Assignment # 1
CS 2001 – Data Structures (CS)
Fall 2021

Purpose: The main purpose of the assignment is to assess your learning in terms of Array, ADT, and the asymptotic complexity analysis.

General Guidelines

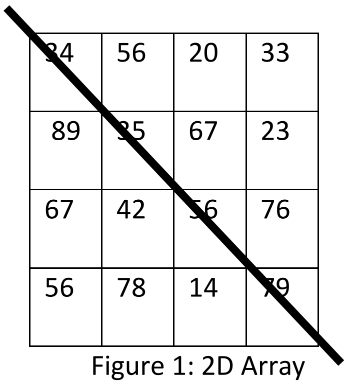
1. Peer plagiarism and the late submissions are strictly not allowed
2. Total Marks: 100

Submission Guidelines

3. Your assignment submission must be in hardcopy (i.e., handwritten, or printed form) and a scanned version should also be uploaded on the classroom within the given deadline.
4. Analytical and mathematical questions can be handwritten, while code questions should preferably be computer-typed [however, handwritten solutions are also allowed for this assignment only]
5. **Deadline (Monday, 27th September till 04:00 pm)**

Task1: Write a C++ program that creates an N x N 2-Dimensional array of integers using DMA. The program should, then, perform the following operations: - [20 marks]

a) Populate the whole 2D-array by the user inputs. Write upper-bound for time on this specific operation at the top of the code as a comment. [5 marks]
b) Write a C++ program to find out, separately, the sum of all elements i) above the diagonal, ii) below the diagonal, and iii) on the diagonal, as shown in the Figure 1. Also, write asymptotic upper-bound for all the three operations separately. [10 marks]
c) Write overall combined asymptotic time upper-bound for part (a) and part (b). [5 marks]



34	56	20	33
89	35	67	23
67	42	56	76
56	78	14	79

Figure 1: 2D Array

Task2: Determine the dominating terms from the given running time functions and express them in form of Big-O notation**. [10 marks]**

(e.g., In $T(n) = \frac{1}{2}\,n + 3$: the dominating term is "$\frac{1}{2}\,n$" and we can write it as $T(n) = O(\,n)$)

Expression	Dominant Term (s)	T(n) = O(?)
$\boldsymbol{T(n) = \frac{1}{2}\,n^2 + 1000.\ n + 3}$		
$\boldsymbol{T(n) = 0.001n^2 + 1000\,n + 3}$		
$\boldsymbol{T(n) = 0.001\log_{20} n + log_{10}\log_{10} n}$		
$\boldsymbol{T(n) = 0.1\,n^{1.2} + 100\,n\log_{10} n + 1000\,n}$		
$\boldsymbol{T(n) = 0.05\,\mathrm{n}\log_2 \mathrm{n}\ +\ \mathrm{n}(\log_2 \mathrm{n})^2}$		

Task3: Consider the following piece of codes. Extract a time function *T(N)* from the codes, and provide asymptotic time analysis for *T(N)*: - [40 marks]

Note: If loop have any conditional statement, then analyze the algorithm for both best and worst cases. In case, there is no conditional statement in the dominating piece of code, then, only upper bound is enough. Moreover, to prove that T(N) belongs to some asymptotic growth set, you will have to provide parameters required as per the definition (e.g., c and N$_0$ etc.).

a. Determine and explain the asymptotic running time complexity of the following piece of code.

```
int n;
cin >> n;
int count = 0;
for (int i = n; i > 0; i--) {
    for (int j = i; j >= 1; j--) {
        count++;
     }
}
```

b. Determine and explain the asymptotic running time complexity of the following piece of code.

```
int n;
cin >> n;
int count = 0;
for (int i = 1; i <= n; i++) {
    for (int j = n; j > 1; j /= 2) {
        count++;
    }
}
```

```
c. Determine and explain the asymptotic running time complexity of the following
   piece of code.

int n;
 cin >> n;
 int count = 0;
 for (int i = 1; i < n; i = i * 2) {

    for (int j = 1; j <= i; j++){
       count++;
    }

 }
```

//Hint: $2^0 + 2^1 + 2^2 + \ldots\ldots + 2^n = 2^{n+1} + 1$

```
d. Determine and explain the asymptotic running time complexity of the following
   piece of code. [Best-case + Worst-case]

int n;
 cin >> n;
 int count = 0;

 if (n % 2 == 0) {
     for (int i = 0; i < n; i++) {
         count++;
     }

 }
 else {
     for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
             count++;
         }
     }
 }
```

Task4: Write a C++ program to implement SquareMatrix ADT as a class. A square matrix can be represented by an NxN 2D-array. Include the following operations as member of the class: - [30 marks]

a) **Constructor,** which dynamically allocates an NxN 2D array as per user provided number.
b) **MakeEmpty(n),** which sets the first n rows and columns to zero
c) **StoreValue(i, j, value),** which stores value into the [i,j] position
d) **ADD,** to add two SquareMatrices and stores result in one of them.
e) **Subtract**, to subtract one matrix from another.
f) **Copy**, which copies one matrix into another
g) **Destructor,** which de-allocates all the dynamic allocations.

Assuming that N, which is size of one dimension, is the input size. Then, write asymptotic upper-bound for each of the above operations.

Note: Identification of proper specification for the Constructor, Add, Subtract, and Copy function is a part of the task.

_____Good Luck 😊_____