SUBMITTED TO: SIR USMAN JOYIA

DATA STRUCTURES

ASSIGNEMENT 1

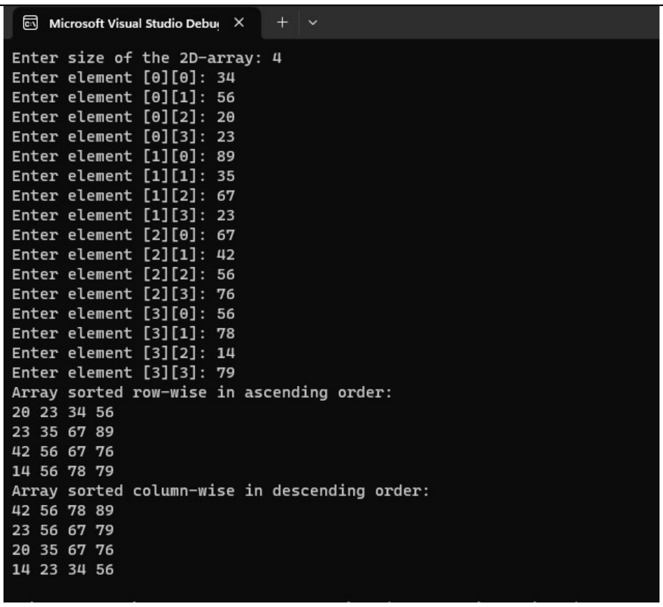
KHUBAIB SHABBIR
F219634
CS-4E

QUESTION NO 1:

CODE:

```
#include <iostream>

using namespace std;

// Time complexity for populating the 2D-array is O(N^2)
// where N is the size of the array.
int** populateArray(int N) {
    int** arr = new int* [N];
    for (int i = 0; i < N; i++) {
        arr[i] = new int[N];
        for (int j = 0; j < N; j++) {
            cout << "Enter element [" << i << "][" << j << "]: ";
            cin >> arr[i][j];
        }
    }
    return arr;
}

// Time complexity for sorting each row in ascending order is O(N^3)
// where N is the size of the row. Since we need to sort N rows, the overall time
complexity
// is O(N^4).
void sortArrayAscending(int** arr, int N) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N - 1; j++) {
            int min_index = j;
            for (int k = j + 1; k < N; k++) {
                if (arr[i][k] < arr[i][min_index]) {
                    min_index = k;
                }
            }
            if (min_index != j) {
                int temp = arr[i][j];
                arr[i][j] = arr[i][min_index];
                arr[i][min_index] = temp;
            }
        }
    }
}

// Time complexity for sorting each column in descending order is O(N^3)
// where N is the size of the column. Since we need to sort N columns, the overall time
complexity
// is O(N^4).
void sortArrayDescending(int** arr, int N) {
    for (int j = 0; j < N; j++) {
        for (int i = 0; i < N - 1; i++) {
            int max_index = i;
            for (int k = i + 1; k < N; k++) {
                if (arr[k][j] > arr[max_index][j]) {
                    max_index = k;
                }
            }
            if (max_index != i) {
                int temp = arr[i][j];
                arr[i][j] = arr[max_index][j];
                arr[max_index][j] = temp;
            }
        }
    }
}

// The overall time complexity for part (a) is O(N^2) and for part (b) and (c) is
O(N^4).
// Therefore, the overall combined asymptotic time upper-bound for part (a), (b), and
(c)
// is O(N^4).
int main() {
    int N;
    cout << "Enter size of the 2D-array: ";// Here, the variable N represents the size
of both dimensions of the 2D array. Since we are working with a square 2D array,
                                          //we only need to take one size as input from
the user. This input size is then used to create a 2D array of size N x N using
                                           //dynamic memory allocation
    cin >> N;

    int** arr = populateArray(N);

    cout << "Array sorted row-wise in ascending order:" << endl;
    sortArrayAscending(arr, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    cout << "Array sorted column-wise in descending order:" << endl;
    sortArrayDescending(arr, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
    for (int i = 0; i < N; i++) {
        delete[] arr[i];
    }
    delete[] arr;
    return 0;
}
```

SCREENSHOT:


Microsoft Visual Studio Debu

```
Enter size of the 2D-array: 4
Enter element [0][0]: 34
Enter element [0][1]: 56
Enter element [0][2]: 20
Enter element [0][3]: 23
Enter element [1][0]: 89
Enter element [1][1]: 35
Enter element [1][2]: 67
Enter element [1][3]: 23
Enter element [2][0]: 67
Enter element [2][1]: 42
Enter element [2][2]: 56
Enter element [2][3]: 76
Enter element [3][0]: 56
Enter element [3][1]: 78
Enter element [3][2]: 14
Enter element [3][3]: 79
Array sorted row-wise in ascending order:
20 23 34 56
23 35 67 89
42 56 67 76
14 56 78 79
Array sorted column-wise in descending order:
42 56 78 89
23 56 67 79
20 35 67 76
14 23 34 56
```

QUESTION NO 2:

BIG O- NOTATION ANALYSIS

EXPRESSION	DOMINANT TERM(s)	T(n) = O(?)
T(n) = 3n2 + 1000n + 360	3n^2	O(n^2)
T(n) = 120n + 4n1.1 + 3	4n^1.1	O(n^1.1)
T(n) = 50 log(n) + 0.3 log(n)2 + 25	0.3 log(n)^2	O(log(n)^2)
T(n) = 30n2 + 45n! + 66	45n!	O(n!)
T(n) = n.log(n) + 15log(n)2 +4	n.log(n)	O(n.log(n))

QUESTION NO 3:

TIME COMPLEXITY ANALYSIS

a. The provided code is an implementation of selection sort algorithm. The outer loop iterates n-1 times, and the inner loop iterates (n-i) times for each i. The inner loop contains a conditional statement that compares two elements, which takes constant time. Therefore, the time function T(n) can be expressed as:

T(n) = (n-1) * (n-1+ n-2 + ... + 1) * c = (n-1) * n/2 * c = O(n^2)

where c is a constant.

b. The outer loop iterates log(n) times, and the inner loop iterates i times for each i, which is a power of 2 less than n. Therefore, the time function T(n) can be expressed as:

T(n) = 1 + 2 + 4 + ... + n/2 + n = 2n - 1 = O(n)

c. In the first loop, the steps are less to cover and jump is greater to reach 'n', as the iteration 'i' is multiplying at every iteration with 2 to make the jump bigger and moreover in the second loop the iterations are dependent on 'i' , therefore, the time function T(n) can be expressed as

T(n) = n = O(logn)

d. In the best case, n is even, and the code executes the outer loop only once, and the inner loop n times. Therefore, the time function T(n) can be expressed as:

T(n) = n = O(n)

In the worst case, n is odd, and the code executes the outer loop and the inner loop n times. Therefore, the time function T(n) can be expressed as:

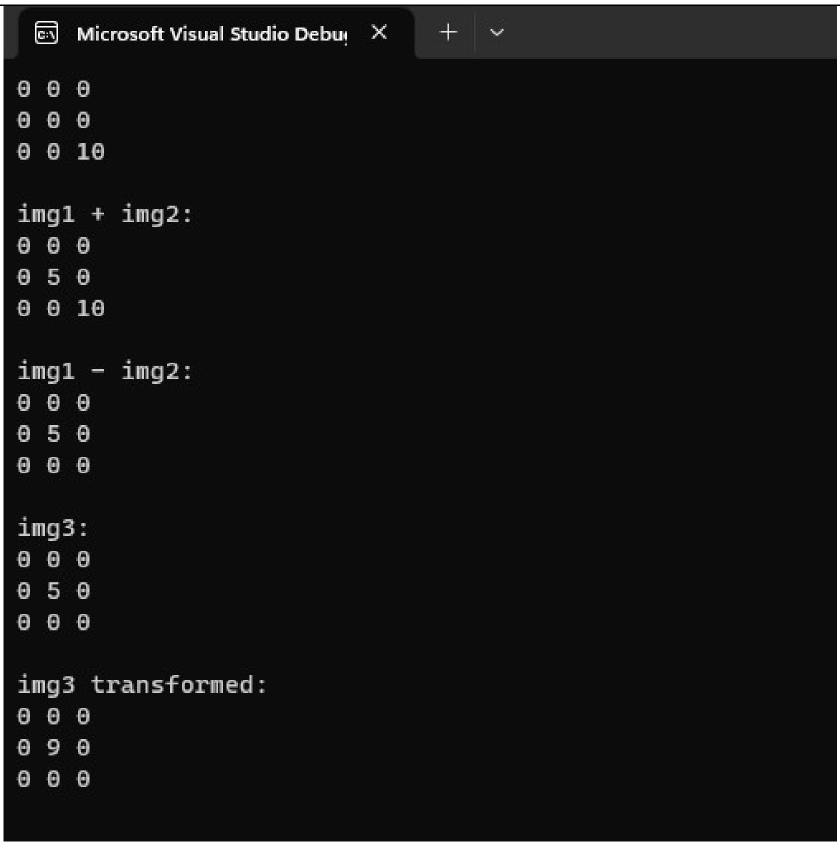
T(n) = n + n^2 = O(n^2)

QUESTION NO 4:
CODE:

```
#include <iostream>
```

```cpp
using namespace std;


class Image {
public:
    // Constructor
    Image(int h, int w);

    // Destructor
    ~Image();

    // Functions
    void MakeEmpty(int n);
    void StoreValue(int i, int j, int value);
    void Add(const Image& other);
    void Subtract(const Image& other);
    void Copy(const Image& other);
    void Transformation();
    void Print() const;

private:
    int** data;
    int height;
    int width;
};

Image::Image(int h, int w) {
    height = h;
    width = w;

    // Allocate memory for the 2D array
    data = new int* [height];
    for (int i = 0; i < height; i++) {
        data[i] = new int[width];
    }

    // Initialize all pixels to zero
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            data[i][j] = 0;
        }
    }
}

Image::~Image() {
    // Deallocate memory for the 2D array
    for (int i = 0; i < height; i++) {
        delete[] data[i];
    }
    delete[] data;
}

void Image::MakeEmpty(int n) {
    // Set the first n rows and columns to zero
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            data[i][j] = 0;
        }
    }
}

void Image::StoreValue(int i, int j, int value) {
    // Store the given value in the specified pixel
    data[i][j] = value;
}

void Image::Add(const Image& other) {
    // Add the pixel values of the other image to this one
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            data[i][j] += other.data[i][j];
        }
    }
}

void Image::Subtract(const Image& other) {
    // Subtract the pixel values of the other image from this one
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            data[i][j] -= other.data[i][j];
        }
    }
}

void Image::Copy(const Image& other) {
    // Copy the pixel values of the other image to this one
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            data[i][j] = other.data[i][j];
        }
    }
}

void Image::Transformation() {
    // Find the mean of the whole image
    int sum = 0;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            sum += data[i][j];
        }
    }
    float mean = (float)sum / (height * width);

    // Divide each pixel value by the mean value
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            data[i][j] = (int)((float)data[i][j] / mean);
        }
    }
}
void Image::Print() const {
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            cout << data[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}


int main() {
    int n = 3;
    Image img1(n, n);
    img1.MakeEmpty(n);
    img1.StoreValue(1, 1, 5);
    cout << "img1:" << std::endl;
    img1.Print();

    Image img2(n, n);
    img2.MakeEmpty(n);
    img2.StoreValue(2, 2, 10);
    cout << "img2:" << std::endl;
    img2.Print();

    // ADD img1 and img2
    img1.Add(img2);
    cout << "img1 + img2:" << std::endl;
    img1.Print();

    // Subtract img2 from img1
    img1.Subtract(img2);
    cout << "img1 - img2:" << std::endl;
    img1.Print();

    // Copy img1 to img3
    Image img3(n, n);
    img3.Copy(img1);
    cout << "img3:" << std::endl;
    img3.Print();

    // Transform img3
    img3.Transformation();
    cout << "img3 transformed:" << std::endl;
    img3.Print();

    return 0;
}
```

SCREENSHOT:



```
0 0 0
0 0 0
0 0 10

img1 + img2:
0 0 0
0 5 0
0 0 10

img1 - img2:
0 0 0
0 5 0
0 0 0

img3:
0 0 0
0 5 0
0 0 0

img3 transformed:
0 0 0
0 9 0
0 0 0
```