University of Engineering and Technology Lahore (New Campus)

Department of Electrical Engineering and Technology

Data Structure and Algorithms LAB

Submitted to:

Dr. Haris Anwar

Submitted By:

Ahmed Shafique 2021-EE-313

Haiqa Qaiser 2022-EE-255

Table of Contents:

Lab 1 & 2: Fundamentals

Q1): Write separate function of each of the following:

Functions:

    a. Sum of Arithmetic Series:

```
double arithmeticSeriesSum(double a, double d, int n)
{
    double sum = (n/2) * (2 * a + (n - 1) * d);
    return sum;
}
```

    b. Sum of Geometric Series:

```
double geometric_series(double a, double r, int n) {
    if (r > 1) {
        return a * (pow(r, n) - 1) / (r - 1);
    } else if (r < 1) {
        return a * (1 - pow(r, n)) / (1 - r);
    } else {
        cout << "The common ratio cannot be 1." << endl;
        return -1;
    }
}
```

    c. Sum of infinite geometric Series:

```
double geometric_infinite_series(double a, double r) {
    if (r < 1) {
        return a / (1 - r);
    } else if (r > 1) {
        return a / (r - 1);
    } else {
        cout << "The common ratio cannot be 1." << endl;
        return -1;
    }
}
```

    d. A function that return a structure Z:

```
void z()

{

    geometric_infinite_series(double a, double r);

    geometric_series(double a, double r, int n);

}
```

Q#02: Write a program that uses the functions to develop in Q1 in order to:

1. Asks the user which function he/she wants to calculate? (3)

2. Accepts from the user appropriate inputs, note that each function mentioned above may require a different set of inputs.

3. Calls the requisite function to calculate the formula, prints the results and exits.

```
int main()

{

    int choice;

    cout<<"Hello! What you want to calculate from the following?
type(1 or 2 or 3)";cin>>choice;
```

```cpp
    double a, d, r;
    int n;
    if (choice==1)
    {
        cout<<"Enter the value of a : ";cin>>a;
        cout<<"Enter the value of d : ";cin>>d;
        cout<<"Enter the value of n : ";cin>>n;
        double sum1 = arithmeticSeriesSum(a,d,n);
        cout<<"Sum of Arithmatic Series: "<<sum1<<endl;
    }
    else if (choice ==2)
    {
        cout<<"Enter the value of a : ";cin>>a;
        cout<<"Enter the value of r : ";cin>>r;
        cout<<"Enter the value of n : ";cin>>n;
        double sum2 = geometric_series(a, r, n);
        cout<<"Sum of Arithmatic Series: "<<sum2<<endl;
    }
    else
    {
        cout<<"Enter the value of a : ";cin>>a;
        cout<<"Enter the value of r : ";cin>>r;
        double sum3 = geometric_infinite_series(a,r);
        cout<<"Sum of Arithmatic Series: "<<sum3<<endl;
    }
    return 0;
}
```

Q#03: Write a program that:

1. Declare a structure named 'student.' Each student should contain a name and marks of three courses, namely, {Math, Chem, Physics}.

2. Make an array of structures, where the size of the array is 10.

3. Initialize the names and marks of these 10 students via the user.

4. Call out a function named 'calculate_average' which accepts this array of structures and evaluates and prints the respective averages of Math, Chemistry and Physics.

5. Call out a function names 'bubble_sort' which accepts this array of structures and sorts and prints the student list (i.e., their names and marks) as per their names.

Code:

```cpp
struct student {
    string name;
    double chem;
    double phy;
    double math;
};
int length = 0;
void initial(int n, struct student **data_arr) {
    cout << "---------- Loading ----------" << endl;
    for (int i = 0; i < n; i++) {
        cout << "[" << i + 1 << "]: " << "Enter the name of Student: ";
        cin >> data_arr[i]->name;
        cout << endl;
        cout << "Enter the chemistry marks: ";
        cin >> data_arr[i]->chem;
        cout << "Enter the physics marks: ";
        cin >> data_arr[i]->phy;
        cout << "Enter the math marks: ";
        cin >> data_arr[i]->math;
        cout << endl;
    }
    cout << "---------- Please Wait ----------" << endl;
    length = length + 1;
    cout << endl;
    return;
}


void calculate_average() {
    // Your implementation for calculating average goes here
    return;
}


bool compareByName(const student* a, const student* b) {
    return a->name < b->name;
}


void sort(int n, struct student **data_arr) {
```

```cpp
        cout << "---------- Alphabetized order ----------" << endl;

        sort(data_arr, data_arr + n, compareByName);

        return;

}


void printing(int n, struct student **data_arr) {

        cout << "---------- Loading ----------" << endl;

        cout << endl << "---------- Recorded Data ----------" << endl;

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

                cout << "[" << i + 1 << "]: " << "Name: " << data_arr[i]->name

                        << " Chemistry: " << data_arr[i]->chem

                        << " Physics: " << data_arr[i]->phy

                        << " Math: " << data_arr[i]->math << endl;

                double avg = (data_arr[i]->chem + data_arr[i]->phy + data_arr[i]->math) / 3.0;

                cout << " Average: " << avg << endl;

        }

        cout << endl;

        length = length + 1;

}


int main() {

        int n;

        cout << "Enter the number of student's data that you want to store in structure: ";

        cin >> n;

        cout << endl;

        student* arr[n]; // array of structure

        student* ptr_arr[n]; // pointer array points to structure.

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

                ptr_arr[i] = new student; // new pointer structure & new multiples dynamic memory allocations.

        }

        cout << endl;

        initial(n, ptr_arr); // calling with original ptr_arr[] instead of data_arr[].

        sort(n, ptr_arr);

        printing(n, ptr_arr);

        cout << "Length of Function: " << length << endl;
```

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

            delete ptr_arr[i];

        }

        return 0;

    }
```

Output:

Enter the number of student's data that you want to store in structure: 2

---------- Loading ----------
[1]: Enter the name of Student: Ahmed

Enter the chemistry marks: 99
Enter the physics marks: 98
Enter the math marks: 99

Fig: 01 Q3

[2]: Enter the name of Student: Haiqa

Enter the chemistry marks: 98
Enter the physics marks: 99
Enter the math marks: 98

Fig: 02 Q3

[1]: Name: Ahmed Chemistry: 99 Physics: 98 Math: 99
 Average: 98.6667
[2]: Name: Haiqa Chemistry: 98 Physics: 99 Math: 98
 Average: 98.3333

Fig: 03 Q3

Hello! What you want to calculate from the following? type(1 or 2 or 3)2
Enter the value of a : 25
Enter the value of r : 4
Enter the value of n : 8
Sum of Arithmatic Series: 546125

Fig: 04 Q1 & Q2

Hello! What you want to calculate from the following? type(1 or 2 or 3)2
Enter the value of a : 4
Enter the value of r : 6
Enter the value of n : 8
Sum of Geometric Series: 1.34369e+006

Fig: 05 Q1 & Q2

Hello! What you want to calculate from the following? type(1 or 2 or 3)3
Enter the value of a : 5
Enter the value of r : 9
Sum of Infinite Geometric Series: 0.625

Fig: 06 Q1 & Q2

Methodology:

This C++ program calculates the sum of either an arithmetic series, geometric series, or infinite geometric series based on user input. It defines functions for each calculation: `arithmeticSeriesSum` for arithmetic series, `geometric_series` for finite geometric series, and `geometric_infinite_series` for infinite geometric series. The `main` function prompts the user to choose the type of series and input necessary values, then computes and displays the corresponding sum.

This C++ program manages student data by defining a `student` struct with attributes for name, chemistry marks, physics marks, and math marks. It implements functions to initialize data, calculate averages, sort data alphabetically by name, and print recorded data with their averages. The `main` function prompts the user to input the number of students, then records and displays their data. Finally, it deallocates memory used by dynamically allocated structures.

Lab 3: Fundamentals

Q1): Write a program that:

Declare a structure named 'student.' Each student should contain a name and marks of three courses, namely, {Math, Chem, Physics}.

Make an array of pointers, where the size of the array is 10. Each pointer should be able to point to a 'student.'

Calls out a function, that initializes the names and marks of these 10 students via the user. The function should have, as an argument, 'n' – the size of the array and 'pointer_array' – a pointer to the 1st location of the array of pointers.

Write a function named 'sort' which accepts (n, pointer_array) and sorts the list as per their names.

Call out a function named 'print_list' which accepts (n, pointer_array) and prints the sorted list as per their names.

Code:

```cpp
#include <iostream>
#include <string>
#include <algorithm> // Added for sorting
#include <cmath>

using namespace std;

struct student {
    string name;
    double chem;
    double phy;
    double math;
};

int length = 0;

void initial(int n, struct student **data_arr) {
    cout << "---------- Loading ----------" << endl;
    for (int i = 0; i < n; i++) {
        cout << "[" << i + 1 << "]: " << "Enter the name of Student: ";
        cin >> data_arr[i]->name;
        cout << endl;
        cout << "Enter the chemistry marks: ";
```

```cpp
        cin >> data_arr[i]->chem;

        cout << "Enter the physics marks: ";

        cin >> data_arr[i]->phy;

        cout << "Enter the math marks: ";

        cin >> data_arr[i]->math;

        cout << endl;

    }
    cout << "---------- Please Wait ----------" << endl;

    length = length + 1;

    cout << endl;

    return;

}


void calculate_average() {

    // Your implementation for calculating average goes here

    return;

}


bool compareByName(const student* a, const student* b) {

    return a->name < b->name;

}


void sort(int n, struct student **data_arr) {

    cout << "---------- Alphabetized order ----------" << endl;

    sort(data_arr, data_arr + n, compareByName);

    return;

}


void printing(int n, struct student **data_arr) {

    cout << "---------- Loading -----------" << endl;

    cout << endl << "---------- Recorded Data -----------" << endl;

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

        cout << "[" << i + 1 << "]: " << "Name: " << data_arr[i]-
>name

            << " Chemistry: " << data_arr[i]->chem

            << " Physics: " << data_arr[i]->phy

            << " Math: " << data_arr[i]->math << endl;
```

```cpp
        double avg = (data_arr[i]->chem + data_arr[i]->phy +
data_arr[i]->math) / 3.0;

        cout << " Average: " << avg << endl;

    }

    cout << endl;

    length = length + 1;

}


int main() {

    int n;

    cout << "Enter the number of student's data that you want to
store in structure: ";

    cin >> n;

    cout << endl;

    student* arr[n]; // array of structure

    student* ptr_arr[n]; // pointer array points to structure.

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

        ptr_arr[i] = new student; // new pointer structure & new
multiples dynamic memory allocations.

    }

    cout << endl;

    initial(n, ptr_arr); // calling with original ptr_arr[] instead
of data_arr[].

    sort(n, ptr_arr);

    printing(n, ptr_arr);

    cout << "Length of Function: " << length << endl;

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

        delete ptr_arr[i];

    }

    return 0;

}
```

Methodology:

This C++ program manages student data by defining a `student` struct with attributes for name, chemistry marks, physics marks, and math marks. It implements functions to initialize data, calculate. averages, sort data alphabetically by name, and print recorded data with their averages. The `main` function prompts the user to input the number of students, then records and displays their data. Finally, it deallocates memory used by dynamically allocated structures.

Outputs:

```
Enter the number of student's data that you want to store in structure: 2


---------- Loading ----------
[1]: Enter the name of Student: Ahmed

Enter the chemistry marks: 99
Enter the physics marks: 98
Enter the math marks: 99
```
Fig: 01

```
[2]: Enter the name of Student: Haiqa

Enter the chemistry marks: 98
Enter the physics marks: 99
Enter the math marks: 98
```
Fig: 02

```
[1]: Name: Ahmed Chemistry: 99 Physics: 98 Math: 99
 Average: 98.6667
[2]: Name: Haiqa Chemistry: 98 Physics: 99 Math: 98
 Average: 98.3333
```
Fig: 03

Lab 4: File handling

Q1): Evaluating execution time:

The library <Time.h> has three important elements:

1. clock_t: this is our data-type,

2. CLOCKS_PER_SEC: a reserved word which evaluatesthe number of clocks the timer makes persecond. This variable changes for different processors. Faster processors have higher clocks per second.

3. clock(): function that measures the number of clocks at a particular instance. It returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, you will need to divide by CLOCKS_PER_SEC

Code:
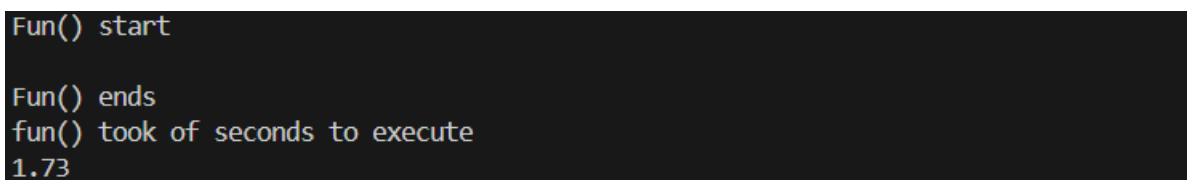
```
#include <iostream>
#include <time.h>
using namespace std;

void fun(){

    cout<<"Fun() start \n";

    while (1)
    {
        if (getchar())
        {
        }
    }

    cout<<"Fun() ends \n";
}

// main function here
int main()
{
    clock_t t1,t2;
    t1 = clock();
    fun();
    t2 = clock() - t1;
    double time_taken = ((double) t2) / CLOCKS_PER_SEC;
    cout<<"fun() took of seconds to execute \n"<<time_taken;
    return 0;
}
```

Output:

```
Fun() start

Fun() ends
fun() took of seconds to execute
1.73
```

<div align="center">Fig: 01</div>

Q#02: Feeding parameters directly from console:

There are two extremely important arguments of the main function, namely, argc and argv[]. Here 'argc' stands for argument count and 'argv' stands for argument values. These are variables passed to

the main function when it starts executing. When we run a program we can give arguments to that program like:

C:\Users\MyComputer\Downloads> Hello.exe 7

Here the argument 7 is directly provided in the console and the program will print Hello a total of seven times. Consider the code mentioned below. Notice, now we are evaluating the execution time by default. Evaluating the execution time should appear in all your Data-Structure codes. Write the code below and save it as "Hello.c"

Code:

```cpp
#include <iostream>
#include <string.h>
#include <bits/stdc++.h>
#include <sstream>
using namespace std;

int main(int argc, char *argv [])
{
    string A;
    try
    {
        if (argc != 3)
        {
            throw(argc);
        }
        else
        {
            cout<<"Lets start the program"<<endl;
            A = argv[0];
            cout<<"Write Program Name: "<<A<<endl;
            A = argv[1];
            cout<<"Name of program to be repeated: "<<A<<endl;
            A = argv[2];
            cout<<"How many times: "<<A<<" x "<<endl;

            // convertng string to number to int to repeatitions
            int i, tau;
            stringstream ss;
            ss<<argv[2];
            ss>>tau;

            // printing how many times depends on user last input
            for (int i = 0; i <= tau; i++)
            {
                cout<<"[ "<<i<<" ]"<<argv[1]<<endl;
            }
        }
    }
    catch(...)
    {
        cout<<"This program has 3 parameters: "<<endl;
        cout<<"[1] Name of program like : [a b c].exe"<<endl;
        cout<<"[2] Name of user you want to print like Ahmed: "<<endl;
        cout<<"[3] How many times you want to print the name of user: "<<endl;
        cout<<endl;
        cout<<"for e.g: Typing Ahmed"<<endl;
        cout<<"DSA Ahmed 3"<<endl;
        cout<<"Will Ahmed print 3 times?"<<endl;
    }
    return 0;
```

```
    }
```

Output:

```
D:\6th Semester\DSA\Lab1\DSA_lab>filehand2.exe Ahmed 3
Lets start the program
Write Program Name: filehand2.exe
Name of program to be repeated: Ahmed
How many times: 3 x
[ 0 ]Ahmed
[ 1 ]Ahmed
[ 2 ]Ahmed
[ 3 ]Ahmed
```

<div align="center">Fig: 02 CMD</div>

Q#03: Input console application:

DST teaches you how to handle data using programming structures. The data encompassed within these structures is huge and goes way beyond asking the user to input details using the keyboard. Hence, reading input from an external file is fundamental. This lab teaches how to read an external file while storing the contents in another file. Focus on the comments, they are mentioned specifically for your understanding. As students are very keen in simply copy-pasting code, the following presents snap-shots (images of the running code) so that you write these codes yourself.

Code:

```cpp
#include <iostream>
#include <fstream>
#include <time.h>
#include <string.h>
#include <ctime>

#pragma warning(disable : 4996) // processor keyword
char *input_file_name, *output_file_name;
using namespace std;

int main(int argc, char *argv[])
{
    clock_t start, middle, finish;
    start = clock();
    double execution_time;

    if (argc != 4)
    {
        cout<<"Author: [Your name here] \n"<<endl;
        cout<<"Dear User! tell me about you how to use program.
"<<endl;
        cout<<"The program has 4 keywords: \n";
        cout<<"     [0] Code.exe "<<endl;
        cout<<"     [1] Name of file (input.text) "<<endl;
        cout<<"     [2] tau "<<endl;
        cout<<"     [3] Name of file (output.txt) "<<endl;
        cout<<"     [*] Example: code.exe input.txt 20 output.txt
"<<endl;
        cout<<"     [*] The code will print the contents of input
20x in output.txt  "<<endl;
    }
    else
    {
        cout<<"Author: [Your name here] \n"<<endl;
```

```cpp
        char date_time[50];
        time_t t = time(NULL);
        //ctime_S(date_time, 50, &t);
        strftime(date_time, sizeof(date_time), "%c", localtime(&t));
        cout<<" [*] Today: "<<date_time<<endl;

        int len = strlen(argv[2]);
        int tau = 0;
        int p, o = 1;
        for ( p = len - 1 ; p > -1; p= p-1)
        {
            tau = tau * ((int)argv[2][p] - 48) * o;
            o = o * 10;
        }

        input_file_name = argv[1];
        output_file_name = argv[3];

        ifstream myfile(input_file_name);
        ofstream output(output_file_name);

        if (myfile.is_open())
        {
            string line;
            while (getline(myfile, line))
            {
                for ( o = 1; o <= tau; o++)
                {
                    output<<"["<<o<<"]: "<<line<<endl;
                    output<<endl;
                }
            }
            myfile.close();
        }
        else
        {
            cout<<"[*] Failed to open file: "<<input_file_name<<endl;
            cout<<"[*] Existing - check your files and folders and
try again"<<endl;
            return EXIT_FAILURE;
        }
        finish = clock();
        execution_time = (double (finish - start)) /CLOCKS_PER_SEC;
        cout<<"[*] time taken: ["<<execution_time<<"] seconds to
complete task";
        }
    cin.get();
    return 0;
    }
```

Output:

```
D:\6th Semester\DSA\Lab1\DSA_lab>filehand3.exe input.txt 20 output.txt
Author: [Your name here]

 [*] Today: 05/07/24 08:58:45
[*] time taken: [0.015] seconds to complete task
```

Fig: 03 CMD

```
Hello
Pakistan
Aqib
Javaid
University
Closed
Enjoy
```

Fig: 04 input.txt file

```
Hello
Pakistan
Aqib
Javaid
University
Closed
Enjoy
Hello
Pakistan
Aqib
Javaid
University
Closed
Enjoy
Hello
Pakistan
Aqib
Javaid
University
Closed
Enjoy
Hello
Pakistan
Aqib
Javaid
University
```

Fig: 05 output.txt file

Methodology:

Code 1: Defines a function fun() which waits for user input. It calculates the time taken for the function to execute and prints the result.

Code 2: Takes input parameters, including a program name, a username, and a repetition count. It then prints the user's name as many times as specified.

Code 3: Reads a file, duplicates its contents a certain number of times, and writes the output to another file. It calculates execution time and handles input errors.

Lab 5: Matrices-I

Q1): Write a program that:

1 A function that initializes $Xm{\times}n$ and $Y{\times}k$, where {m, n, l, k} are user defined and also its 10 contents.

2 A function that evaluates $Z = X + Y$. 5

3 A function that evaluates $Z = X - Y$. 5

4 A function called 'print' that simply prints the answer.

Code:

```
struct element
{
    int r,c, *p;
};

void initialize_matrix(int row, int col, int *ptr)
{
    int k;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cin>>k;
            *(ptr + i*col + j) = k;
        }
    }
    return;
}


element addition_matrces(int r1, int c1, int r2, int c2, int *p1, int *p2)
{
    element matrix;
    try
    {
        if ((r1==r2) && (c1==c2))
        {
            cout<<"Processing addition of matrices"<<endl;
            matrix.c = c1;
            matrix.r = r1;
            matrix.p = new int [r1*c1];
            int a,b,c;
            for (int i = 0; i < r1; i++)
            {
                for (int j = 0; j < c1; j++)
                {
                    a= *(p1 + i*c1 + j);
                    b = *(p2 + i*c1 + j);
                    c = a + b;
                    *(matrix.p + i*c1 + j) = c;
                }
            }
        }
        else throw(c1);
    }
    catch(...)
    {
        cout<<"Addition is not possible at this time"<<endl;
        matrix.c = 0;
```

```cpp
        matrix.r = 0;
        matrix.p = NULL;
    }
    return matrix;
}

element substraction_matrces(int r1, int c1, int r2, int c2, int *p1, int
*p2)
{
    element matrix;
    try
    {
        if ((r1==r2) && (c1==c2))
        {
            cout<<"Processing subtraction of matrices"<<endl;
            matrix.c = c1;
            matrix.r = r1;
            matrix.p = new int [r1*c1];
            int a,b,c;
            for (int i = 0; i < r1; i++)
            {
                for (int j = 0; j < c1; j++)
                {
                    a= *(p1 + i*c1 + j);
                    b = *(p2 + i*c1 + j);
                    c = a - b;
                    *(matrix.p + i*c1 + j) = c;
                }
            }
        }
        else throw(c1);
    }
    catch(...)
    {
        cout<<"substraction is not possible at this time"<<endl;
        matrix.c = 0;
        matrix.r = 0;
        matrix.p = NULL;
    }
    return matrix;
}

void printing_matrix(int row, int col, int *ptr, string name)
{
    cout<<"Dear "<<name<<"! "<<"Matrix X has following contents: "<<endl;
    for (int i = 0; i < row; i++)
    {
        cout << "[";
        for (int j = 0; j < col; j++)
        {
            cout << " " << *ptr; // Corrected dereferencing here
            ptr = ptr + 1;
        }
        cout << " ]" << endl;
    }
    return;
}

void add_printing_matrix(int row, int col, int *ptr, string name)
{
    cout<<"Dear "<<name<<"! "<<"Z = X + Y is shown below:"<<endl;
    for (int i = 0; i < row; i++)
    {
        cout << "[";
        for (int j = 0; j < col; j++)
```

```cpp
        {
            cout << " " << *ptr; // Corrected dereferencing here
            ptr = ptr + 1;
        }
        cout << " ]" << endl;
    }
    return;
}

void sub_printing_matrix(int row, int col, int *ptr, string name)
{
    cout<<"Dear "<<name<<"! "<<"Z = X - Y is shown below:"<<endl;
    for (int i = 0; i < row; i++)
    {
        cout << "[";
        for (int j = 0; j < col; j++)
        {
            cout << " " << *ptr; // Corrected dereferencing here
            ptr = ptr + 1;
        }
        cout << " ]" << endl;
    }
    return;
}


int main()
{
    int r1, r2, c1, c2, *p1, *p2;
    string user;
    cout<<"Enter name of current user?: ";cin>>user;cout<<endl;
    cout<<"Dear "<<user<<"! "<<"Tell me what are the dimensions of X?: "<<endl;
    cin>>r1;
    cin>>c1;
    p1 = new int [r1*c1];
    cout<<"Dear "<<user<<"! "<<"Enter the contents of X:"<<endl;
    initialize_matrix(r1,c1,p1);
    printing_matrix(r1,c1,p1, user);

    cout<<"Dear "<<user<<"! "<<"Tell me what are the dimensions of Y?: "<<endl;
    cin>>r2;
    cin>>c2;
    p2 = new int [r2*c2];
    cout<<"Dear "<<user<<"! "<<"Enter the contents of Y:"<<endl;
    initialize_matrix(r2,c2,p2);
    printing_matrix(r2,c2,p2,user);

    element matrix1, matrix2;
    matrix1 = addition_matrces(r1,c1,r2,c2,p1,p2);
    add_printing_matrix(matrix1.r, matrix1.c, matrix1.p, user);

    matrix2 = substraction_matrces(r1,c1,r2,c2,p1,p2);
    sub_printing_matrix(matrix2.r, matrix2.c, matrix2.p, user);
    delete [] p1;
    delete [] p2;
    return 0;
}
```

Output:

```
Enter name of current user?: Ahmed

Dear Ahmed! Tell me what are the dimensions of X?:
2
2
Dear Ahmed! Enter the contents of X:
4
4
4
4
Dear Ahmed! Matrix X has following contents:
[ 4 4 ]
[ 4 4 ]
Dear Ahmed! Tell me what are the dimensions of Y?:
2
2
```

Fig: 01

```
Dear Ahmed! Enter the contents of Y:
3
2
2
3
Dear Ahmed! Matrix X has following contents:
[ 3 2 ]
[ 2 3 ]
Processing addition of matrices
Dear Ahmed! Z = X + Y is shown below:
[ 7 6 ]
[ 6 7 ]
Processing subtraction of matrices
Dear Ahmed! Z = X - Y is shown below:
[ 1 2 ]
[ 2 1 ]
```

Fig: 02

Methodology:

This code performs matrix addition and subtraction operations based on user input. It first prompts the user to enter their name and the dimensions of two matrices, X and Y. Then, it initializes memory space for the matrices and prompts the user to input their contents. After that, it calculates the addition and subtraction of the matrices, ensuring that they have the same dimensions.

For each operation, it dynamically allocates memory for the resultant matrix and performs the arithmetic calculations. It also handles exceptions if the dimensions of the matrices are incompatible for addition or subtraction. Finally, it prints the contents of the input matrices and the result matrices, along with a personalized message addressed to the user.

Memory allocated for the matrices is properly deallocated at the end to avoid memory leaks. Overall, the code is structured to facilitate matrix operations while providing clear feedback to the user throughout the process.

Lab 6: Matrices-II

Q1): Write the program that:

A function that initializes $Xm{\times}n$ and $Yl{\times}k$, where {m, n, l, k} are user defined and also its contents.

Code:

```
void initialize_matrix(int row, int col, int *ptr)
{
    int k=0;
    for(int i=0;i<row;i++)
    {
        for (int j = 0; j < col; j++)
        {
            cin>>k;
            *(ptr + (i*col)+ j) = k;
        }
    }
    return;
}
```
A function that evaluates $Z = X \times Y$.

Code:

```
element matrices_multiplication(int r1, int c1, int r2, int c2, int
*p1, int *p2)
{
    element matrix;
    if (c1 == r2)
    {
        matrix.c = c2;
        matrix.r = r1;
        matrix.p = new int [r1*c2];
        for (int i = 0; i < r1; i++)
        {
            for (int j = 0; j < c2; j++)
            {
                long long sum = 0;
                for (int k = 0; k < r2; k++)
                {
                    sum += *(p1 + i*r2 + k) * *(p2 + k * c2 + j);
                }
                *(matrix.p + i *c2 + j) =  sum;
                sum = 0;
            }

        }
    }
    else
    {
        cout << "Can't perform multiplication" << endl;
        matrix.c = 0;
        matrix.r = 0;
        matrix.p = NULL;
    }
    return matrix;
}
```
A function that evaluates $Z = \det(X)$.

Code:

```
double determinant(int m, int *p)
{
    double ans = 0;
    double inner_determinant, inner_sol;
    int a, b, c, d;
    if ((m == 1) || (m = 2))
    // stopping criteria
    {
        if (m == 1)
        {
            ans = *p;
        }
        else
        {
            a = *p;
            b = *(p+1);
            c = *(p+2);
            d = *(p+3);
            ans = (a*d) - (b*c);
        }
    }
    else
    {
        int n, l, sign, basic, element;
        n = 0;
        sign = +1;
        int *q;
        q = new int [(m-1)*(m-1)];
        for (int i = 0; i < m; i++)
        {
            l = 0;
            n = 0;
            basic = *(p+i);
            for (int j = 0; j < m; j++)
            {
                for (int k = 0; k < m; k++)
                {
                    element = *(p+l);
                    cout<<element<<" ";
                    if ((j==0) || (i==k));
                    else
                    {
                        *(q + n) = element;
                        n = n + 1;
                    }
                    l = l + 1;
                }
            }

            cout<<endl<<basic<<"x"<<endl;
            printing((m-1),(m-1), q);
            inner_determinant = determinant(m-1, q);
            inner_sol = sign * basic * inner_determinant;
            cout<<" sign "<<sign<<" x basic "<<basic<<" x Determinant
"<<inner_determinant<<" = "<<inner_sol<<endl;
            ans = ans + inner_sol;
            sign = sign * -1;
        }
        delete [] q;
    }
    return ans;
}
```

A function called 'print' that simply prints the answer.

Code:

```
void printing(int row, int col, int *ptr)
{
    for (int i = 0; i < row; i++)
    {
        cout << "[";
        for (int j = 0; j < col; j++)
        {
            cout << " " << *ptr; // Corrected dereferencing here
            ptr = ptr + 1;
        }
        cout << " ]" << endl;
    }
    return;
}
```

Output:

```
D:\6th Semester\DSA\Lab1\DSA_lab>matrix_det_and_multi.exe matrix_det_and_multi.cpp array_data.txt
```

Fig: 01 CMD Input

```
Resultant Matrix:
[ 6525 5778 3844 ]
[ 2388 2187 1373 ]
[ 4090 3708 4990 ]
Determinant: -6.62893e+006
```

Fig: 02 CMD Output

```
DSA_lab > ≡ array_data.txt
   1    54 89 54 12 31 45 87 45 55 10 9 45 63 54 8 7
```

Fig: 03 Input text file

Methodology:

This code implements matrix multiplication and determinant calculation. It begins by prompting the user to input the dimensions of two matrices. Memory space is then allocated for both matrices, and the user is asked to input their respective elements. The code then performs matrix multiplication if the number of columns in the first matrix matches the number of rows in the second matrix.

During multiplication, it dynamically allocates memory for the resultant matrix and computes each element using nested loops. The function for computing the determinant is also defined, recursively calculating the determinant of a matrix using cofactor expansion until it reaches the base case of a 1x1 or 2x2 matrix.

The determinant function handles matrix sizes recursively, calculating the determinant by recursively calling itself for submatrices. It prints intermediate steps, including individual determinants and their corresponding signs. Finally, it prints the resultant matrix and the calculated determinant.

Memory allocated for the matrices is properly deallocated at the end to prevent memory leaks. Overall, the code facilitates matrix multiplication and determinant calculation with clear user prompts and informative output messages throughout the process.

Lab 7: Single Linked lists

Q1): Write a program that implements the following:

Functions:

    b. Create () adding the very first node.

```
void creating_node(int value)
{
    try
    {
        if (start == NULL)
        {
            start=new node;
            start->data=value;
            start->next=NULL;
            length=length+1;
        }
        else
        {
            throw(value);
        }
    }
    catch(int data)
    {
        cout<<"Linked list already exist, Please try to add instead
of creating linked list"<<endl;
    }
    return;
}
```

    c. Add() the method should insert numbers in sorted order.

```
void add(int value)
{
    if (length==0)
    {
        cout<<"There is no sign of linked list, Try to create linked
list first"<<endl;
    }
    else
    {
```

```cpp
        node *temp;

        temp = start;

        while (temp->next!=NULL)

        {

              temp=temp->next;

        }

        node * new_node;

        new_node = new node;

        new_node->data=value;

        new_node->next=NULL;

        temp->next=new_node;

        length = length+1;

    }


    return;

}
```

d. Delete() the function should clearly show, with appropriate comments, deleting a node (if it is present) by enumerating all the elements of the list after deleting the concerned node, and "not found" if the requested node is not present in the list.

```cpp
void deleting(int value)

{

    try

    {

        if (length==0)

        {

            throw(1);

            cout<<"Linked list is empty, deleting is not
possible"<<endl;

        }

        else

        {

            node *current, *prev;

            current = start;

            while ((current->next!=NULL) && (current->data!= value))

            {

                prev = current;

                current = current->next;

            }
```

```
        if (current->data==value)
        {
                // if it is first node then
                if (current==start)
                {
                        start = current->next;
                }
                else
                {
                        prev->next=current->next;
                }
                delete current;
                length=length-1;
            }
        }
        cout<<"Noting to delete"<<endl;
    }
    catch(...)
    {
        cout<<"Node having value ["<<value<<"] not found, Hence
nothing to delete"<<endl;
    }
    return;
}
```

e. Traverse() informs the user of the length of the linked list

```
int sized_of_linked_list()
{
    int size=1;
    node *temp = start;
      while (temp->next!=NULL)
      {
        temp=temp->next;
        size = size +1;
    }
    return size;
}
```

e. Traverse () simply prints the elements of the list.

```cpp
void traverse()
{
    try
    {
        if (length==0)
        {
            throw(1);
        }
        else
        {
            node *temp;
            temp = start;
            int len=1;
            while (temp->next!=NULL)
            {
                cout<<"["<<len<<"]: "<<temp->data<<" "<<endl;
                temp=temp->next;
                len = len +1;
            }
            cout<<"["<<len<<"]: "<<temp->data<<" "<<endl;
        }
    }
    catch(...)
    {
        cout<<"Linked list is empty, Try create and add then
traverse"<<endl;
    }

    return;
}
```

A main( ) function that:

  a. Text calls upon an external file containing the numbers [1, 3, 7, 99, 101, 103,107] (you can create a text file yourself.
  b. creates and populates the linked list with all the elements presented to it in the text file of part (2a)
  c. thereafter, the function asks the user a series of actions that he/she would like to continue to perform iteratively up until he/she chooses to stop. Actions are:
     (i) add a node
     (ii) delete a node
     (iii) enumerate the elements of the list

(iv) length of the list
(v) end – stop the process and stores the current linked list in an external file called
"output.txt"

Code:

```cpp
int main()

{

    ifstream inputFile("sli.txt");

    if (inputFile.is_open()) {

        int num;

        while (inputFile >> num) {

            if (start == NULL) {

                creating_node(num);

            } else {

                add(num);

            }

        }

        inputFile.close();

    } else {

        cout << "Unable to open numbers.txt" << endl;

        return 1;

    }

    int choice;

    do

    {

        cout<<endl;

        cout << "Choose any single action:" << endl;

        cout << "[1] Add a node?" << endl;

        cout << "[2] Delete a node?" << endl;

        cout << "[3] Enumerate the elements of the list?" << endl;

        cout << "[4] Length of the list?" << endl;

        cout << "[5] End and store the current linked list?" << endl;

        cout<<"----------------------------------------------------------------------------------"<<endl;

        cout << "Enter your choice: ";cin>>choice;cout<<endl;

        switch (choice)

        {

        case 1:{

            int input_value;
```

```cpp
                cout<<"Enter the value of node:
    ";cin>>input_value;cout<<endl;

                add(input_value);

                break;

        }

        case 2:{

                int input_value;

                cout<<"Enter the value of node that you want to delete:
    ";cin>>input_value;cout<<endl;

                deleting(input_value);

                break;

        }

        case 3:{

                traverse();

                break;

        }

        case 4:{

                cout<<"Length of the single linked list:
    "<<size_of_linked_list()<<endl;

                break;

        }

        case 5:{

                saveListToFile();

                break;

        }

        default:

                cout << "Invalid choice. Please try again." << endl;

        }

    }while(choice!=5);

    return 0;

    }
```

Output:

Command Prompt:

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\AhmedShafique>D:

D:\>cd "6th Semester"

D:\6th Semester>cd DSA

D:\6th Semester\DSA>cd Lab1

D:\6th Semester\DSA\Lab1>cd DSA_lab

D:\6th Semester\DSA\Lab1\DSA_lab>single_ll.exe single_ll.cpp sli.txt
```

Fig: 01 Programming running on CMD

```
Choose any single action:
[1] Add a node?
[2] Delete a node?
[3] Enumerate the elements of the list?
[4] Length of the list?
[5] End and store the current linked list?
------------------------------------------------------------------------
Enter your choice: |
```

Fig: 02 Accepting Options

```
Enter your choice: 3

[1]: 1
[2]: 3
[3]: 9
[4]: 99
[5]: 101
[6]: 103
[7]: 107
```

Fig: 03 Traversing Linked list

```
Choose any single action:
[1] Add a node?
[2] Delete a node?
[3] Enumerate the elements of the list?
[4] Length of the list?
[5] End and store the current linked list?
------------------------------------------------------------------------
Enter your choice: 1

Enter the value of node: 48
```

Fig: 04 Adding node

```
Choose any single action:
[1] Add a node?
[2] Delete a node?
[3] Enumerate the elements of the list?
[4] Length of the list?
[5] End and store the current linked list?
--------------------------------------------------------------------------
Enter your choice: 2

Enter the value of node that you want to delete: 48
```

Fig: 05 Deleting node

```
Choose any single action:
[1] Add a node?
[2] Delete a node?
[3] Enumerate the elements of the list?
[4] Length of the list?
[5] End and store the current linked list?
--------------------------------------------------------------------------
Enter your choice: 4

Length of the single linked list: 7
```

Fig: 06 Linked lists length

```
Choose any single action:
[1] Add a node?
[2] Delete a node?
[3] Enumerate the elements of the list?
[4] Length of the list?
[5] End and store the current linked list?
--------------------------------------------------------------------------
Enter your choice: 5

Linked list saved to output.txt
```

Fig: 07 Saving text file

Methodology:

First, we added required header files to run the program which are:

#include <iostream>

#include <new>

#include <fstream>

using namespcace std;

The according to the requirement of making structure of linked list we code the structure of linked list which are as follows:

struct node

{

    int data;

    node *next;

};

Then we initialize the linked list to zero because there is no sign of linked list initially using following code:

```
int length = 0;
```

```
node *start = NULL;
```

We declare them as global because now these can easily calls in any function so there is no need to additionally passed them into function as a argument. Then we coded the `creating_node` function to create the node as user input text file. Then we coded `traverse` function, the traverse function prints the value in whole linked list and we set that if there no node then it return nothing and we use `while` loop and pass the condition that it stops where the next structure pointer points to `NULL`.

We coded the `add` function to add additional nodes from the user and then, we coded another function called `deleting` to delete the node of linked list that user want to delete based on the value of the node input by the user.

We coded `size_of_linked_list` function to calculate the length of linked list, it same as some portion of `traverse` function but it have one statement different from traverse which is:

```
length = length + 1;
```

then it returning length using `return length;`

At last we coded the last part of program in which user can save the linked list data into the output text file. So, We initialize the file input in main function then we pass one condition that if there is no sign of linked list and it initialize to zero then it calls the `creating_node` function to create the node in the linked list and initialize it to `1`. `Else` it add the nodes in the linked list using `add` function.

Also we give choice to user to save it linked lists data that the user ran into output text file using `ofstream`, here is the code below:

```
void saveListToFile() {
    ofstream outputFile("slo.txt");
    if (outputFile.is_open()) {
        node *temp = start;
        while (temp != nullptr) {
            outputFile << temp->data << " ";
            temp = temp->next;
        }
        outputFile.close();
        cout << "Linked list saved to output.txt" << endl;
    } else {
        cout << "Unable to open output.txt for writing" << endl;
    }
}
```

Lab: 08 Double Linked lists

Q1. Write a program that implements the following:

Functions:

a) create(): adding the very first node

```
void create (int value) {
 if (length == 0) {
      start = new node;
    start->data = value;
    start->previous = NULL; // Use nullptr instead of NULL for
better clarity
    start->next = NULL;
    length = length + 1; // Don't forget semicolon at the end of the
statement
 }
 else {
      cout << "List not empty. Try Again." << endl; // Fixed the error
message string
 }
    return;
}
```

b) add (): the method should insert numbers in sorted order.

```
void add(int value) {
    node* newNode = new node;
    newNode->data = value;
    newNode->previous = NULL;
    newNode->next = start;
    if (start != NULL)
        start->previous = newNode;
    start = newNode;
    length = length + 1;
}
```

c) delete () : the function should clearly show, with appropriate comments, deleting a node (if it is present), and "not found" if the requested node is absent.

```
void deleteNode(int value) {
    if (length == 0) {
        cout << "List is empty." << endl;
```

```cpp
            return;
        }


        node* current = start;
        node* previous = NULL;
        while (current != NULL) {
            if (current->data == value) {
                if (current == start) {
                    start = current->next;
                    if (start != NULL)
                        start->previous = NULL;
                } else {
                    previous->next = current->next;
                    if (current->next != NULL)
                        current->next->previous = previous;
                }
                delete current;
                length=length-1;
                cout << "Node with value " << value << " deleted." <<
    endl;
                return;
            }
            previous = current;
            current = current->next;
        }
        cout << "Node with value " << value << " not found." << endl;
    }
```

d) length() : informs the user of the length of the linked list

```cpp
    int length_of_linked_list() {
        return length;
    }
```

e) traverse() simply prints the elements of the list either in increasing or decreasing order

```cpp
    void traverse(bool increasingOrder) {
        if (length == 0) {
            cout << "List is empty." << endl;
            return;
        }
```

```
        node* current;

        if (increasingOrder) {

            current = start;

            cout << "List elements in increasing order: ";

            while (current != NULL) {

                cout << current->data << " ";

                current = current->next;

            }

        } else {

            current = start;

            while (current->next != NULL)

                current = current->next;

            cout << "List elements in decreasing order: ";

            while (current != NULL) {

                cout << current->data << " ";

                current = current->previous;

            }

        }

        cout << endl;

    }
```

A main( ) function that:

   a. Text calls upon an external file containing the numbers [1, 3, 7, 99, 101, 103,107] (you can create a text file yourself.

   b. creates and populates the linked list with all the elements presented to it in the text file of part (2a)

   c. thereafter, the function asks the user a series of actions that he/she would like to continue to perform iteratively up until he/she chooses to stop. Actions are:
   (i) add a node
   (ii) delete a node
   (iii) enumerate the elements of the list
   (iv) length of the list
   (v) end – stop the process and stores the current linked list in an external file called "output.txt"

Code:

```
int main() {

    ifstream inputFile("sli.txt");

    if (inputFile.is_open()) {

        int num;

        while (inputFile >> num) {
```

```cpp
            if (start == NULL) {
                create(num);
            } else {
                add(num);
            }
        }
        inputFile.close();
    } else {
        cout << "Unable to open numbers.txt" << endl;
        return 1;
    }
    // User interaction loop
    int choice;
    do {
        cout << "-----------------------------------------------------------------" << endl;
        cout << "Dear User " << endl;
        cout << "Choose an action:" << endl;
        cout << "[1]. Add a node" << endl;
        cout << "[2]. Delete a node" << endl;
        cout << "[3]. Enumerate the elements of the list" << endl;
        cout << "[4]. Length of the list" << endl;
        cout << "[5]. End and save to file" << endl;
        cout << "Enter your choice: ";cin >> choice;cout<<endl;
        switch (choice) {
            case 1: {
                int newValue;
                cout << "Enter the value to add: ";cin >> newValue;cout<<endl;
                add(newValue);
                break;
            }
            case 2: {
                int newValue;
                cout << "Enter the value to add: ";cin >> newValue;cout<<endl;
                deleteNode(newValue);
                break;
            }
```

```
        case 3: {
                int order;
                cout << "Enumerate the elements of the list:" <<
endl;
                cout << "1. Deccreasing order" << endl;
                cout << "2. Increasing order" << endl;
                cout << "Enter your choice: ";cin >>
order;cout<<endl;
                traverse(order == 1);
                break;
        }
        case 4: {
                cout << "Length of the list: " <<
length_of_linked_list() << endl;
                break;
        }
        case 5: {
                saveListToFile();
                cout << "Linked list saved to output.txt. Exiting..."
<< endl;
                break;
        }
        default:
                cout << "Invalid choice. Please enter a valid
option." << endl;
            }
        } while (choice != 5);


        return 0;
    }
```

Output:

```
Command prompt:
```

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\AhmedShafique>D:

D:\>cd "6th Semester"

D:\6th Semester>cd DSA

D:\6th Semester\DSA>cd Lab1

D:\6th Semester\DSA\Lab1>cd DSA_lab

D:\6th Semester\DSA\Lab1\DSA_lab>double_ll.exe double_ll.cpp dli.txt
```

Fig:01 Running on command prompt

```
--------------------------------------------------------------------
Dear User
Choose an action:
[1]. Add a node
[2]. Delete a node
[3]. Enumerate the elements of the list
[4]. Length of the list
[5]. End and save to file
Enter your choice: |
```

Fig: 02 Accepting user choice

```
--------------------------------------------------------------------
Dear User
Choose an action:
[1]. Add a node
[2]. Delete a node
[3]. Enumerate the elements of the list
[4]. Length of the list
[5]. End and save to file
Enter your choice: 3

Enumerate the elements of the list:
1. Deccreasing order
2. Increasing order
Enter your choice: 1

List elements in increasing order: 107 103 101 99 9 3 1
```

Fig: 03 Traversing decreasingly

```
--------------------------------------------------------------------
Dear User
Choose an action:
[1]. Add a node
[2]. Delete a node
[3]. Enumerate the elements of the list
[4]. Length of the list
[5]. End and save to file
Enter your choice: 3

Enumerate the elements of the list:
1. Deccreasing order
2. Increasing order
Enter your choice: 2

List elements in decreasing order: 1 3 9 99 101 103 107
```

Fig: 04 Traversing Increasingly

```
--------------------------------------------------------------
Dear User
Choose an action:
[1]. Add a node
[2]. Delete a node
[3]. Enumerate the elements of the list
[4]. Length of the list
[5]. End and save to file
Enter your choice: 1

Enter the value to add: 45
```

Fig: 05 Adding node

```
--------------------------------------------------------------
Dear User
Choose an action:
[1]. Add a node
[2]. Delete a node
[3]. Enumerate the elements of the list
[4]. Length of the list
[5]. End and save to file
Enter your choice: 2

Enter the value to add: 45

Node with value 45 deleted.
```

Fig: 06 Deleting node

```
--------------------------------------------------------------
Dear User
Choose an action:
[1]. Add a node
[2]. Delete a node
[3]. Enumerate the elements of the list
[4]. Length of the list
[5]. End and save to file
Enter your choice: 4

Length of the list: 8
```

Fig: 07 Linked lists length

```
--------------------------------------------------------------
Dear User
Choose an action:
[1]. Add a node
[2]. Delete a node
[3]. Enumerate the elements of the list
[4]. Length of the list
[5]. End and save to file
Enter your choice: 5

Linked list saved to output.txt. Exiting...
```

Fig: 08 Saving Output text file

Methodology:

First, we added required header files to run the program which are:

```
#include <iostream>

#include <new>

#include <fstream>

using namespcace std;
```

According to the requirement of making structure for double linked list we code the structure of double linked list which is as follows:

```
struct node

{

     int data ;

    node* previous ;

    node * next ;

};
```

Then we initialize the double linked list to zero because there is no sign of double linked list initially using following code:

```
int length = 0;

node* start = NULL;
```

We declare them as global because now these can easily calls in any function so there is no need to additionally passed them into function as a argument. Then we coded the `create( )` function to create the node as user input text file. Then we coded `traverse( )` function, the traverse function prints the value in whole linked list and we set that if there no node then it return nothing and we use while loop and pass the condition that it stops where the next structure pointer points to `NULL`.

We coded the `add( )` function to add additional nodes from the user and then, we coded another function called `delete ( )` to delete the node of double linked list that user want to delete based on the value of the node input by the user.

We coded `size_of_linked_list` function to calculate the length of our double linked list, it same as some portion of traverse function.

At last we coded the last part of program in which user can save the double linked list data into the output text file. So, We initialize the file input in main function then we pass one condition that if there is no sign of linked list and it initialize to zero then it calls the `create` function to create the node in the linked list and initialize it to 1. Else it `add` the nodes in the double linked list using add function.

Also we give choice to user to save it linked lists data that the user ran into output text file using `ofstream`, here is the code below:

```
void saveListToFile() {

    ofstream outputFile("dlo.txt");

    if (outputFile.is_open()) {

        node *temp = start;

        while (temp != nullptr) {

            outputFile << temp->data << " ";

            temp = temp->next;
```

```
        }
        outputFile.close();
    } else {
        cout << "Unable to open output.txt for writing" << endl;
    }
}
```

Lab 10 &11: Sorting

Q1): Write a program that implements the following functions:

Functions:

   a. Selection_sort().

```
void selectionSort(int arr[], int lenght)
{
    int min_idx;

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

        // Find the minimum element in
        // unsorted array
        min_idx = i;
        for (int j = i + 1; j < lenght; j++) {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }

        // Swap the found minimum element
        // with the first element
        if (min_idx != i)
            swap(arr[min_idx], arr[i]);
    }
}
```

   b. Bubble_sort().

```
 int bubbleSort(int arr[], int length)
{
    int temp;
    for (int j = 0; j < length; j++)
    {
        int swap = 0;
        for (int i = 0; i < length-1; i++)
        {
            if (arr[i] > arr[i+1])
            {
                temp = arr[i];
                arr[i]= arr[i+1];
                arr[i+1]= temp;
                swap = swap + 1;
            }
        }
    }
}
```

   c. Insertion_sort().
```
void insertionSort(int arr[], int length)
{
    int i, j, key;
    for (i = 1; i < length; i++) {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key,
        // to one position ahead of their
        // current position
```

```
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }
```

d. Merge_sort().

```
    void mergesort(int arr[], int left, int mid, int right)
    {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        int* left_half = new int[n1];
        int* right_half = new int[n2];

        // Copy data to temp arrays left_half[] and right_half[]
        for (int i = 0; i < n1; i++)
            left_half[i] = arr[left + i];
        for (int j = 0; j < n2; j++)
            right_half[j] = arr[mid + 1 + j];

        // Merge the temp arrays back into arr[l..r]
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2)
        {
            if (left_half[i] <= right_half[j])
            {
                arr[k] = left_half[i];
                i++;
            }
            else
            {
                arr[k] = right_half[j];
                j++;
            }
            k++;
        }

        // Copy the remaining elements of left_half[], if any
        while (i < n1)
        {
            arr[k] = left_half[i];
            i++;
            k++;
        }

        // Copy the remaining elements of right_half[], if any
        while (j < n2)
        {
            arr[k] = right_half[j];
            j++;
            k++;
        }

        delete[] left_half;
        delete[] right_half;
    }

    // for recursively call merge sort because its recursive sort
    // cutting array recursively
    void merge_sort(int arr[], int left, int right)
    {
```

```
            if (left<right)
            {
                int mid = left+(right-left)/2;
                merge_sort(arr, left, mid);
                merge_sort(arr, mid + 1, right);
                mergesort(arr,left,mid,right);
            }

        }
```
e. Quick_sort().

```
        int partition(int arr[], int start, int end)
        {
            int pivot = arr[start];
            int count = 0;
            for (int i = start + 1; i <= end; i++) {
                if (arr[i] <= pivot)
                    count++;
            }
            // Giving pivot element its correct position
            int pivotIndex = start + count;
            swap(arr[pivotIndex], arr[start]);
            // Sorting left and right parts of the pivot element
            int i = start, j = end;
            while (i < pivotIndex && j > pivotIndex) {

                while (arr[i] <= pivot) {
                    i++;
                }

                while (arr[j] > pivot) {
                    j--;
                }

                if (i < pivotIndex && j > pivotIndex) {
                    swap(arr[i++], arr[j--]);
                }
            }
            return pivotIndex;
        }

        void quickSort(int arr[], int start, int end)
        {
            // base case
            if (start >= end)
                return;
            // partitioning the array
            int p = partition(arr, start, end);
            // Sorting the left part
            quickSort(arr, start, p - 1);
            // Sorting the right part
            quickSort(arr, p + 1, end);
        }
```
f. Heap_sort()
```
        void heapify(int arr[], int length, int index) {
            int largest = index;
            int left = 2 * index + 1;
            int right = 2 * index + 2;

            if (left < length && arr[left] > arr[largest]) {
                largest = left;
            }

            if (right < length && arr[right] > arr[largest]) {
```

```
                largest = right;
            }

            if (largest != index) {
                swap(arr[index], arr[largest]);
                heapify(arr, length, largest);
            }
        }

        void heapsort(int arr[], int length) {
            // Build heap (rearrange array)
            for (int i = length / 2 - 1; i >= 0; i--) {
                heapify(arr, length, i);
            }

            // One by one extract an element from heap
            for (int i = length - 1; i >= 0; i--) {
                // Move current root to end
                swap(arr[0], arr[i]);

                // call max heapify on the reduced heap
                heapify(arr, i, 0);
            }
        }
```

g. Radix_sort() & Count_sort()

```
        int getMax(int arr[], int n)
        {
            int mx = arr[0];
            for (int i = 1; i < n; i++)
                if (arr[i] > mx)
                    mx = arr[i];
            return mx;
        }

        void countSort(int arr[], int length, int exp)
        {
            // output array
            int output[length];
            int i, count[10] = { 0 };

            // Store count of occurrences in count[]
            for (i = 0; i < length; i++)
                count[(arr[i] / exp) % 10]++;

            // Change count[i] so that count[i] now contains actual
            // position of this digit in output[]
            for (i = 1; i < 10; i++)
                count[i] += count[i - 1];

            // Build the output array
            for (i = length - 1; i >= 0; i--) {
                output[count[(arr[i] / exp) % 10] - 1] = arr[i];
                count[(arr[i] / exp) % 10]--;
            }

            // Copy the output array to arr[], so that arr[] now
            // contains sorted numbers according to current digit
            for (i = 0; i < length; i++)
                arr[i] = output[i];
        }

        // The main function to that sorts arr[] of size n using
        // Radix Sort
```

```
void radixsort(int arr[], int length)
{
    // Find the maximum number to know number of digits
    int m = getMax(arr, length);
    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is 10^i
    // where i is current digit number
    for (int exp = 1; m / exp > 0; exp *= 10)
        countSort(arr, length, exp);
}
```

h. Bucket_sort()

```
void bucketsort(float arr[], int length) {

const int bucketSize = 10; // Define the number of buckets

float maxVal = *max_element(arr, arr + length);

// Find the maximum value in the array

float minVal = *min_element(arr, arr + length);

// Find the minimum value in the array

// Create an array of buckets

float buckets[bucketSize][length];

// Initialize buckets

for (int i = 0; i < bucketSize; i++) {

    fill_n(buckets[i], length, -1);

// Initialize each bucket with -1

}

// Scatter the array elements into buckets

for (int i = 0; i < length; i++) {

    int index = (int)((arr[i] - minVal) / (maxVal -     minVal) *
(bucketSize - 1));

    int j = 0;

    while (buckets[index][j] != -1) {

        j++;

    }

    buckets[index][j] = arr[i];

}


    // Sort individual buckets and merge them back into the original
array

    int index = 0;

    for (int i = 0; i < bucketSize; i++) {

        InsertionSort(buckets[i], length);

        for (int j = 0; j < length; j++) {

            if (buckets[i][j] != -1) {
```

```
                arr[index++] = buckets[i][j];
            }
        }
    }
}
```

Q2) Main Function:

```
int main()
{
    int n, store[500];
    const char* file_open = "array_data.txt";
    readdata(file_open, store, n);
    printing(store, n);



    cout<<"HeapSort Applying................."<<endl;
    // Measure run-time for HeapSort
    auto start = high_resolution_clock::now();


    heapsort(store, n);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(stop - start);
    cout << "HeapSort Time: " << duration.count() << " milliseconds"
<< endl;
    printing_sorted_arrays(store,n);



    cout<<"CountSort Applying................."<<endl;
    start = high_resolution_clock::now();
    CountSort(store, n);
    stop = high_resolution_clock::now();
    duration = duration_cast<milliseconds>(stop - start);
    cout << "CountSort Time: " << duration.count() << " milliseconds"
<< endl;
    printing_sorted_arrays(store,n);



    cout<<"RadixSort Applying................."<<endl;
    start = high_resolution_clock::now();
```

```cpp
    radixsort(store, n);

    stop = high_resolution_clock::now();

    duration = duration_cast<milliseconds>(stop - start);

    cout << "RadixSort Time: " << duration.count() << " milliseconds"
<< endl;

    printing_sorted_arrays(store,n);




    cout<<"BubbleSort Applying................"<<endl;

    start = high_resolution_clock::now();

    bubbleSort(store, n);

    stop = high_resolution_clock::now();

    duration = duration_cast<milliseconds>(stop - start);

    cout << "BubbleSort Time: " << duration.count() << "
milliseconds" << endl;

    printing_sorted_arrays(store,n);




    cout<<"QuickSort Applying................"<<endl;

    start = high_resolution_clock::now();

    quickSort(store, 0, n - 1);

    stop = high_resolution_clock::now();

    duration = duration_cast<milliseconds>(stop - start);

    cout << "QuickSort Time: " << duration.count() << " milliseconds"
<< endl;

    printing_sorted_arrays(store,n);




    cout<<"BucketSort Applying................"<<endl;

    start = high_resolution_clock::now();

    int len = 9;

    float array[len] = {8.48, 5.27, 9.10, 7.89, 3.01, 2.48, 6.32,
1.95, 1.27};

    bucketsort(array, n);

    stop = high_resolution_clock::now();

    duration = duration_cast<milliseconds>(stop - start);

    cout << "BucketSort Time: " << duration.count() << "
milliseconds" << endl;

    fprinting_sorted_arrays(array,len);
```

```
        cout<<"InsertionSort Applying................."<<endl;

        start = high_resolution_clock::now();

        insertionSort(store, n);

        stop = high_resolution_clock::now();

        duration = duration_cast<milliseconds>(stop - start);

        cout << "InsertionSort Time: " << duration.count() << "
milliseconds" << endl;

        printing_sorted_arrays(store,n);




        cout<<"MergeSort Applying................."<<endl;

        start = high_resolution_clock::now();

        merge_sort(store,0,n-1);

        stop = high_resolution_clock::now();

        duration = duration_cast<milliseconds>(stop - start);

        cout << "MergeSort Time: " << duration.count() << " milliseconds"
<< endl;

        printing_sorted_arrays(store,n);




        cout<<"SelectionSort Applying................."<<endl;

        start = high_resolution_clock::now();

        selectionSort(store, n);

        stop = high_resolution_clock::now();

        duration = duration_cast<milliseconds>(stop - start);

        cout << "SelectionSort Time: " << duration.count() << "
milliseconds" << endl;

        printing_sorted_arrays(store,n);

        return 0;

    }
```

Methodology:

Certainly! Our code executes various sorting algorithms on an array of integers read from a file, measuring their execution times using `std::chrono`. After sorting, it prints the sorted arrays along with the time taken by each algorithm. To enhance readability and efficiency, you can use concise naming conventions, organize the code into functions for each sorting algorithm, and optimize where possible. For visualizing the results, you can plot execution time against the number of entries using a logarithmic scale for the x-axis, with each algorithm represented by a different color. This approach helps in comparing the performance of sorting algorithms and understanding their efficiency for different input sizes.

Header files:

```cpp
#include <iostream>

#include <fstream>

#include <algorithm>

#include <chrono>

#include <time.h>

#include <random>

using namespace std;

using namespace std::chrono;
```

Random Data Generator:

```cpp
void generateRandomData(int arr[], int size) {

    random_device rd;

    mt19937 gen(rd());

    uniform_int_distribution<> dis(1, 1000);

    for (int i = 0; i < size; ++i) {

        arr[i] = dis(gen);

    }

}
```

Saving time data in CSV file:

```cpp
void saveTimingData(const string& filename, const string& algorithm, int
size, long long duration) {

    ofstream file(filename, ios::app);

    if (file.is_open()) {

        file << algorithm << "," << size << "," << duration << endl;

        file.close();

    } else {

        cout << "Unable to open file: " << filename << endl;

    }

}
```

Results:

Outputs on terminal window of vscode.

```
[0]: 8
[1]: 7
[2]: 6
[3]: 8
[4]: 5
[5]: 10
[6]: 27
[7]: 68
[8]: 55
[9]: 99
HeapSort Applying................
HeapSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
CountSort Applying................
CountSort Time: 0 milliseconds
```

Fig:01

```
CountSort Applying................
CountSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
RadixSort Applying................
RadixSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
BubbleSort Applying................
BubbleSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
QuickSort Applying................
QuickSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
BucketSort Applying................
BucketSort Time: 0 milliseconds
[0]: 5.89505e-039[1]: 1.27[2]: 1.95[3]: 2.48[4]: 3.01[5]: 5.27[6]: 6.32[7]: 7.89[8]: 8.48
```

Fig:02

```
InsertionSort Applying................
InsertionSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
MergeSort Applying................
MergeSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
SelectionSort Applying................
SelectionSort Time: 0 milliseconds
[0]: 5[1]: 6[2]: 7[3]: 8[4]: 8[5]: 10[6]: 27[7]: 55[8]: 68[9]: 99
PS D:\6th Semester\DSA\Lab1\DSA_lab>
```

Fig:03

Python Graph: