OBE IMPLEMENTATION: UNIVERSITY SETTING

By

**G. ABHINAY [AP23110011114]**

**T. SWETHA PRIYA [AP23110011116]**

**A. MANOJ [AP23110011118]**

**G. VISHNU VARDHAN [AP23110011138]**

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



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY AP:: AMARAVATI

**INDEX**

**Introduction**

Project Module

**Architecture Diagram**

**Module Description**

Programming Details naming conventions to be used:

Field/table details: University

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**

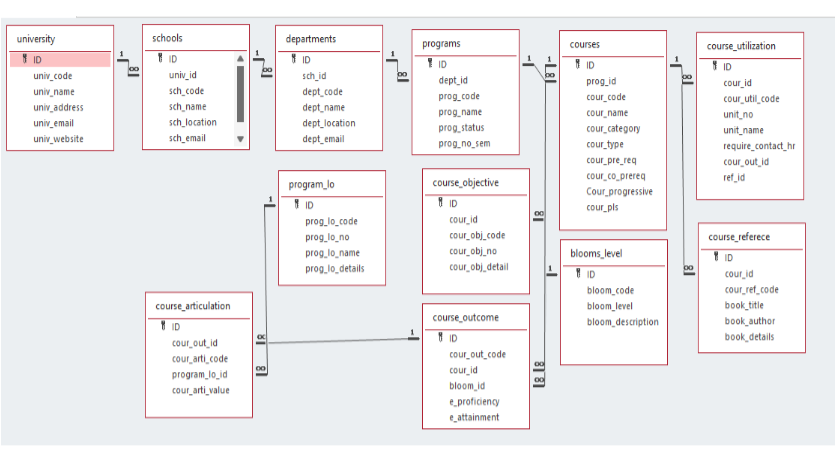
**Introduction**

Our university (herewith considered as SRM-AP) is going to implement OBE (Outcome Based Education) in the university and assigned in the project to develop an application with any programming Language and well versed and supposed to do searching and sorting using learned algorithms, comparing sorting algorithm with anyone of existing algorithm, displaying the time complexity of both algorithms and explaining

advantages and disadvantages of the algorithm.

**Project Module:** University

**Architecture Diagram**

****

**Module Description**

Module Name: university

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.

Programming Details naming conventions to be used:

* **File name:** university
* **Function/method name**
* **Create:** university\_create
* **Update:** university\_update
* **Retrieve:** university\_retrieve
* **Delete:** university\_delete
* **Sorting:** university\_mergesort

university\_quicksort

* **Searching:**university\_linearsearch university\_binary search
* **Storing:** university\_storing
* **Comparison (both searching and sorting):**
* For Searching-

university\_compare\_linearsearch

university\_compare\_binarysearch

* For Sorting-

university\_compare\_quicksort

university\_compare\_mergesort

* **Time complexity (both searching and sorting):**
* For searching-

university\_complexity\_linearsearch

university\_complexity\_binarysearch

* For sorting-

university\_complexity\_mergesort

university\_complexity\_quicksort

* **Algorithm Details (pseudocode or steps) (both searching and sorting):**
* For searching-

university\_algo\_linearsearch

university\_algo\_binarysearch

* For sorting-

university\_algo\_quicksort

university\_algo\_mergesort

* **File name (for storing the details):** university\_setting.txt

Field/table details: university

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

Algorithm Details:

1. Sorting : We used quick sort and merge sort for this code
2. Searching : We used linear search and binary search for this code
3. 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\_UNIVERSITIES 100**

**#define MAX\_STRING\_LENGTH 100**

**// Define the structure for university records**

**typedef struct {**

**int id;**

**char univ\_code[MAX\_STRING\_LENGTH];**

**char univ\_name[MAX\_STRING\_LENGTH];**

**char univ\_address[MAX\_STRING\_LENGTH];**

**char univ\_email[MAX\_STRING\_LENGTH];**

**char univ\_website[MAX\_STRING\_LENGTH];**

**} University;**

**// Function prototypes**

**void university\_create(University universities[], int \*count);**

**void university\_update(University universities[], int count);**

**void university\_retrieve(University universities[], int count);**

**void university\_delete(University universities[], int \*count);**

**void university\_mergesort(University universities[], int left, int right);**

**void merge(University universities[], int left, int mid, int right);**

**void university\_quicksort(University universities[], int low, int high);**

**int partition(University universities[], int low, int high);**

**int university\_linearsearch(University universities[], int count, int id);**

**int university\_binarysearch(University universities[], int left, int right, int id);**

**void university\_storing(University universities[], int count);**

**void university\_compare\_linearsearch(University universities[], int count, int id);**

**void university\_compare\_binarysearch(University universities[], int count, int id);**

**void university\_compare\_quicksort(University universities[], int count);**

**void university\_compare\_mergesort(University universities[], int count);**

**void university\_complexity\_linearsearch(int n);**

**void university\_complexity\_binarysearch(int n);**

**void university\_complexity\_mergesort(int n);**

**void university\_complexity\_quicksort(int n);**

**void writeToFile(University universities[], int count);**

**int main() {**

**University universities[MAX\_UNIVERSITIES];**

**int count = 0;**

**int choice;**

**do {**

**printf("\nUniversity Management System\n");**

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

**printf("2. Update University\n");**

**printf("3. Retrieve Universities\n");**

**printf("4. Delete University\n");**

**printf("5. Sort Universities (Merge 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. Store Universities to File\n");**

**printf("10. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**university\_create(universities, &count);**

**break;**

**case 2:**

**university\_update(universities, count);**

**break;**

**case 3:**

**university\_retrieve(universities, count);**

**break;**

**case 4:**

**university\_delete(universities, &count);**

**break;**

**case 5:**

**university\_compare\_mergesort(universities, count);**

**break;**

**case 6:**

**university\_compare\_quicksort(universities, count);**

**break;**

**case 7: {**

**int id;**

**printf("Enter university ID to search: ");**

**scanf("%d", &id);**

**university\_compare\_linearsearch(universities, count, id);**

**break;**

**}**

**case 8: {**

**int id;**

**printf("Enter university ID to search: ");**

**scanf("%d", &id);**

**university\_compare\_binarysearch(universities, count, id);**

**break;**

**}**

**case 9:**

**university\_storing(universities, count);**

**break;**

**case 10:**

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

**break;**

**default:**

**printf("Invalid choice! Please try again.\n");**

**}**

**} while (choice != 10);**

**return 0;**

**}**

**// Function implementations**

**void university\_create(University universities[], int \*count) {**

**if (\*count >= MAX\_UNIVERSITIES) {**

**printf("Cannot add more universities.\n");**

**return;**

**}**

**University u;**

**printf("Enter ID: ");**

**scanf("%d", &u.id);**

**printf("Enter University Code: ");**

**scanf("%s", u.univ\_code);**

**printf("Enter University Name: ");**

**scanf(" %[^\n]", u.univ\_name); // To read string with spaces**

**printf("Enter University Address: ");**

**scanf(" %[^\n]", u.univ\_address);**

**printf("Enter University Email: ");**

**scanf("%s", u.univ\_email);**

**printf("Enter University Website: ");**

**scanf("%s", u.univ\_website);**

**universities[\*count] = u;**

**(\*count)++;**

**printf ("University added successfully!\n");**

**}**

**void university\_update(University universities[], int count) {**

**int id, found = 0;**

**printf("Enter ID of the university to update: ");**

**scanf("%d", &id);**

**for (int i = 0; i < count; i++) {**

**if (universities[i].id == id) {**

**found = 1;**

**printf("Updating details for ID %d:\n", id);**

**printf("Enter new University Code: ");**

**scanf("%s", universities[i].univ\_code);**

**printf("Enter new University Name: ");**

**scanf(" %[^\n]", universities[i].univ\_name);**

**printf("Enter new University Address: ");**

**scanf(" %[^\n]", universities[i].univ\_address);**

**printf("Enter new University Email: ");**

**scanf("%s", universities[i].univ\_email);**

**printf("Enter new University Website: ");**

**scanf("%s", universities[i].univ\_website);**

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

**break;**

**}**

**}**

**if (!found) {**

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

**}**

**}**

**void university\_retrieve(University universities[], int count) {**

**printf("List of Universities:\n");**

**for (int i = 0; i < count; i++) {**

**printf("ID: %d, Code: %s, Name: %s, Address: %s, Email: %s, Website: %s\n",**

**universities[i].id, universities[i].univ\_code, universities[i].univ\_name,**

**universities[i].univ\_address, universities[i].univ\_email, universities[i].univ\_website);**

**}**

**}**

**void university\_delete(University universities[], int \*count) {**

**int id, found = 0;**

**printf("Enter ID of the university to delete: ");**

**scanf("%d", &id);**

**for (int i = 0; i < \*count; i++) {**

**if (universities[i].id == id) {**

**found = 1;**

**for (int j = i; j < \*count - 1; j++) {**

**universities[j] = universities[j + 1];**

**}**

**(\*count)--;**

**printf("University with ID %d deleted successfully!\n", id);**

**break;**

**}**

**}**

**if (!found) {**

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

**}**

**}**

**void university\_mergesort(University universities[], int left, int right) {**

**if (left < right) {**

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

**university\_mergesort(universities, left, mid);**

**university\_mergesort(universities, mid + 1, right);**

**merge(universities, left, mid, right);**

**}**

**}**

**void merge(University universities[], int left, int mid, int right) {**

**int i, j, k;**

**int n1 = mid - left + 1;**

**int n2 = right - mid;**

**University \*L = (University \*)malloc(n1 \* sizeof(University));**

**University \*R = (University \*)malloc(n2 \* sizeof(University));**

**for (i = 0; i < n1; i++)**

**L[i] = universities[left + i];**

**for (j = 0; j < n2; j++)**

**R[j] = universities[mid + 1 + j];**

**i = 0;**

**j = 0;**

**k = left;**

**while (i < n1 && j < n2) {**

**if (L[i].id <= R[j].id) {**

**universities[k] = L[i];**

**i++;**

**} else {**

**universities[k] = R[j];**

**j++;**

**}**

**k++;**

**}**

**while (i < n1) {**

**universities[k] = L[i];**

**i++;**

**k++;**

**}**

**while (j < n2) {**

**universities[k] = R[j];**

**j++;**

**k++;**

**}**

**free(L);**

**free(R);**

**}**

**void university\_quicksort(University universities[], int low, int high) {**

**if (low < high) {**

**int pi = partition(universities, low, high);**

**university\_quicksort(universities, low, pi - 1);**

**university\_quicksort(universities, pi + 1, high);**

**}**

**}**

**int partition(University universities[], int low, int high) {**

**int pivot = universities[high].id;**

**int i = (low - 1);**

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

**if (universities [j].id < pivot) {**

**i++;**

**University temp = universities[i];**

**universities[i] = universities[j];**

**universities[j] = temp;**

**}**

**}**

**University temp = universities[i + 1];**

**universities[i + 1] = universities[high];**

**universities[high] = temp;**

**return (i + 1);**

**}**

**int university\_linearsearch(University universities[], int count, int id) {**

**for (int i = 0; i < count; i++) {**

**if (universities[i].id == id) {**

**return i; // Return the index if found**

**}**

**}**

**return -1; // Return -1 if not found**

**}**

**int university\_binarysearch(University universities[], int left, int right, int id) {**

**while (left <= right) {**

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

**if (universities[mid].id == id) {**

**return mid; // Return the index if found**

**}**

**if (universities[mid].id < id) {**

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

**} else {**

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

**}**

**}**

**return -1; // Return -1 if not found**

**}**

**void university\_storing(University universities[], int count) {**

**writeToFile(universities, count);**

**}**

**void writeToFile(University universities[], int count) {**

**FILE \*file = fopen("university\_setting.txt", "w");**

**if (file == NULL) {**

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

**return;**

**}**

**for (int i = 0; i < count; i++) {**

**fprintf(file, "ID: %d, Code: %s, Name: %s, Address: %s, Email: %s, Website: %s\n",**

**universities[i].id, universities[i].univ\_code, universities[i].univ\_name,**

**universities[i].univ\_address, universities[i].univ\_email, universities[i].univ\_website);**

**}**

**fclose(file);**

**printf("Data stored successfully in university\_setting.txt\n");**

**}**

**void university\_compare\_linearsearch(University universities[], int count, int id) {**

**clock\_t start = clock();**

**int index = university\_linearsearch(universities, count, id);**

**clock\_t end = clock();**

**if (index != -1) {**

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

**} else {**

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

**}**

**university\_complexity\_linearsearch(count);**

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

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

**}**

**void university\_compare\_binarysearch(University universities[], int count, int id) {**

**university\_quicksort(universities, 0, count - 1); // Ensure the array is sorted**

**clock\_t start = clock();**

**int index = university\_binarysearch(universities, 0, count - 1, id);**

**clock\_t end = clock();**

**if (index != -1) {**

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

**} else {**

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

**}**

**university\_complexity\_binarysearch(count);**

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

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

**}**

**void university\_compare\_quicksort(University universities[], int count) {**

**clock\_t start = clock();**

**university\_quicksort(universities, 0, count - 1);**

**clock\_t end = clock();**

**printf("Universities sorted using Quick Sort.\n");**

**university\_complexity\_quicksort(count);**

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

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

**}**

**void university\_compare\_mergesort(University universities[], int count) {**

**clock\_t start = clock();**

**university\_mergesort(universities, 0, count - 1);**

**clock\_t end = clock();**

**printf("Universities sorted using Merge Sort.\n");**

**university\_complexity\_mergesort(count);**

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

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

**}**

**void university\_complexity\_linearsearch(int n) {**

**printf("Time Complexity of Linear Search: O(n)\n");**

**}**

**void university\_complexity\_binarysearch(int n) {**

**printf("Time Complexity of Binary Search: O(log n )\n");**

**}**

**void university\_complexity\_mergesort(int n) {**

**printf("Time Complexity of Merge Sort: O(n log n)\n");**

**}**

**void university\_complexity\_quicksort(int n) {**

**printf("Time Complexity of Quick Sort: O(n log n) on average, O(n^2) in the worst case\n");**

**}**

**Comparison of Sorting Algorithms**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**#include <string.h>**

**#define SIZE 10000**

**struct University {**

**int id;**

**int univ\_code;**

**char univ\_name[100];**

**char univ\_address[200];**

**char univ\_email[100];**

**char univ\_website[100];**

**};**

**// Function prototypes ```c**

**void university\_algo\_quicksort(struct University universities[], int low, int high);**

**void university\_algo\_mergesort(struct University universities[], int left, int right);**

**void generate\_random\_universities(struct University universities[], int n);**

**void copy\_array(struct University source[], struct University destination[], int n);**

**void print\_results(int quicksort\_time, int mergesort\_time, int quicksort\_space, int mergesort\_space);**

**int main() {**

**struct University original\_array[SIZE];**

**struct University quicksort\_array[SIZE];**

**struct University mergesort\_array[SIZE];**

**// Step 1: Generate random universities**

**generate\_random\_universities(original\_array, SIZE);**

**// Step 2: Copy original array to both sorting arrays**

**copy\_array(original\_array, quicksort\_array, SIZE);**

**copy\_array(original\_array, mergesort\_array, SIZE);**

**// Step 3: Measure time for Quick Sort**

**clock\_t start = clock();**

**university\_algo\_quicksort(quicksort\_array, 0, SIZE - 1);**

**clock\_t end = clock();**

**int quicksort\_time = (int)((end - start) \* 1000 / CLOCKS\_PER\_SEC);**

**// Step 4: Measure time for Merge Sort**

**start = clock();**

**university\_algo\_mergesort(mergesort\_array, 0, SIZE - 1);**

**end = clock();**

**int mergesort\_time = (int)((end - start) \* 1000 / CLOCKS\_PER\_SEC);**

**// Step 5: Output the results**

**print\_results(quicksort\_time, mergesort\_time, sizeof(quicksort\_array), sizeof(mergesort\_array));**

**return 0;**

**}**

**void generate\_random\_universities(struct University universities[], int n) {**

**for (int i = 0; i < n; i++) {**

**universities[i].id = rand() % 10000; // Random ID**

**universities[i].univ\_code = rand() % 1000; // Random university code**

**snprintf(universities[i].univ\_name, sizeof(universities[i].univ\_name), "University %d", i);**

**snprintf(universities[i].univ\_address, sizeof(universities[i].univ\_address), "Address %d", i);**

**snprintf(universities[i].univ\_email, sizeof(universities[i].univ\_email), "email%d@example.com", i);**

**snprintf(universities[i].univ\_website, sizeof(universities[i].univ\_website), "www.university%d.com", i);**

**}**

**}**

**void copy\_array(struct University source[], struct University destination[], int n) {**

**memcpy(destination, source, n \* sizeof(struct University));**

**}**

**void print\_results(int quicksort\_time, int mergesort\_time, int quicksort\_space, int mergesort\_space) {**

**printf("Quick Sort Time: %d milliseconds\n", quicksort\_time);**

**printf("Merge Sort Time: %d milliseconds\n", mergesort\_time);**

**printf("Quick Sort Space Complexity: %d bytes\n", quicksort\_space);**

**printf("Merge Sort Space Complexity: %d bytes\n", mergesort\_space);**

**}**

**Comparison of Searching Algorithms**

**FUNCTION compare\_search\_algorithms()**

**// Step 1: Generate a random array of universities**

**INTEGER n = 10000 // Size of the array**

**DECLARE ARRAY universities[n]**

**generate\_random\_universities(universities, n)**

**// Step 2: Sort the array for binary search**

**sort\_universities\_by\_id(universities, n) // Use any sorting algorithm (e.g., Quick Sort)**

**// Step 3: Choose a target ID to search for**

**INTEGER target\_id = universities[RANDOM\_INDEX(n)].id // Randomly select an ID from the array**

**// Step 4: Measure time for Linear Search**

**START\_TIMER()**

**INTEGER linear\_search\_result = university\_algo\_linearsearch(universities, n, target\_id)**

**END\_TIMER()**

**INTEGER linear\_search\_time = GET\_ELAPSED\_TIME()**

**// Step 5: Measure time for Binary Search**

**START\_TIMER()**

**INTEGER binary\_search\_result = university\_algo\_binarysearch(universities, 0, n - 1, target\_id)**

**END\_TIMER()**

**INTEGER binary\_search\_time = GET\_ELAPSED\_TIME()**

**// Step 6: Output the results**

**PRINT "Linear Search Time: " + linear\_search\_time + " milliseconds"**

**PRINT "Binary Search Time: " + binary\_search\_time + " milliseconds"**

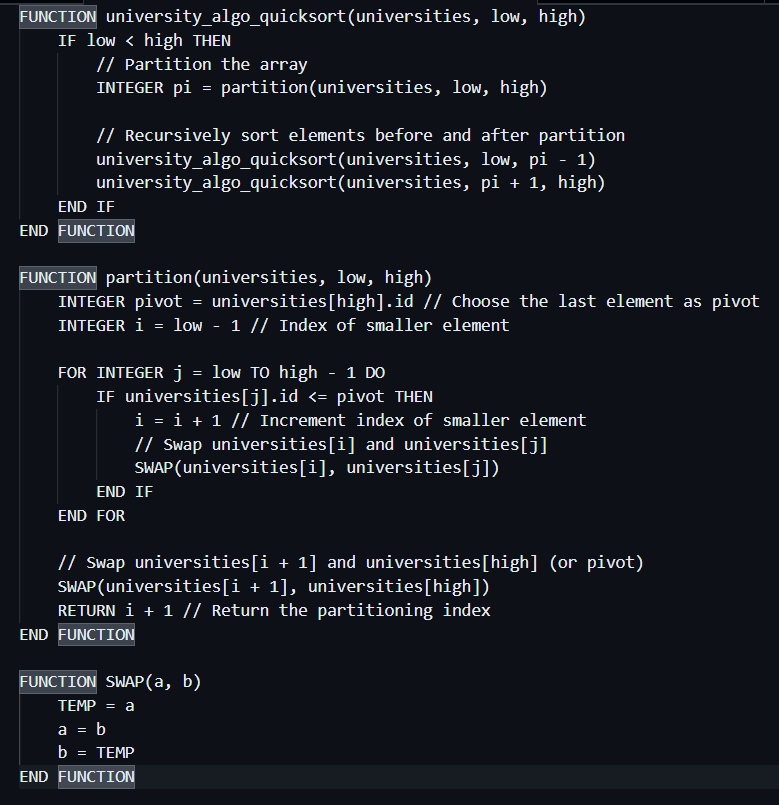
**PRINT "Linear Search Result Index: " + linear\_search\_result**

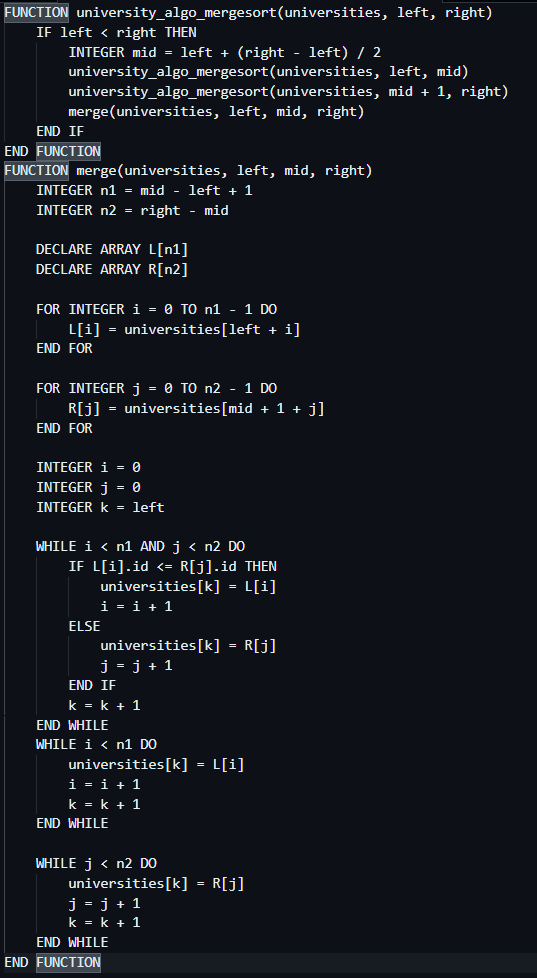
**PRINT "Binary Search Result Index: " + binary\_search\_result**

**END FUNCTION**

**Screen Shots**

**QUICK SORT (university\_algo\_quicksort).**

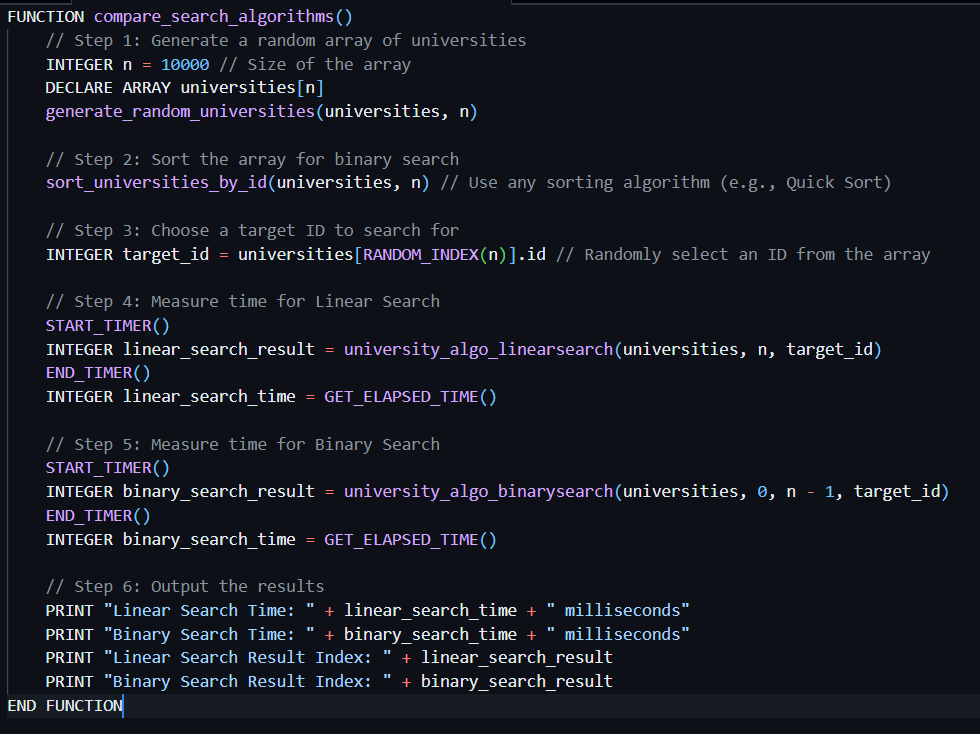
****

**  
MERGE SORT (university\_algo\_mergesort).**

LINEAR SEARCH & BINARY SEARCH

(university\_algo\_linearsearch

university\_algo\_binarysearch)



**CONCLUSION**

The University Management System project represents a significant step forward in the digital transformation of university operations. By leveraging technology to enhance administrative efficiency and improve the overall educational experience, the UMS positions the university to better serve its students and faculty in an increasingly competitive academic environment. The successful implementation of this system not only meets current needs but also lays the groundwork for future innovations in university management.