Project Report on

Department Module

Submitted By:

Team Arka:

AP23110011191-ch.Harini

AP23110011197-G.Hema sri

AP23110011202-k.Rishita

AP23110011204-sk.Shabana sulthana

AP23110011184-Kalyan Ram

AP23110011187-Eswar



**INDEX**

**Introduction............................................................................................................................................................**

Project Modules:................................................................................................................................................

**Architecture Diagram.............................................................................................................................................**

**Module Description................................................................................................................................................**

Programming Details naming conventions to be used:......................................................................................

Field/table details:(eg department)............................................................................

Algorithm Details:...............................................................................................................................................

(i)Sorting.......................................................................................................................................................

(ii)Searching.................................................................................................................................................

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

**Source Code...........................................................................................................................................................**

**Comparison of Sorting Algorithms......................................................................................................................**

**Comparison of Searching Algorithms.................................................................................................................**

**Screen Shots..........................................................................................................................................................**

**Conclusion..............................................................................................................................................................**



## Introduction

## Our Department Management System is a C program designed to manage departmental data within an educational institution. It allows users to create, view, update, delete, and search department records, with data stored in a file for persistence. The program supports sorting by department code, name, or email, using insertion and selection sort algorithms, and provides both linear and binary search functionalities. A menu-driven interface makes it easy to perform these operations, showcasing essential concepts in file handling and algorithm comparisons.

#### Project Modules:

Various Modules available in the project are 1.Blooms Level setting

2.Program Level Objective Setting 3.University

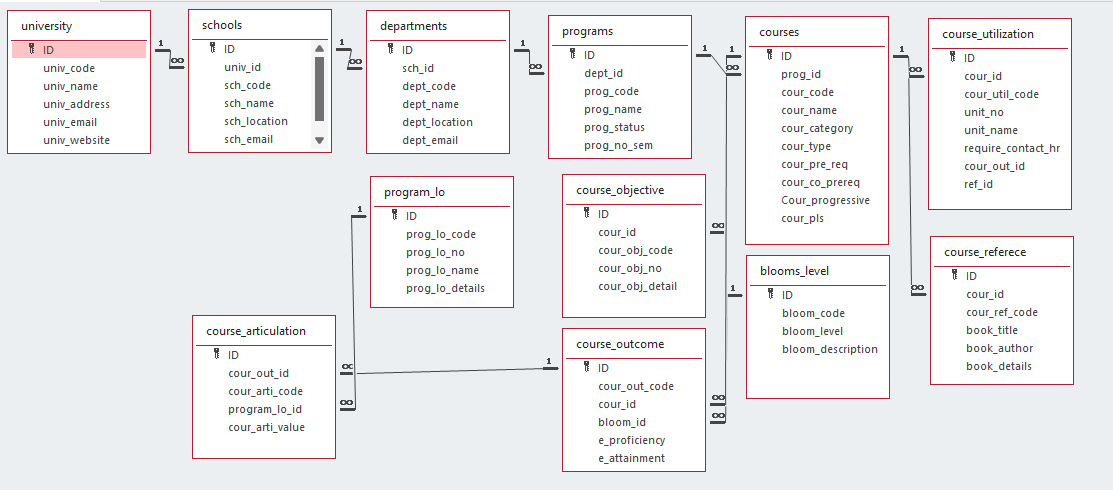
4.Schools 5.Department 6.Programs 7.Courses

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:**

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:**department\_data.txt
* **Function/method name**
  + **Create:**arka\_department\_create
  + **Update:**arka\_department\_update
  + **Retrieve:**arka\_department\_retrive
  + **Delete:**arka\_department\_delete
  + **Sorting: arka\_department\_insertion\_sorting**
  + **Searching:**arka\_department\_linear\_searching
  + **Storing:**arka\_department\_storing
  + **Comparison(both searching and Sorting)**:
    - For Searching-arka\_department\_Compare\_Searching\_algorithm\_linear
    - For Sorting-arka\_department\_Compare\_Sorting\_algorithm\_insertion
  + **Time Complexity(both searching and Sorting):**
    - For Searching-arka\_department\_complexity\_Searching
    - For Sorting-arka\_department\_complexity\_sorting
  + **Algorithm Details(pseudocode or steps)(both searching and Sorting):**
    - For Searching- arka\_department\_linear\_search\_details
    - For Sorting-arka\_department\_insertion\_sorting\_details
* **File name(for storing the details)**
  + File name to be used is:-department\_data .txt

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

|  |  |
| --- | --- |
| **Field Name** | **Data type** |
| id | integer |
| sch\_id | integer |
| dept\_code | String |



|  |  |
| --- | --- |
| dept\_name | String |
| dept\_location | String |
| dept\_email | String |

Algorithm Details:

(i)Sorting

* You have to provide sorting based on **department\_code,department\_name,department\_email**
* Compare the algorithm you have used with any of the other sorting algorithm:

**Apporach:**

**Insertion Sort**: Builds the sorted portion of the array one element at a time, by repeatedly

taking the next unsorted element and placing it in its correct position within the sorted part of the

list. It is similar to the way you might sort playing cards in your hands.

**Selection Sort:**Repeatedly finds the minimum (or maximum) element from the unsorted part of the

array and swaps it with the first unsorted element. The sorted portion of the array grows from left to right.

**Insertion Sort:** Stable (It preserves the relative order of records with equal keys.

**Selection sort:**Unstable(It doesn’t pressure the relative order of elements with equal keys)

* Display the time complexity of both algorithms.

**Insertion Sort:** Best case(O(n))

Average case(O(n^2))

Worst case(O(n^2))

**Selection Sort:** Best case(O(n^2))

Average case(O(n^2))

Worst case(O(n^2))

* Display the pseudocode/algorithm of the sorting algorithm used by you

1. For i from 1 to department\_count - 1:

- Set current to departments[i].

- Set j to i - 1.

- While j >= 0 and departments[j].dept\_code is greater than current.dept\_code:

- Shift departments[j] to departments[j + 1].

- Decrement j by 1.

- Place current at departments[j + 1].

2. The departments array is now sorted by dept\_code.

(ii)Searching

* You have provide searching based on **dept\_code,dept\_name**
* Compare the algorithm used with any of the other algorithm you have learned

**Approach:**

**Linear Search**: It is an iterative search method that checks each element one by one, starting from the beginning of the list, until it finds the target or reaches the end.

**Binary Search**: It is a divide-and-conquer algorithm that works by repeatedly dividing a sorted list into two halves. It compares the target value with the middle element and eliminates half of the remaining elements in each step.

**Data Requirement:**

**Linear Search**: Can be used on **both sorted and unsorted data**. It does not require any specific ordering of the elements.

**Binary Search**: **Requires sorted data**. The list must be ordered (either ascending or descending) for binary search to function correctly.

* Display the time complexity of both algorithms.

**Linear Search:** Best case(O(1))

Average case(O(n))

Worst case(O(n))

**Selection Sort:** Best case(O(1))

Average case(O(log(n))

Worst case(O(log(n))

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

1. For each department in departments:

- If department.dept\_code matches search\_code:

- Print the department details.

- Return "Department Found".

2. If the end of the departments array is reached without finding a match:

- Print "Department not found".

1. Storing the details in a text file

Storing the details in the text file once details are entered.

**1. Open file** in append mode (fopen("department\_data.txt", "r")).

If the file can't be opened, display an error and exit.

**2. Write department data** (sch\_id,code,name,location,email) to the file using fprintf().

**3. Close the file** using fclose()

Delete the detail from the text file once details are deleted.

1. Open department\_data.txt in read mode.

2. If the file can't be opened, print an error and exit.

3. Loop through all department records (for (int i= 0; i < department\_count; i++)):

4. If `dept\_id` matches the target, then department[i]=department[i+1].

5.If department not found then it prints “Department with id not found”.

6. Close the file.

Update the text file once details are updated.

**1. Open file** in write mode (fopen("department\_data.txt", "r")):

If the file can't be opened, display an error and exit.

**2. Loop through all department records** (for (int i= 0; i < department\_count; i++)):

For each department, write its details (sch\_id,code,name,location,email) to the file using fprintf().

**3. Close the file** using fclose().

**Display success message**: "Department updated successfully.”

# Source Code

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

#define MAX 100 typedef struct {

int id;

int school\_id;

char dept\_code[10]; char dept\_name[50]; char dept\_location[50]; char dept\_email[50];

} Department;

Department departments[MAX]; int department\_count = 0;

const char\* FILE\_NAME = "department\_data.txt"; void arka\_department\_create();

void arka\_department\_update(); void arka\_department\_retrieve(); void arka\_department\_delete(); void arka\_department\_storing();

void arka\_department\_insertion\_sorting(int sort\_option); void arka\_department\_linear\_searching();

void arka\_department\_Compare\_Search\_algorithm\_linear(); void arka\_department\_Compare\_Sorting\_algorithm\_insertion(); void arka\_department\_complexity\_searching();

void arka\_department\_complexity\_sorting(); void arka\_department\_linear\_search\_details();

void arka\_department\_insertion\_sorting\_details();

void load\_from\_file() {

FILE \*file = fopen(FILE\_NAME, "r"); if (file == NULL) {

return; // No file exists yet

}

department\_count = 0;

while (fscanf(file, "%d %d %s %s %s %s\n", &departments[department\_count].id, &departments[department\_count].school\_id, departments[department\_count].dept\_code, departments[department\_count].dept\_name, departments[department\_count].dept\_location, departments[department\_count].dept\_email) != EOF) {

department\_count++;

}

fclose(file);

}

int main()

{

load\_from\_file();

printf(" \*\*\*\*DEPARTMENT\*\*\*\*\*\n"); printf("Choose you option\n");

while(1){

printf("1.Create New department\n"); printf("2.Update department\n"); printf("3.Retrive department Details\n"); printf("4.Delete department Details\n");

printf("5.Sort departments Details(by dept\_code,dept\_name,dept\_email)\n"); printf("6.Search departments Details(by dept\_code,dept\_name )\n"); printf("7.Store department Details\n");

printf("8.Comparision of search with other algorithm(binary search)\n"); printf("9.Comparision of sorting with other algorithm(selection sort)\n"); printf("10.Comparision of Time Complexity of sorting Algorithms used\n"); printf("11.Comparision of Time Complexity of searching Algorithms used\n"); printf("12.Displaying searching Algorithm Details\n");

printf("13.Displaying sorting Algorithm Details\n"); printf("14.Exit Application\n");

printf("Please Select the option.\n"); int option;

scanf("%d",&option); switch (option)

{

case 1:

arka\_department\_create(); break;

case 2:

arka\_department\_update(); break;

case 3:

arka\_department\_retrieve(); break;

case 4:

arka\_department\_delete(); break;

case 5:

int sort\_option;

printf("Enter option for sorting 1. Department Code\n2. Department Name\n3. Department Email\n"); scanf("%d",&sort\_option);

arka\_department\_insertion\_sorting(sort\_option); break;

case 6:

arka\_department\_linear\_searching(); break;

case 7: arka\_department\_storing(); break;

case 8:

arka\_department\_Compare\_Search\_algorithm\_linear(); break;

case 9:

arka\_department\_Compare\_Sorting\_algorithm\_insertion(); break;

case 10: arka\_department\_complexity\_searching(); break;

case 11: arka\_department\_complexity\_sorting(); break;

case 12:

arka\_department\_linear\_search\_details(); break;

case 13:

arka\_department\_insertion\_sorting\_details(); break;

default:

printf("please select correct option"); break;

}

}

}

void arka\_department\_storing() {

FILE \*file = fopen(FILE\_NAME, "w"); if (file == NULL) {

printf("Error opening file!\n"); return;

}

for (int i = 0; i < department\_count; i++) {

fprintf(file, "%d %d %s %s %s %s\n", departments[i].id, departments[i].school\_id, departments[i].dept\_code, departments[i].dept\_name, departments[i].dept\_location, departments[i].dept\_email);

}

fclose(file);

}

void arka\_department\_create()

{

if (department\_count >= MAX) { printf("Department list is full!\n"); return;

}

Department d;

printf("Enter Department ID: "); scanf("%d", &d.id); printf("Enter School ID: "); scanf("%d", &d.school\_id);

printf("Enter Department Code: "); scanf("%s", d.dept\_code); printf("Enter Department Name: "); scanf("%s", d.dept\_name); printf("Enter Department Location: "); scanf("%s", d.dept\_location); printf("Enter Department Email: "); scanf("%s", d.dept\_email);

departments[department\_count++] = d; arka\_department\_storing(); printf("Department created successfully!\n");

}

void arka\_department\_update(){ int id;

printf("Enter Department ID to update: "); scanf("%d", &id);

for (int i = 0; i < department\_count; i++) { if (departments[i].id == id) {

printf("Enter new School ID: "); scanf("%d", &departments[i].school\_id); printf("Enter new Department Code: "); scanf("%s", departments[i].dept\_code); printf("Enter new Department Name: "); scanf("%s", departments[i].dept\_name); printf("Enter new Department Location: "); scanf("%s", departments[i].dept\_location); printf("Enter new Department Email: "); scanf("%s", departments[i].dept\_email);

arka\_department\_storing(); printf("Department updated successfully!\n");

return;

}

}

printf("Department with ID %d not found.\n", id);

}

void arka\_department\_retrieve() { printf("\nList of Departments:\n");

for (int i = 0; i < department\_count; i++) {

printf("ID: %d\nSchool ID: %d\nCode: %s\nName: %s\nLocation: %s\nEmail: %s\n\n", departments[i].id, departments[i].school\_id,

departments[i].dept\_code, departments[i].dept\_name, departments[i].dept\_location, departments[i].dept\_email);

}

}

void arka\_department\_delete(){ int id;

printf("Enter Department ID to delete: "); scanf("%d", &id);

for (int i = 0; i < department\_count; i++) { if (departments[i].id == id) {

for (int j = i; j < department\_count - 1; j++) { departments[j] = departments[j + 1];

}

department\_count--; arka\_department\_storing(); printf("Department deleted successfully!\n"); return;

}

}

printf("Department with ID %d not found.\n", id);

}

void arka\_department\_linear\_searching(){ int option;

char search\_term[50];

printf("Search by:\n1. Department Code\n2. Department Name\nEnter choice: "); if (scanf("%d", &option) != 1) {

printf("Invalid input for search option.\n"); while (getchar() != '\n');

return;

}

switch (option) { case 1:

printf("Enter Department Code to search: "); if (scanf("%s", search\_term) != 1) {

printf("Invalid input for Department Code.\n"); while (getchar() != '\n');

return;

}

for (int i = 0; i < department\_count; i++) {

if (strcmp(departments[i].dept\_code, search\_term) == 0) { printf("\nDepartment Found:\n");

printf("ID: %d\n", departments[i].id);

printf("School ID: %d\n", departments[i].school\_id); printf("Code: %s\n", departments[i].dept\_code); printf("Name: %s\n", departments[i].dept\_name); printf("Location: %s\n", departments[i].dept\_location); printf("Email: %s\n\n", departments[i].dept\_email); return;

}}

printf("Department with code '%s' not found.\n", search\_term); break;

case 2:

printf("Enter Department Name to search: "); if (scanf("%s", search\_term) != 1) {

printf("Invalid input for Department Name.\n"); while (getchar() != '\n');

return;

}

for (int i = 0; i < department\_count; i++) {

if (strcmp(departments[i].dept\_name, search\_term) == 0) { printf("\nDepartment Found:\n");

printf("ID: %d\n", departments[i].id);

printf("School ID: %d\n", departments[i].school\_id); printf("Code: %s\n", departments[i].dept\_code); printf("Name: %s\n", departments[i].dept\_name); printf("Location: %s\n", departments[i].dept\_location); printf("Email: %s\n\n", departments[i].dept\_email); return;

}

}

printf("Department with name '%s' not found.\n", search\_term); break;

default:

printf("Invalid search option!\n");

}

}

void arka\_department\_insertion\_sorting(int sort\_option){ for (int i = 1; i < department\_count; i++) {

Department key = departments[i]; int j = i - 1;

while (j >= 0) {

int compare = 0; switch (sort\_option) {

case 1:

compare = strcmp(departments[j].dept\_code, key.dept\_code); break;

case 2:

compare = strcmp(departments[j].dept\_name, key.dept\_name); break;

case 3:

compare = strcmp(departments[j].dept\_email, key.dept\_email); break;

}

if (compare > 0) {

departments[j + 1] = departments[j]; j--;

} else {

break;

}

}

departments[j + 1] = key;

}

printf("Departments sorted!\n"); arka\_department\_retrieve();

}

void arka\_department\_Compare\_Search\_algorithm\_linear() { char search\_code[10];

printf("Enter Department Code to search: "); scanf("%s", search\_code);

int left = 0, right = department\_count - 1; while (left <= right) {

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

int compare = strcmp(departments[mid].dept\_code, search\_code);

if (compare == 0) {

// Department found printf("\nDepartment Found:\n"); printf("ID: %d\n", departments[mid].id);

printf("School ID: %d\n", departments[mid].school\_id); printf("Code: %s\n", departments[mid].dept\_code); printf("Name: %s\n", departments[mid].dept\_name); printf("Location: %s\n", departments[mid].dept\_location); printf("Email: %s\n\n", departments[mid].dept\_email); return;

} else if (compare < 0) {

left = mid + 1; // Search in the right half

} else {

right = mid - 1; // Search in the left half

}

}

printf("Department with code '%s' not found.\n", search\_code);

}

void arka\_department\_Compare\_Sorting\_algorithm\_insertion(int sort\_option) {

//Comparing by selection sorting

for (int i = 0; i < department\_count - 1; i++) { int min\_idx = i;

for (int j = i + 1; j < department\_count; j++) { int compare = 0;

switch (sort\_option) {

case 1: // Sort by dept\_code

compare = strcmp(departments[j].dept\_code, departments[min\_idx].dept\_code);

break;

case 2: // Sort by dept\_name

compare = strcmp(departments[j].dept\_name, departments[min\_idx].dept\_name);

break;

case 3: // Sort by dept\_email

compare = strcmp(departments[j].dept\_email, departments[min\_idx].dept\_email);

break;

}

if (compare < 0) { min\_idx = j;}}

// Swap the found minimum element with the first element if (min\_idx != i) {

Department temp = departments[i]; departments[i] = departments[min\_idx]; departments[min\_idx] = temp;

}

printf("Departments sorted!\n"); arka\_department\_retrieve();

}

}

void arka\_department\_complexity\_sorting() { printf("Time Complexity Analysis:\n"); printf("Insertion Sort: O(n^2)\n"); printf("Selection Sort: O(n^2)\n");

}

void arka\_department\_complexity\_searching(){ printf("Time Complexity Analysis:\n"); printf("Linear Search: O(n)\n");

printf("Binary Search: O(log n)\n");

}

void arka\_department\_linear\_search\_details(){ printf("Pseudocode for Linear Search:\n"); printf("For each department in departments:\n");

printf("1. If department.dept\_code matches the search\_code:\n- Print the department details.\n - Return Department Found.");

printf("2. If the end of the departments array is reached without finding a match:\n- Print Department not found.\n\n");

printf("Pseudocode for Binary Search:\n");

printf("1. Initialize left to 0 and right to department\_count - 1.\n"); printf("2. While left is less than or equal to right:\n");

printf(" a. Set middle to (left + right) / 2.\n");

printf(" b. If departments[middle].dept\_code matches search\_code:\n"); printf(" - Print the department details.\n");

printf(" - Return \"Department Found.\"\n");

printf(" c. If departments[middle].dept\_code is less than search\_code:\n"); printf(" - Set left to middle + 1.\n");

printf(" d. Else (if departments[middle].dept\_code is greater than search\_code):\n"); printf(" - Set right to middle - 1.\n");

printf("3. If no match is found after exiting the loop:\n"); printf(" - Print \"Department not found.\"\n");

}

void arka\_department\_insertion\_sorting\_details(){ printf("Pseudocode for Insertion Sort:\n"); printf("1. For i from 1 to department\_count - 1:\n"); printf(" a. Set current to departments[i].\n"); printf(" b. Set j to i - 1.\n");

printf(" c. While j >= 0 and departments[j].dept\_code is greater than current.dept\_code:\n");

printf(" - Shift departments[j] to departments[j + 1].\n"); printf(" - Decrement j by 1.\n");

printf(" d. Place current at departments[j + 1].\n");

printf("2. The departments array is now sorted by dept\_code.\n");

}

void arka\_department\_selection\_sorting\_details(){ printf("Pseudocode for Selection Sort:\n"); printf("1. For i from 0 to department\_count - 2:\n"); printf(" a. Set min\_index to i.\n");

printf(" b. For j from i + 1 to department\_count - 1:\n"); printf(" - If departments[j].dept\_code is less than departments[min\_index].dept\_code:\n");

printf(" - Set min\_index to j.\n"); printf(" c. If min\_index is not i:\n");

printf(" - Swap departments[i] with departments[min\_index].\n"); printf("2. The departments array is now sorted by dept\_code.\n");}

### Comparison of Searching Algorithms

Search Method:

Linear Search:

Checks each element in the list sequentially from the beginning to the end. Data Structure: Works on unsorted or sorted arrays/lists.

Binary Search:

Repeatedly divides the search space in half by comparing the target value with the middle element of the list. Data Structure: Requires the list to be sorted.

Efficiency Linear Search:

Inefficient for large lists because it checks each element individually. Performance degrades as the size of the list increases.

Binary Search:

Efficient for large, sorted lists because it reduces the number of elements to check by half each time. Its efficien is much better for large datasets compared to linear search.

Time Complexity Linear Search:

Worst-case time complexity: O(n)

Linear search may need to examine every element in the list, where n is the number of elements. Binary Search:

Worst-case time complexity: O(log n)

The search space is halved with each step, making binary search much faster for large datasets when compared to linear search.

Space Complexity Linear Search:

Space complexity: O(1)

Linear search only requires a constant amount of space, as it does not need any extra memory other t han for the input list.

Binary Search:

Space complexity: O(1) for iterative approach, but O(log n) for recursive approach (due to recursion stack). The iterative version has constant space complexity, but the recursive version might require additional space due to function call stack.

### Comparison of sorting Algorithms

Insertion Sort:

Builds the sorted portion of the array one element at a time, by repeatedly taking the next unsorted element and placing it in its correct position within the sorted part of the list.

It is similar to the way you might sort playing cards in your hands.

Selection Sort:

Repeatedly finds the minimum (or maximum) element from the unsorted part of the array and swaps it with the first unsorted element. The sorted portion of the array grows from left to right.

Time Complexity Insertion Sort:

Best-case time complexity: O(n)

Occurs when the array is already sorted (only requires a linear scan). Average-case time complexity: O(n²)

For a random unsorted array, each insertion can take O(n) time. Worst-case time complexity: O(n²)

Occurs when the array is sorted in reverse order (each insertion requires shifting all elements).

Selection Sort:

Best-case time complexity: O(n²)

Even if the array is already sorted, Selection Sort always performs the same number of comparisons and swaps.

Average-case time complexity: O(n²) Consistently takes O(n²) comparisons. Worst-case time complexity: O(n²)

No variation in the number of comparisons, even in the worst case.

Space Complexity Insertion Sort:

Space complexity: O(1)

Insertion Sort is an in-place sorting algorithm, meaning it doesn't require additional memory beyond the original input array.

Selection Sort:

Space complexity: O(1)

Like Insertion Sort, Selection Sort is an in-place sorting algorithm that requires a constant amount of space for swaps.

Stability Insertion Sort:

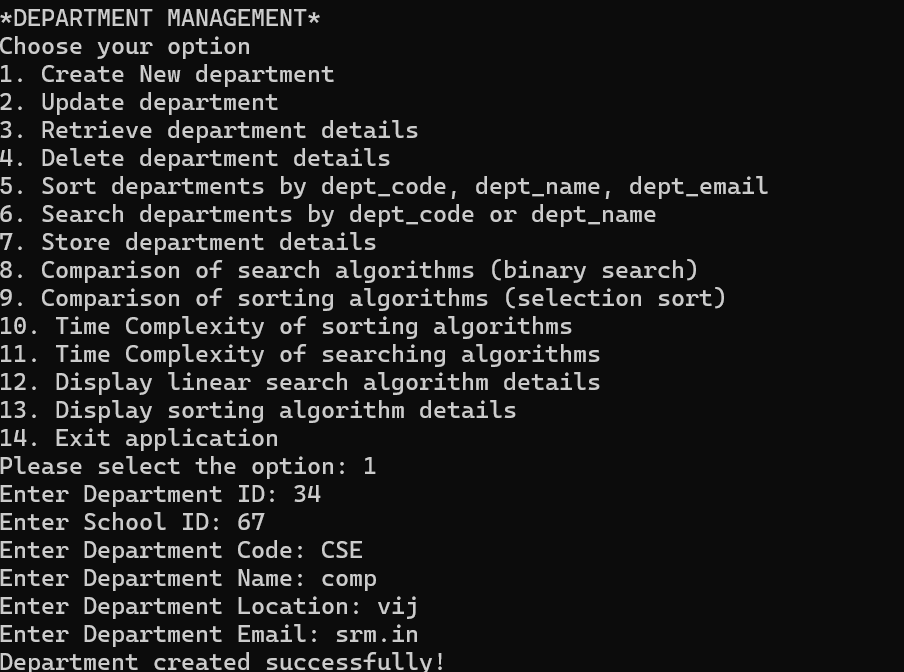
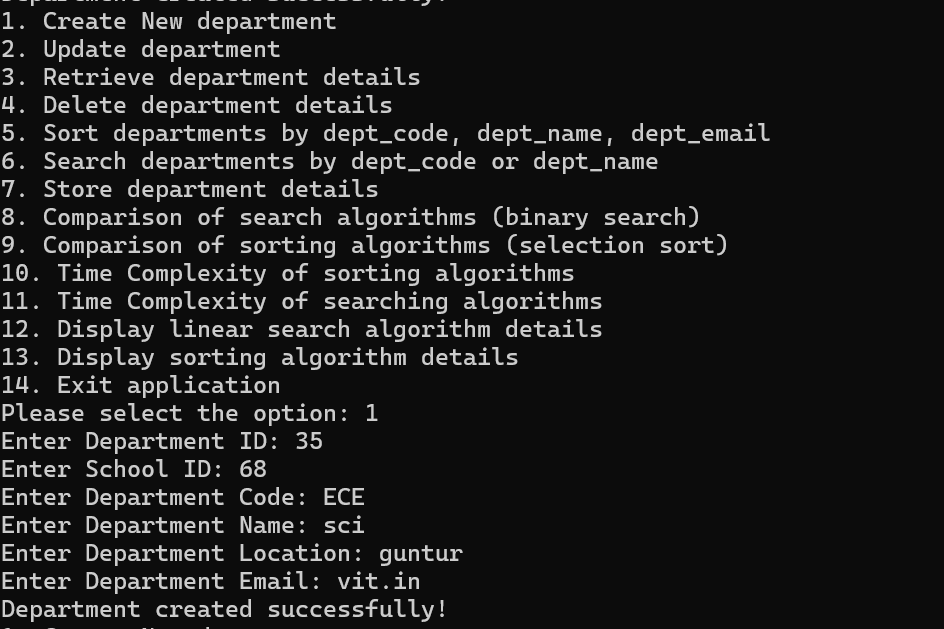
Stable: If two elements have the same value, they remain in the same relative order in the sorted array.

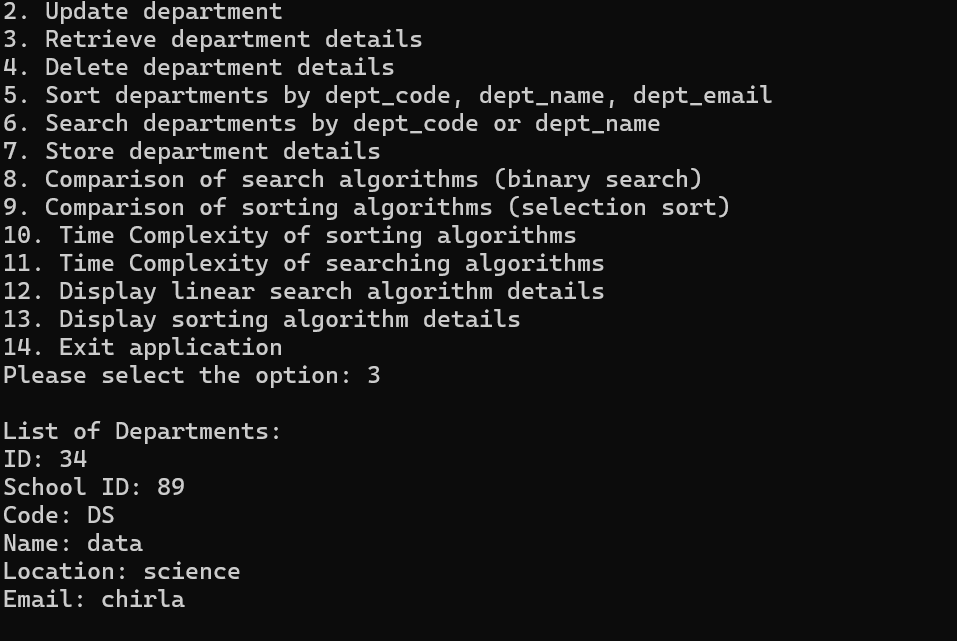
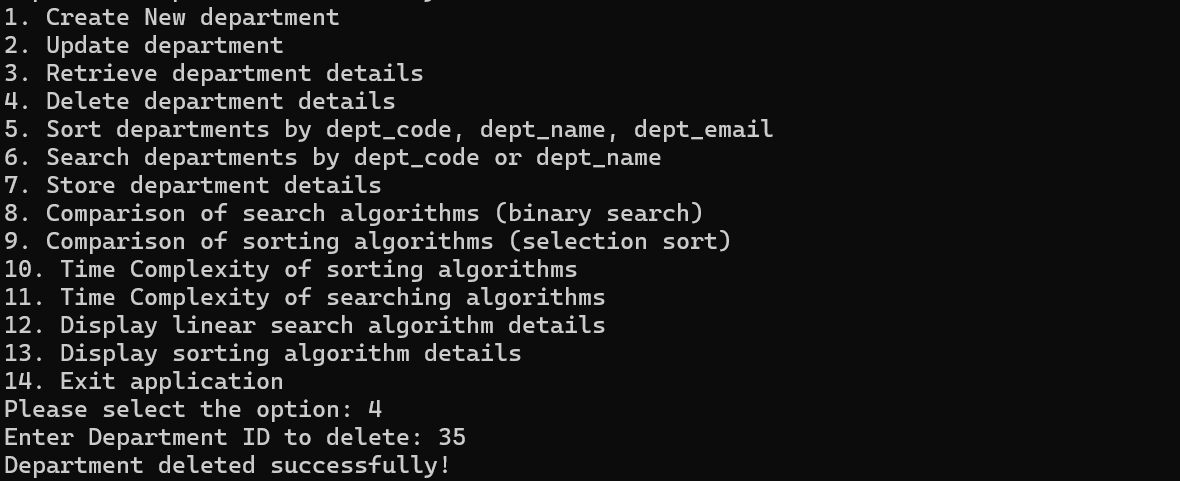
Selection Sort:

Not stable: If two elements have the same value, their relative order may change after sorting.

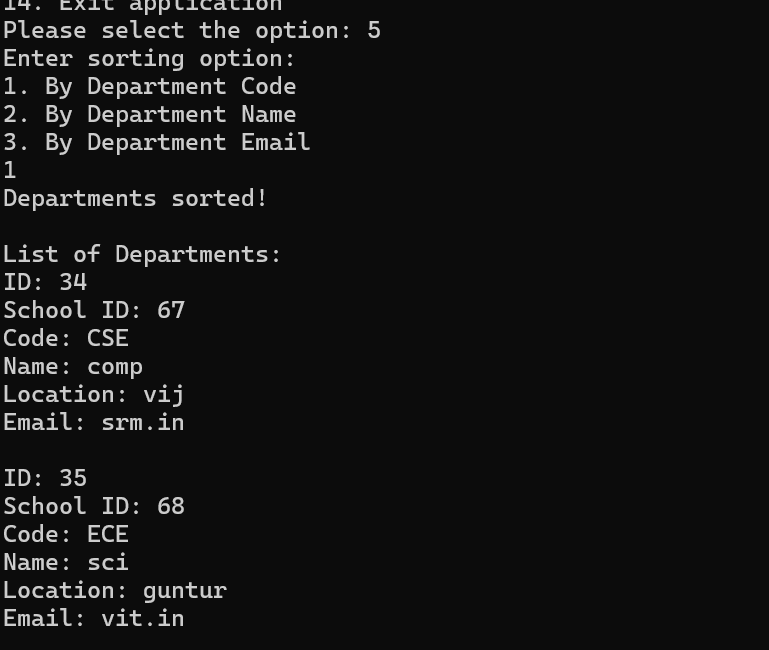
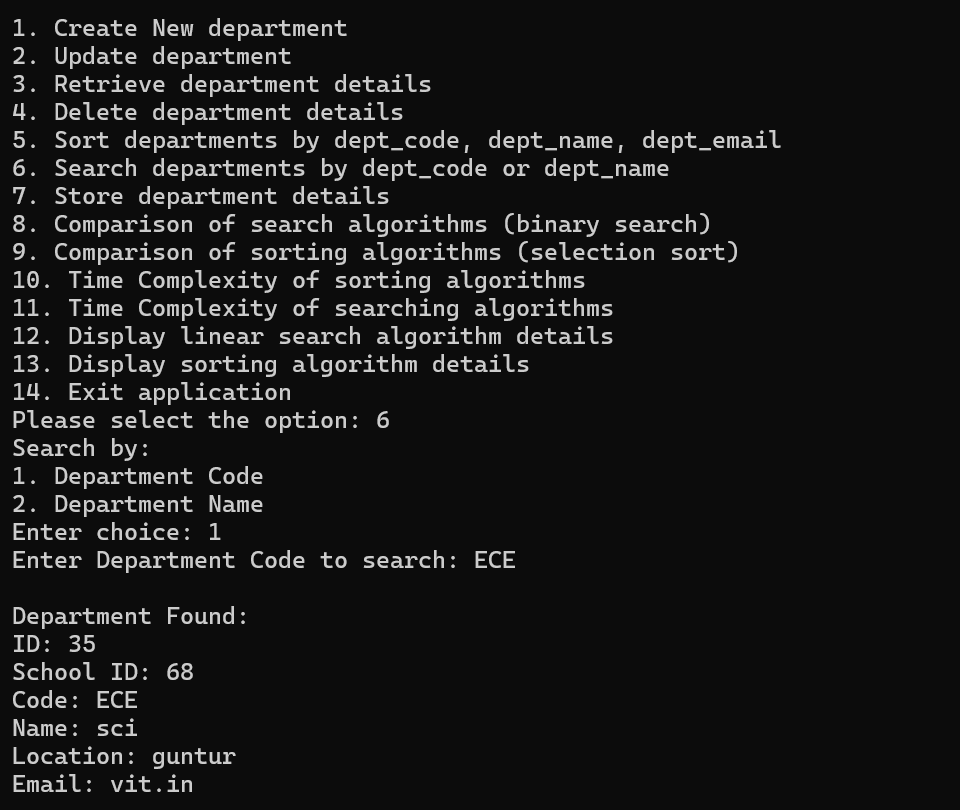
# Screen Shots

1.Create



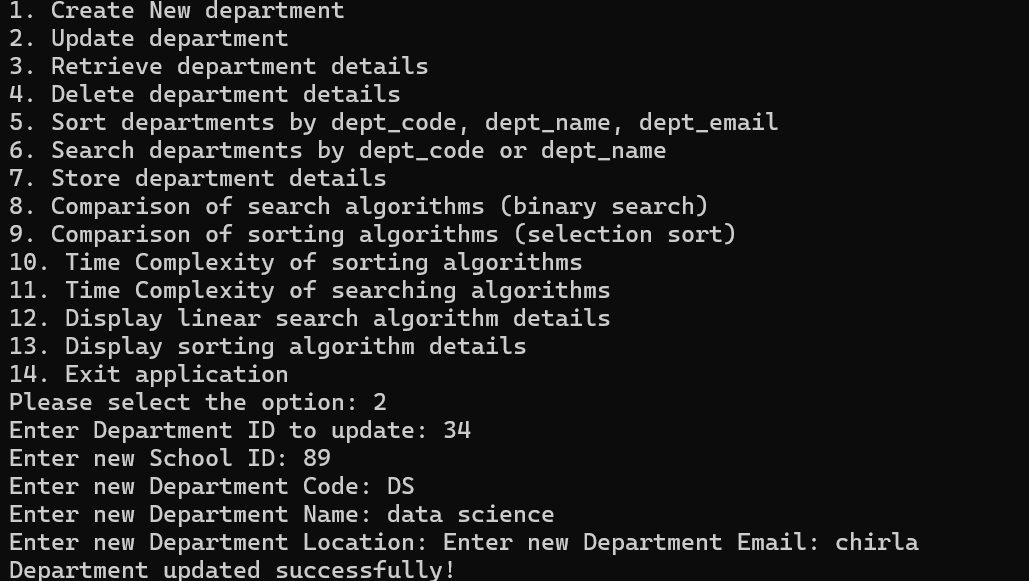
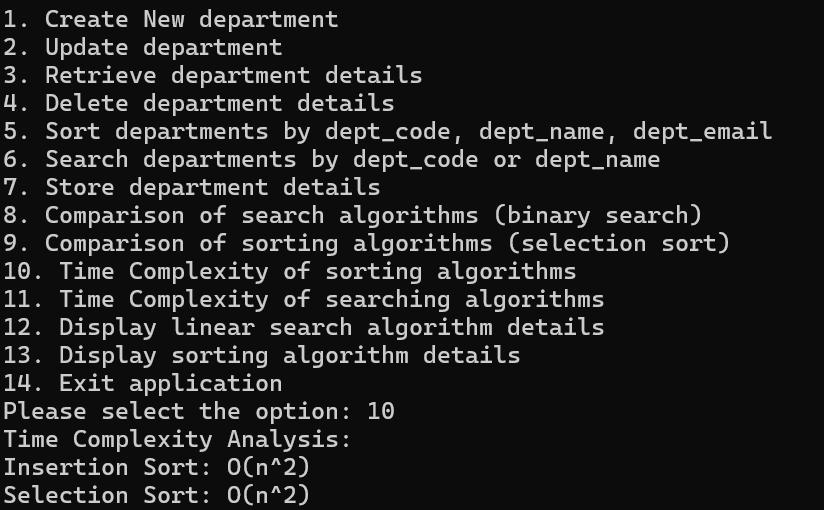
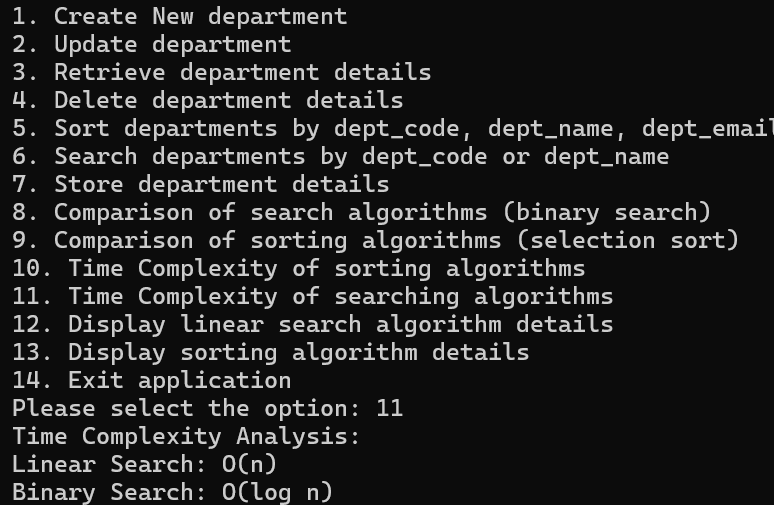
 2.Retrieve

3.Delete



5.Sorting using Insertion sort

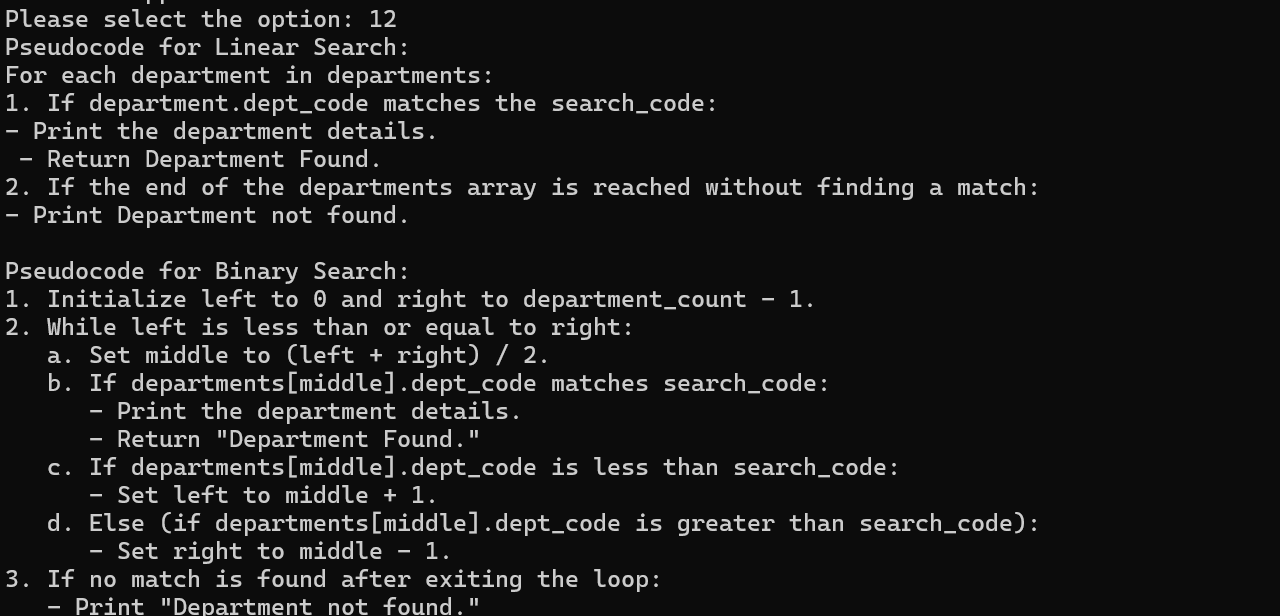
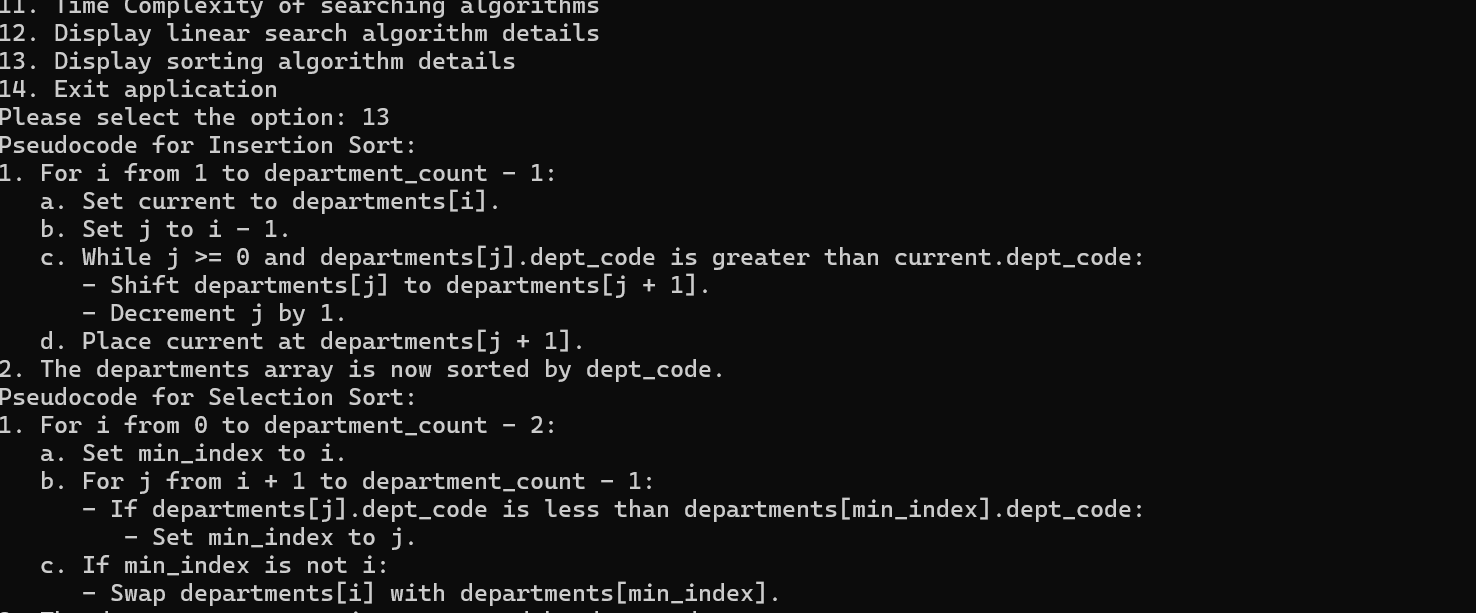
6.Linear Search



7.Time Complexity for Searching

8.Time Complexity of Searching

9.Update

****

10.Display Pseudocodes for Sorting

11.Dispaly Pseudocode for Searching

**Conclusion**

##### This project implements a Department Management System that allows users to manage department data through various operations such as creating, updating, deleting, retrieving, sorting, and searching.

The system uses basic sorting and searching algorithms, specifically Insertion Sort

##### and Selection Sort, for organizing department records by attributes like department code, name, or email.

For searching, it employs Linear Search and compares it with Binary Search once the departments are sorted, highlighting the efficiency improvements of binary search on sorted data.

##### The department data is stored in a text file, ensuring persistence across sessions. The project also incorporates time complexity analysis to compare the performance of these algorithms, with a focus on how sorting impacts search efficiency. Overall, this project demonstrates key concepts in Data Structures and Algorithms while providing a practical, real-world

application of sorting and searching techniques.