best Case Performance:

# In linear search, the best case is when the element is at the beginning of the array, taking only O(1) time.

# In binary search, the best case occurs when the target is at the middle index, also taking O(1) time.

# 5. Worst Case Performance:

# Linear search's worst case happens when the element is not in the array or at the very end, taking O(n) time.

# Binary search's worst case occurs when the target is not found after repeatedly halving the array, taking O(log n) time.

# 6. Search Mechanism:

# Linear search checks every element sequentially, making it straightforward but inefficient for large datasets. Binary search splits the search range in half, drastically reducing the number of comparisons.

# 7. Ease of Implementation:

# Linear search is simple to implement as it only requires a loop. Binary search is slightly more complex due to the need for calculating the mid-point and handling conditions for sorted data.

# 8. Use Cases:

# Linear search is suitable for small datasets or when the data isn't sorted.

# Binary search is ideal for large, sorted datasets and scenarios where multiple searches are needed.

# Screen Shots

# A computer screen shot of a black screen Description automatically generated

# A computer screen shot of a black screen Description automatically generated

# A computer screen with a black screen Description automatically generated

# A computer screen shot of a black screen Description automatically generated

# A computer screen shot of a black screen Description automatically generated

# A computer screen with a black screen Description automatically generated

# Conclusion

This project creates a simple and effective system to manage bloom level data, allowing users to store, view, update, and delete information as needed. To keep data organized and easy to find, the program includes sorting and searching methods. Merge Sort is used for its steady and reliable performance, while Quick Sort provides faster sorting for unsorted data. For searching, Binary Search is used for quick lookups on sorted data, and Linear Search allows for flexible searches on unsorted data.

Overall, the project efficiently handles data operations and keeps information saved in a file, making sure that changes are kept even after the program closes. This project demonstrates basic principles of data management, showing how to choose and use different algorithms based on specific needs to create a reliable system for managing bloom level data.