**Searching and Sorting**

**assignment1.cpp :-**

/\*

Assignment 1

Title :- Searching and Sorting

Aim :- To create a student database and perform searching(linear and binary search) and sorting(quick, insertion and bubble sort).

\*/

#include <iostream> // header file

#include <cstring> // to perform string opretions

#include <chrono> // to measure time required

using namespace std;

// struct student :- to store information of a student in structured manner

struct Student

{ // structure of a student

string name; // student information

int rollNo;

float SGPA;

};

// class student database :- to maintain database of students

class StudentDB

{

private:

int swaps, iterations, comparisons; // initialization of variables

int NoOfStudents = 0;

struct Student S[15]; // array of struct for storing database of students

public:

void addStudent(); // adds a student into the database

void displayStudents(); // display all students

void record(); // search and displays a student

// UTILITY FUNCTIONS

void variable\_init(); // initialize all variables to 0

void variable\_display(); // display variables

int menu(); // menu for user

void swap(Student \*a, Student \*b); // swap 2 students

int getNo(); // returns no of students in database

void linearSearch(); // linear search - SGPA

void binarySearch(); // binary search - Name

void insertionSort(); // insertion sort - Name

void bubbleSort(); // bubble sort - Roll no

int partition(int lb, int ub); // helper function for quick sort

void quickSort(int lb, int ub); // quick sort - SGPA

};

/\*------------------ADD STUDENT-------------------\*/

void StudentDB ::addStudent()

{

cout << "Enter your first name : " << endl;

cin >> S[NoOfStudents].name;

cout << "Enter Roll No : " << endl;

cin >> S[NoOfStudents].rollNo;

cout << "Enter SGPA (Only between 0 to 10) : " << endl;

cin >> S[NoOfStudents].SGPA;

NoOfStudents++;

}

/\*------------------SWAP STUDENTS-------------------\*/

void StudentDB ::swap(Student \*a, Student \*b)

{

Student temp;

temp = \*a;

\*a = \*b;

\*b = temp;

swaps++;

}

/\*------------------MENU-------------------\*/

int StudentDB ::menu()

{

cout << "Enter your choice\n1 : Add student record.\n2 : Display the record of student.\n";

cout << "3 : Display student database.\n4 : Search the student by name.\n5 : Search the student by SGPA.\n";

cout << "6 : Sort students by name and view list.\n7 : Sort students by roll no and view list.\n";

cout << "8 : Sort students by CGPA and view list.\n9 : exit.\n";

int choice;

cin >> choice;

return choice;

}

/\*------------------DISPLAY VARIABLES-------------------\*/

void StudentDB ::variable\_display()

{

cout << "No of swaps : " << swaps << endl;

cout << "No of comparisons : " << comparisons << endl;

cout << "No of iterations : " << iterations << endl;

}

/\*------------------INITIALIZE VARIABLES-------------------\*/

void StudentDB ::variable\_init()

{

swaps = 0;

comparisons = 0;

iterations = 0;

}

/\*------------------RETURN NO OF STUDENTS-------------------\*/

int StudentDB ::getNo()

{

return this->NoOfStudents;

}

/\*------------------DISPLAY STUDENTS-------------------\*/

void StudentDB ::displayStudents()

{

cout << "--------------------RECORD OF STUDENTS----------------------\n\n";

cout << "\t\tName\t\tRoll No\t\tSGPA\n";

for (int i = 0; i < NoOfStudents; i++)

{

cout << "------------------------------------------------------------\n";

cout << "Sr no. " << i + 1 << "\t";

cout << S[i].name << "\t\t";

cout << S[i].rollNo << "\t\t";

cout << S[i].SGPA << endl;

cout << "------------------------------------------------------------\n";

}

}

/\*------------------SEARCH AND DISPLAY STUDENT-------------------\*/

void StudentDB ::record()

{

int indexStu;

cout << "Enter the sr no of student to display record : ";

cin >> indexStu;

cout << "Name : ";

cout << S[indexStu - 1].name << endl;

cout << "Roll no : ";

cout << S[indexStu - 1].rollNo << endl;

cout << "SGPA : ";

cout << S[indexStu - 1].SGPA << endl;

}

/\*------------------LINEAR SEARCH(SGPA)-------------------\*/

void StudentDB ::linearSearch()

{

float key;

cout << "Enter SGPA of student : ";

cin >> key;

int flag = 0;

for (int i = 0; i < NoOfStudents; i++)

{

comparisons++;

iterations++;

if (S[i].SGPA == key)

{

comparisons++;

cout << "Name : " << S[i].name << "\n";

cout << "Roll No : " << S[i].rollNo << "\n";

cout << "SGPA : " << S[i].SGPA << "\n";

flag = 1;

}

}

if (flag == 0)

{

cout << "No valid entry found\nPlease enter valid information";

}

comparisons++;

}

/\*------------------BINARY SEARCH(NAME)-------------------\*/

void StudentDB ::binarySearch()

{

string key;

cout << "enter name of student : ";

cin >> key;

int start = 0;

int end = NoOfStudents;

int mid, check, result = 0;

while (start <= end)

{

comparisons++;

iterations++;

mid = (start + end) / 2;

// int result = strcmp(arr[mid].name, key.name);

if (S[mid].name == key)

{

cout << "key match\n";

result = mid;

check = 1;

break;

}

else if (S[mid].name < key)

{

start = mid + 1;

}

else

{

end = mid - 1;

}

comparisons += 2;

}

if (check == 0)

{

cout << "Key not found\n";

}

else

{

cout << "Name : " << S[result].name << "\n";

cout << "Roll no : " << S[result].rollNo << "\n";

cout << "SGPA : " << S[result].SGPA << "\n";

}

comparisons++;

}

/\*------------------INSERTION SORT(NAME)-------------------\*/

void StudentDB ::insertionSort()

{

Student temp;

for (int i = 1; i < NoOfStudents; i++)

{

iterations++;

comparisons++;

temp = S[i];

int j;

j = i - 1;

while (j >= 0 && temp.name < S[j].name)

{

comparisons++;

swaps++;

S[j + 1] = S[j];

j--;

}

S[j + 1] = temp;

}

}

/\*------------------BUBBLE SORT(ROLL NO)-------------------\*/

void StudentDB ::bubbleSort()

{

for (int i = 0; i < NoOfStudents - 1; i++)

{

iterations++;

comparisons++;

for (int j = 0; j < NoOfStudents - i - 1; j++)

{

comparisons++;

if (S[j].rollNo > S[j + 1].rollNo)

{

Student temp;

temp = S[j];

S[j] = S[j + 1];

S[j + 1] = temp;

swaps++;

}

comparisons++;

}

}

}

/\*------------------PARTITIONS THE ARRAY - HEPER FUNCTION OF QUICK SORT-------------------\*/

int StudentDB ::partition(int lb, int ub)

{

iterations++;

int i = lb + 1;

int j = ub;

Student temp;

int pivot = S[lb].SGPA;

do

{

while (S[i].SGPA >= pivot)

{

i++;

comparisons++;

}

while (S[j].SGPA < pivot)

{

j--;

comparisons++;

}

if (i < j)

{

comparisons++;

swap(&S[i], &S[j]);

}

} while (i < j);

swap(&S[lb], &S[j]);

return j;

}

/\*------------------QUICK SORT(SGPA)-------------------\*/

void StudentDB ::quickSort(int lb, int ub)

{

// base case

if (ub > lb)

{

comparisons++;

int pi = partition(lb, ub);

// left part

quickSort(lb, pi - 1);

// right part

quickSort(pi + 1, ub);

}

}

int main()

{

StudentDB S1; // creating object of a student

while (1)

{

int choice = S1.menu(); // menu for user

if (choice == 1) // add student

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.addStudent();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 2) // search and display student

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.record();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 3) // displays al students

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.displayStudents();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 4) // binary search - name

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.variable\_init();

S1.binarySearch();

S1.variable\_display();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 5) // linear search - sgpa

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.variable\_init();

S1.linearSearch();

S1.variable\_display();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 6) // insertion sort - name

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.variable\_init();

S1.insertionSort();

S1.displayStudents();

S1.variable\_display();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 7) // bubble sort - roll no

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.variable\_init();

S1.bubbleSort();

S1.displayStudents();

S1.variable\_display();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 8) // quick sort - sgpa

{

auto start = std::chrono::high\_resolution\_clock::now();

S1.variable\_init();

int NoOfStudents;

NoOfStudents = S1.getNo();

S1.quickSort(0, NoOfStudents - 1);

S1.displayStudents();

S1.variable\_display();

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Time taken by the program: " << duration.count() << " microseconds" << std::endl;

}

else if (choice == 9) // exit from program

{

exit(0);

}

else // invalid case

{

cout << "Please enter valid information";

break;

}

}

return 0;

}