OBE IMPLEMENTATION:SCHOOL SETTING

BY

SRAVYA. A AP23110010710

SARAYU.K AP23110010707

LAVANYA.B AP23110011146

RISHVIK.K AP23110011103

JYOSHNA.P AP23110011145

A report for the CSE204: Design and Analysis of Algorithm project

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY AP::AMARAVATI

INDEX

INTRODUCTION:

To design and implement a database system for managing school information, including details like ID, university ID, code, name, location, and email address. The system should provide efficient search and sorting functionalities for easy access and management of school data.

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

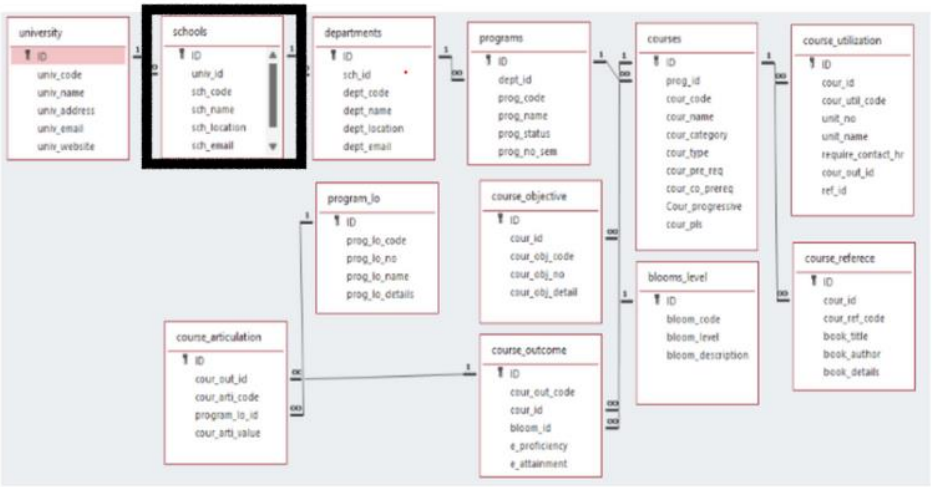
9.Course Outcome Setting

10.Course Articulation matrix Setting

11.Course Utilization Setting

12.Course Reference Setting.

Architecture Diagram



MODULE DESCRIPTION

: **Module Name: School**

Module Description

1. Data Structure Module Description:

Defines the structure to represent school information, including fields like ID, university ID, code, name, location, and email.

2. Input/Output Module Description:

Handles input and output operations, including reading data from a file, displaying information to the console, and writing data to a file.

3. Search Module Description:

Implements linear and binary search algorithms to find a school record based on its ID.

4. CRUD Module Description:

Handles the creation, reading, updating, and deletion of school records.

5. Sort Module Description:

Implements insertion sort and merge sort algorithms to sort the school records based on their ID.

6. Main Module Description:

The main entry point of the program. It coordinates the execution of other modules, handles user input, and displays output

PROGRAMMING DETAILS NAMING CONVENTIONS TO BE USED:

File name:AP23110010707_School

Function/method name:

1.Create: AP23110010707_School_create

2.Display:AP23110010710_School_display

3.Free:AP231001011146_School_free

Comparison(both searching and sorting):

■ For Searching-

AP23110010710_school_Compare_Search_youralgorithm name

■ For Sorting-

AP23110010707_school_Compare_sorting_youralgorithm nameTIME COMPLEXITY:(for both searching and sorting techniques)

■ For Searching-AP23110010707_school_complexity_Search

■ For Sorting- AP23110010710_school_compexity_sorting

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

■ For Searching-

AP23110011145_module_name_your_search_algorithmname_details

■ For Sorting-

AP23110011103_module_name_your_sort_algorithmname_details

● File name(for storing the details)

○ File name to be used is :-School_setting .txt

Field Name	Data type
id	integer
school_code	String
school_name	String
school_location	String
school_ email	String

ALGORITHM DETAILS:

(i)Sorting

● You have to provide sorting based on school code ,school_name , school_email.

● Compare the algorithm you have used with any of the other sorting algorithm

● Display the time complexity of both algorithms.

● Display the pseudocode/algorithm of the sorting algorithm used by you.

(ii)Searching

● You have provide sorting based on school code, school_name, school_email

● Compare the algorithm used with any of the other algorithm you have learned

● Display the time complexity of both algorithms.

● Display the pseudocode/algorithm of the searching algorithm used by you.

(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 <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure to represent a school
typedef struct School {
   int ID;
   int univ_id;
   char sch_code[50];
   char sch_name[100];
   char sch_location[100];
   char sch_email[100];
} School;

// Function to create a new school
School* createSchool(int ID, int univ_id, const char* sch_code, const
char* sch_name, const char* sch_location, const char* sch_email) {
   School* school = (School*) malloc(sizeof(School));
   if (school == NULL) {
      fprintf(stderr, "Memory allocation failed\n");
      exit(1);
   }
   school->ID = ID;
   school->univ_id = univ_id;
   strcpy(school->sch_code, sch_code);
   strcpy(school->sch_name, sch_name);
   strcpy(school->sch_location, sch_location);
   strcpy(school->sch_email, sch_email);
   return school;
}

// Function to display a school
void displaySchool(const School* school) {
   printf("ID: %d\n", school->ID);
```

```
    printf("Univ ID: %d\n", school->univ_id);
    printf("School Code: %s\n", school->sch_code);
    printf("School Name: %s\n", school->sch_name);
    printf("School Location: %s\n", school->sch_location);
    printf("School Email: %s\n", school->sch_email);
    printf("\n");
}

// Insertion sort function
void insertionSort(School** schools, int num_schools) {
    for (int i = 1; i < num_schools; i++) {
        School* key = schools[i];
        int j = i - 1;

        // Move elements of schools[0..i-1] that are greater than key
        // to one position ahead of their current position
        while (j >= 0 && schools[j]->ID > key->ID) {
            schools[j + 1] = schools[j];
            j--;
        }
        schools[j + 1] = key;
    }
}

// Merge function for merge sort
void merge(School** schools, int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1; // Size of left subarray
    int n2 = right - mid; // Size of right subarray

    // Create temporary arrays
    School** L = (School*)malloc(n1 * sizeof(School));
    School** R = (School*)malloc(n2 * sizeof(School));

    // Copy data to temporary arrays
```

```
for (i = 0; i < n1; i++)
    L[i] = schools[left + i];

for (j = 0; j < n2; j++)
    R[j] = schools[mid + 1 + j];

// Merge the temporary arrays back into schools[left..right]
i = 0; // Initial index of first subarray
j = 0; // Initial index of second subarray
k = left; // Initial index of merged subarray

while (i < n1 && j < n2) {
    if (L[i]->ID <= R[j]->ID) {
        schools[k] = L[i];
        i++;
    } else {
        schools[k] = R[j];
        j++;
    }
    k++;
}

// Copy the remaining elements of L[], if there are any
while (i < n1) {
    schools[k] = L[i];
    i++;
    k++;
}

// Copy the remaining elements of R[], if there are any
while (j < n2) {
    schools[k] = R[j];
    j++;
    k++;
}

// Free temporary arrays
free(L);
free(R);
```

```
}

// Merge sort function
void mergeSort(School** schools, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        // Sort first and second halves
        mergeSort(schools, left, mid);
        mergeSort(schools, mid + 1, right);

        // Merge the sorted halves
        merge(schools, left, mid, right);
    }
}

// Linear search function
School* linearSearch(School** schools, int num_schools, int searchID) {
    for (int i = 0; i < num_schools; i++) {
        if (schools[i]->ID == searchID) {
            return schools[i]; // Return the found school
        }
    }
    return NULL; // Return NULL if not found
}

// Binary search function
School* binarySearch(School** schools, int num_schools, int searchID) {
    int left = 0;
    int right = num_schools - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (schools[mid]->ID == searchID) {
            return schools[mid]; // Return the found school
        }
        if (schools[mid]->ID < searchID) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
```

```
  }
  return NULL; // Return NULL if not found
}

int main() {
  // Create some schools in an unsorted order for demonstration purposes
  School* schools[] = {
createSchool(1, 102, "S001", "Happy Valley International School", "Agiripally",
"Happyvalley@gmail.com"),
    createSchool(4, 103, "S004", "Bluebells International School", "Delhi",
"Delhi123@gmail.com"),
    createSchool(2, 104, "S002", "Wise Woods", "Gudivada",
"wisewoods123@gmail.com"),
    createSchool(7, 107, "S007", "Dhirubhai Ambani International School",
"Mumbai", “ambani123@gmail.com"),
    createSchool(9, 109, "S009", “Oberoi International School", "Mumbai",
“oberoi789@gmail.com"),
     createSchool(3, 103, "S003", "Pathway School", "Noida ",
“pathwaynodia@gmail.com"),
      createSchool(5, 105, "S005", “Heritage learning  School",
"Gurugram",”heritage1@gmail.com"),
     createSchool(8, 108, "S008", "Sherwood School", "Naintal
“,”osco@gmail.com"),
      createSchool(10, 110, "S010", "Lawence School", "Ooty",
“lawence789@gmail.com"),
     createSchool(6, 101, "S006", "DPS", "Vijayawada",”DPS1@gmail.com”),
  };
};

  int num_schools = sizeof(schools) / sizeof(schools[0]);

  // Sort schools using Insertion Sort
  insertionSort(schools, num_schools);

  // Display all sorted schools using Insertion Sort
  printf("Sorted Schools using Insertion Sort:\n");
  for (int i = 0; i < num_schools; i++) {
    displaySchool(schools[i]);
  }
```

```
  // Sort again using Merge Sort for demonstration purposes
  mergeSort(schools, 0, num_schools - 1);

  // Display all sorted schools using Merge Sort
  printf("Sorted Schools using Merge Sort:\n");
  for (int i = 0; i < num_schools; i++) {
    displaySchool(schools[i]);
  }

  // Search for a specific school by ID using Linear Search
  int searchIDLinear = 3; // Example search ID for linear search
  School* foundSchoolLinear = linearSearch(schools, num_schools,
searchIDLinear);

  if (foundSchoolLinear != NULL) {
    printf("Found School using Linear Search:\n");
    displaySchool(foundSchoolLinear);
  } else {
    printf("No school found with ID: %d using Linear Search\n",
searchIDLinear);
  }

  // Search for a specific school by ID using Binary Search
  int searchIDBinary = 5; // Example search ID for binary search
  School* foundSchoolBinary = binarySearch(schools, num_schools,
searchIDBinary);

  if (foundSchoolBinary != NULL) {
    printf("Found School using Binary Search:\n");
    displaySchool(foundSchoolBinary);
  } else {
    printf("No school found with ID: %d using Binary Search\n",
searchIDBinary);
  }

  // Free the memory allocated for the schools
  for (int i = 0; i < num_schools; i++) {
    free(schools[i]);
  }
```

```
  return 0;
}
```

COMPARISON OF SORTING ALGORITHMS:

```
void compareSortingAlgorithms(School** schools, int num_schools)
 {
 School** mergeSortSchools = malloc(num_schools *
sizeof(School*));
for (int i = 0; i < num_schools; i++) {
mergeSortSchools[i] = schools[i];
}
 clock_t start_merge = clock();
 mergeSort(mergeSortSchools, 0, num_schools - 1);
 clock_t end_merge = clock();
double time_merge = ((double)(end_merge - start_merge)) /
CLOCKS_PER_SEC;
 printf("Merge Sort:\n");
 for (int i = 0; i < num_schools; i++) {
displaySchool(mergeSortSchools[i]);
 }
 printf("Time taken for Merge Sort: %.6f seconds\n", time_merge);
free(mergeSortSchools);
 School** insertionSortSchools = malloc(num_schools *
sizeof(School*));
for (int i = 0; i < num_schools; i++) {
 insertionSortSchools[i] = schools[i];
}
 clock_t start_insertion = clock();
insertionSort(insertionSortSchools, num_schools);
clock_t end_insertion = clock();
double time_insertion = ((double)(end_insertion - start_insertion)) /
CLOCKS_PER_SEC; printf("\nInsertion Sort:\n");
for (int i = 0; i < num_schools; i++) {
 displaySchool(insertionSortSchools[i]);
 }
 printf("Time taken for Insertion Sort: %.6f seconds\n",
time_insertion);
 free(insertionSortSchools);
```

Time Complexity : O(n2)

COMPARISON OF SEARCHING ALGORITHMS

```
int compareSchools(const void* a, const void* b)
 {
 School* schoolA = *(School**)a;
 School* schoolB = *(School**)b;
 return (schoolA->ID - schoolB->ID);
}
```

CONCLUSION:

In summary, this module framework effectively manages school information through a well-structured design that enhances functionality and maintainability. Key Takeaways Data Structure Module: Provides a solid foundation for creating, displaying, and comparing school records. Input/Output Module: Ensures smooth data interaction with users and external files. Search Module: Implements efficient search algorithms for quick record retrieval. Sort Module: Organizes school records using effective sorting techniques. CRUD Module: Manages the lifecycle of school records through essential functions for adding, updating, and deleting entries. Main Module: Coordinates the execution of the program, integrating all modules based on user input. Future Enhancements Potential

improvements could include: User Authentication: To secure sensitive data. Advanced Search Filters: For more refined search capabilities. Graphical User Interface (GUI): To enhance user interaction. Data Analytics: To provide insights into school performance. This modular approach not only streamlines development but also allows for scalability and adaptability to meet future needs, ensuring efficient management of school data with high integrity and usability.