**OBE IMPLEMENTATION:UNIVERSITY SETTING**

***by***

# **Your\_name[Reg No]**

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



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# **SRM UNIVERSITY AP::AMARAVATI**

**INDEX**

**Introduction**

Project Modules:

**Architecture Diagram**

**Module Description**

Programming Details naming conventions to be used: Field/table details:(eg university)[you consider you module ] 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 University (herewith considered as SRM-AP) is going to implement OBE(Outcome Based Education) in their university and you assigned in the project to develop an application with any programming Language you are well versed and you were supposed to do searching and sorting using learned algorithms,comparing your sorting algorithm with any one of existing algorithm,displaying the time complexity of both algorithms and explaining advantages and disadvantages of the algorithm.

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

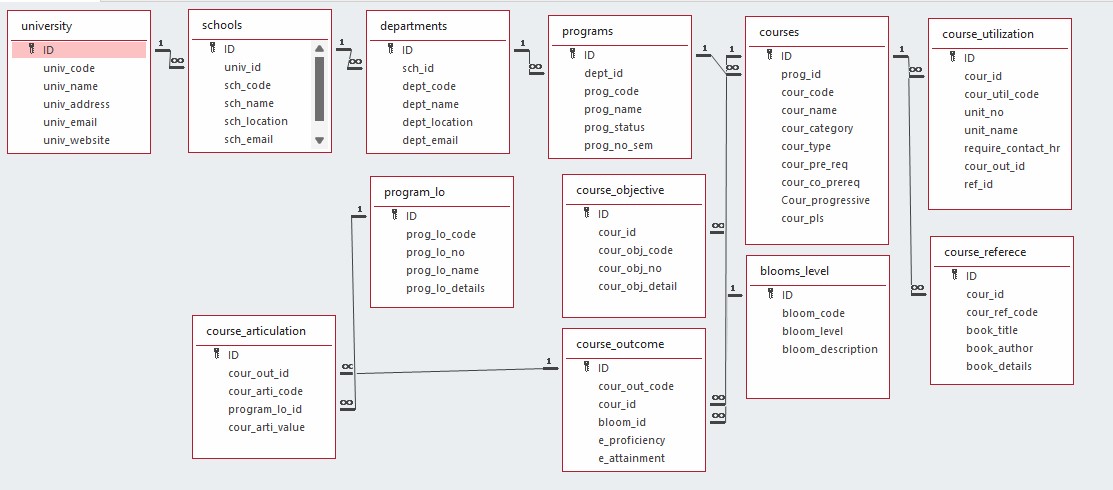
10.Course Articulation matrix Setting

11.Course Utilization Setting

12.Course Reference Setting.

# Architecture Diagram

***\*highlight your module***



# Module Description

**Module Name:**Eg)university[you have to consider module given to you] **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:**yourregno\_module\_name ● **Function/method name**

○ **Create:**yourregno\_module\_name\_create

○ **Update:**yourregno\_module\_name\_update

○ **Retrieve:**yourregno\_module\_name\_retrive

○ **Delete:**yourregno\_module\_name\_delete

○ **Sorting:**yourregno\_module\_name\_youralgorithm name

○ **Searching:**yourregno\_module\_name\_youralgorithm name

○ **Storing:**yourregno\_module\_name\_storing ○ **Comparison(both searching and Sorting)**:

■ For Searchingyourregno\_module\_name\_Compare\_Search\_youralgorithm name

■ For Sortingyourregno\_module\_name\_Compare\_sorting\_youralgorithm name ○ **Time Complexity(both searching and Sorting):**

■ For Searching-yourregno\_module\_name\_complexity\_Search

■ For Sorting-yourregno\_module\_name\_compexity\_sorting

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

■ For Searchingyourregno\_module\_name\_your\_search\_algorithmname\_details

■ For Sortingyourregno\_module\_name\_your\_sort\_algorithmname\_details ● **File name(for storing the details)**

○ File name to be used is:-university\_setting .txt

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

|  |  |
| --- | --- |
| **Field Name** | **Data type** |
| id | integer |
| univ\_code | String |
| univ\_name | String |
| univ\_address | String |
| univ\_email | String |
| univ\_website | String |

Algorithm Details:

(i)Sorting

* You have to provide sorting based on **university code ,university\_name , university\_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 **university code,university\_name,university\_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>

#include <time.h>

#define max 100

struct University {

int univ\_Id;

char univ\_code[10];

char univ\_name[20];

char univ\_address[30];

char univ\_email[15];

char univ\_website[10];

} typedef uni;

uni university[max];

int university\_count = 0;

const char \*filename = "university\_data.txt"; // Text file to store data

// Function to save data to a text file

void save\_to\_file() {

FILE \*file = fopen(filename, "w"); // Open in write mode (text)

if (file == NULL) {

printf("Error opening file for writing.\n");

return;

}

fprintf(file, "%d\n", university\_count); // Write count of universities

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

fprintf(file, "%d %s %s %s %s %s\n", university[i].univ\_Id,

university[i].univ\_code, university[i].univ\_name,

university[i].univ\_address, university[i].univ\_email,

university[i].univ\_website); // Write each university's data

}

fclose(file);

printf("Data saved to file successfully.\n");

}

// Function to load data from a text file

void V\_Dart\_university\_load\_from\_file() {

FILE \*file = fopen(filename, "r"); // Open in read mode (text)

if (file == NULL) {

printf("No previous data found. Starting fresh.\n");

return;

}

fscanf(file, "%d", &university\_count); // Read the count of universities

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

fscanf(file, "%d %s %s %s %s %s", &university[i].univ\_Id,

university[i].univ\_code, university[i].univ\_name,

university[i].univ\_address, university[i].univ\_email,

university[i].univ\_website); // Read each university's data

}

fclose(file);

printf("Data loaded from file successfully.\n");

}

void getuniversitydetails(uni \*u) {

printf("Enter University ID: ");

scanf("%d", &u->univ\_Id);

printf("Enter University Code: ");

scanf("%s", u->univ\_code);

printf("Enter University Name: ");

scanf("%s", u->univ\_name);

printf("Enter University Address: ");

scanf("%s", u->univ\_address);

printf("Enter University Email: ");

scanf("%s", u->univ\_email);

printf("Enter University Website: ");

scanf("%s", u->univ\_website);

}

void showuniversitydetails(uni \*u) {

printf("%d %s %s %s %s %s\n", u->univ\_Id, u->univ\_code, u->univ\_name, u->univ\_address, u->univ\_email, u->univ\_website);

}

// Bubble Sort for Sorting the universities by univ\_Id

void V\_Dart\_university\_bubble\_sort() {

int i, j;

uni temp;

for (i = 0; i < university\_count - 1; i++) {

for (j = 0; j < university\_count - 1 - i; j++) {

if (university[j].univ\_Id > university[j + 1].univ\_Id) {

temp = university[j];

university[j] = university[j + 1];

university[j + 1] = temp;

}

}

}

}

// Quick Sort for Sorting the universities by univ\_Id

void V\_Dart\_university\_quick\_sort(int low, int high) {

if (low < high) {

int pivot = university[high].univ\_Id;

int i = low - 1;

for (int j = low; j < high; j++) {

if (university[j].univ\_Id < pivot) {

i++;

uni temp = university[i];

university[i] = university[j];

university[j] = temp;

}

}

uni temp = university[i + 1];

university[i + 1] = university[high];

university[high] = temp;

int pi = i + 1;

V\_Dart\_university\_quick\_sort(low, pi - 1);

V\_Dart\_university\_quick\_sort(pi + 1, high);

}

}

// Linear Search for a university by univ\_Id

int V\_Dart\_university\_linear\_search(int id) {

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

if (university[i].univ\_Id == id) {

return i; // University found at index i

}

}

return -1; // University not found

}

// Binary Search for a university by univ\_Id

int V\_Dart\_university\_binary\_search(int id) {

int low = 0, high = university\_count - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (university[mid].univ\_Id == id) {

return mid;

}

if (university[mid].univ\_Id < id) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1; // University not found

}

void V\_Dart\_university\_create() {

if (university\_count >= max) {

printf("University List is Full\n");

return;

}

uni u;

getuniversitydetails(&u);

university[university\_count++] = u;

save\_to\_file(); // Save data after adding

printf("University Created Successfully!\n");

}

void V\_Dart\_university\_delete() {

if (university\_count == 0) {

printf("University List not found\n");

return;

}

int id;

printf("Enter University ID to Delete: ");

scanf("%d", &id);

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

if (university[i].univ\_Id == id) {

for (int j = i; j < university\_count - 1; j++) {

university[j] = university[j + 1];

}

university\_count--;

save\_to\_file(); // Save data after deletion

printf("University deleted successfully!\n");

return;

}

}

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

}

void V\_Dart\_university\_update() {

if (university\_count == 0) {

printf("University List not found\n");

return;

}

int id;

printf("Enter University ID to Update: ");

scanf("%d", &id);

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

if (university[i].univ\_Id == id) {

printf("Enter details of ID %d again\n", id);

getuniversitydetails(&university[i]);

save\_to\_file(); // Save data after updating

printf("University updated successfully!\n");

return;

}

}

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

}

void V\_Dart\_university\_retrieve() {

if (university\_count == 0) {

printf("University List is Empty\n");

return;

}

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

showuniversitydetails(&university[i]);

}

}

void V\_Dart\_university\_display\_time\_complexities() {

printf("\nTime Complexities:\n");

printf("1. Bubble Sort: O(n^2)\n");

printf("2. Quick Sort: O(n log n) (on average)\n");

printf("3. Linear Search: O(n)\n");

printf("4. Binary Search: O(log n) (requires sorted data)\n");

}

int main() {

V\_Dart\_university\_load\_from\_file(); // Load data at the start of the program

int choice;

do {

printf("\n--- University Management System ---\n");

printf("1. Create University\n");

printf("2. Delete University\n");

printf("3. Update University\n");

printf("4. Retrieve University List\n");

printf("5. Sort Universities (Bubble Sort)\n");

printf("6. Sort Universities (Quick Sort)\n");

printf("7. Search University (Linear Search)\n");

printf("8. Search University (Binary Search)\n");

printf("9. Display Time Complexities\n");

printf("10. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

V\_Dart\_university\_create();

break;

case 2:

V\_Dart\_university\_delete();

break;

case 3:

V\_Dart\_university\_update();

break;

case 4:

V\_Dart\_university\_retrieve();

break;

case 5: {

clock\_t start = clock();

V\_Dart\_university\_bubble\_sort();

clock\_t end = clock();

double time\_taken = ((double)end - start) / CLOCKS\_PER\_SEC;

printf("Bubble Sort Time: %f seconds\n", time\_taken);

break;

}

case 6: {

clock\_t start = clock();

V\_Dart\_university\_quick\_sort(0, university\_count - 1);

clock\_t end = clock();

double time\_taken = ((double)end - start) / CLOCKS\_PER\_SEC;

printf("Quick Sort Time: %f seconds\n", time\_taken);

break;

}

case 7: {

int id;

printf("Enter University ID to search: ");

scanf("%d", &id);

clock\_t start = clock();

int result = V\_Dart\_university\_linear\_search(id);

clock\_t end = clock();

double time\_taken = ((double)end - start) / CLOCKS\_PER\_SEC;

if (result != -1)

printf("University found at index: %d\n", result);

else

printf("University not found\n");

printf("Linear Search Time: %f seconds\n", time\_taken);

break;

}

case 8: {

int id;

printf("Enter University ID to search: ");

scanf("%d", &id);

clock\_t start = clock();

int result = V\_Dart\_university\_binary\_search(id);

clock\_t end = clock();

double time\_taken = ((double)end - start) / CLOCKS\_PER\_SEC;

if (result != -1)

printf("University found at index: %d\n", result);

else

printf("University not found\n");

printf("Binary Search Time: %f seconds\n", time\_taken);

break;

}

case 9:

V\_Dart\_university\_display\_time\_complexities();

break;

case 10:

printf("Exiting...\n");

break;

default:

printf("Invalid choice! Please enter a valid option.\n");

}

} while (choice != 10);

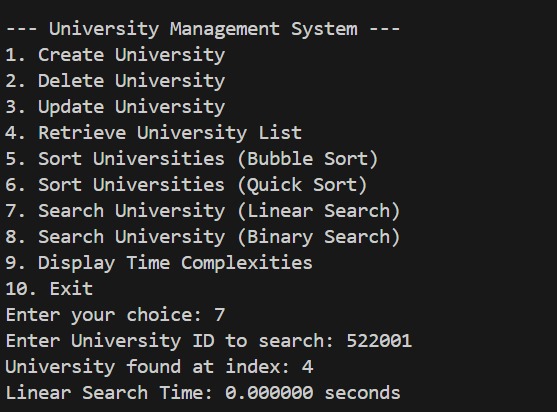
return 0;

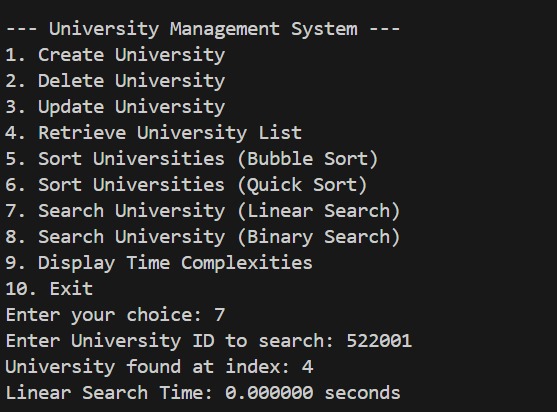
}

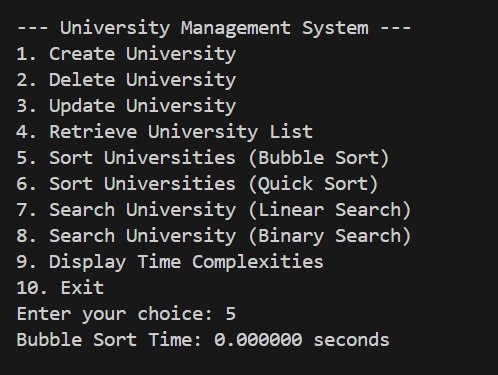
* Comparison of Sorting Algorithms
* // Quick Sort for Sorting the universities by univ\_Id
* void V\_Dart\_university\_quick\_sort(int low, int high) {
* if (low < high) {
* int pivot = university[high].univ\_Id;
* int i = low - 1;
* for (int j = low; j < high; j++) {
* if (university[j].univ\_Id < pivot) {
* i++;
* uni temp = university[i];
* university[i] = university[j];
* university[j] = temp;
* }
* }
* uni temp = university[i + 1];
* university[i + 1] = university[high];
* university[high] = temp;
* int pi = i + 1;
* V\_Dart\_university\_quick\_sort(low, pi - 1);
* V\_Dart\_university\_quick\_sort(pi + 1, high);
* }
* }
* Comparison of Searching Algorithms

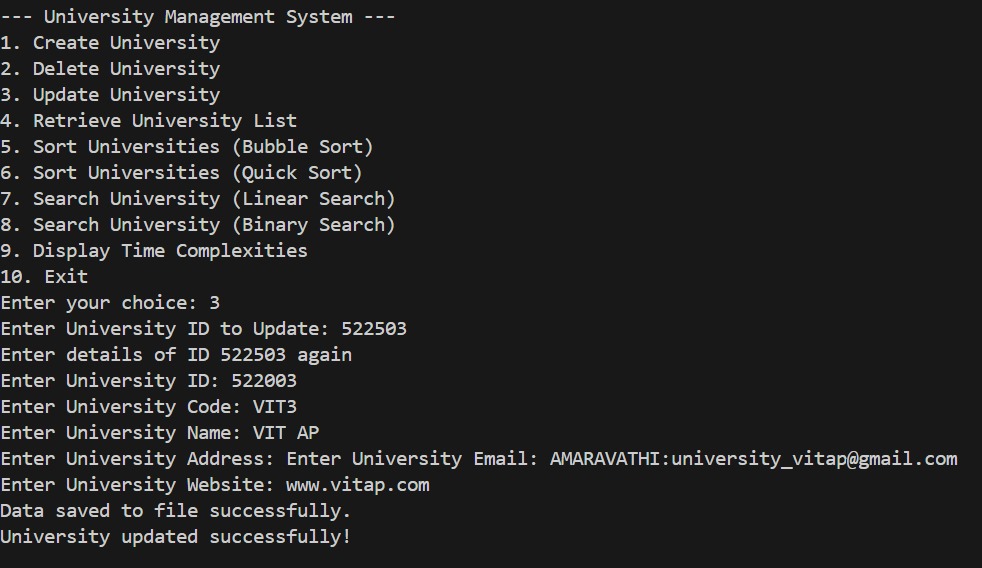
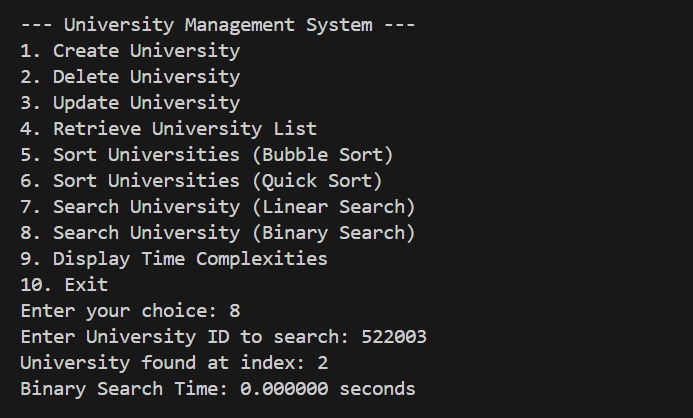
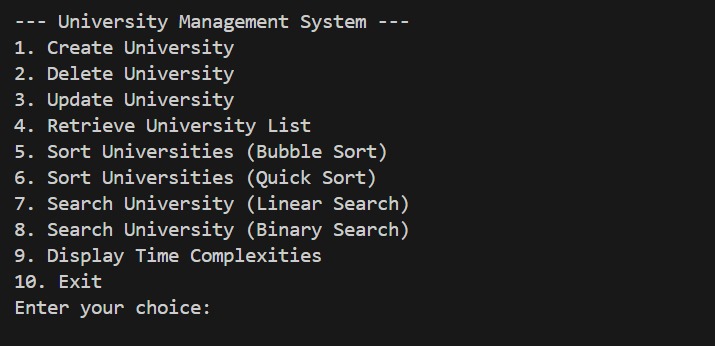
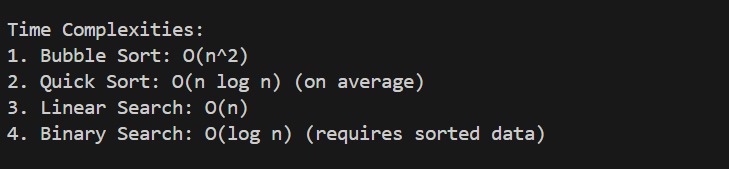
Linear Search for a university by univ\_Id

* int V\_Dart\_university\_linear\_search(int id) {
* for (int i = 0; i < university\_count; i++) {
* if (university[i].univ\_Id == id) {
* return i; // University found at index i
* }
* }
* return -1; // University not found
* }
* Binary Search for a university by univ\_Id
* int V\_Dart\_university\_binary\_search(int id) {
* int low = 0, high = university\_count - 1;
* while (low <= high) {
* int mid = low + (high - low) / 2;
* if (university[mid].univ\_Id == id) {
* return mid;
* }
* if (university[mid].univ\_Id < id) {
* low = mid + 1;
* } else {
* high = mid - 1;
* }
* }
* return -1; // University not found
* }
* Output Screenshorts







# Conclusion

The University Management System effectively demonstrates the use of basic algorithms and data manipulation strategies in the school's and educational institution's data organization and management. By integrating sorting algorithms like Bubble Sort and Quick Sort with searching algorithms like Linear and Binary Search, it proposes efficient and scalable solutions for data retrieval and management. Data persistence improves with the system's ability to save and load the data from a file, thus providing reliability and usability following program termination.