

# DSA ASSIGNMENT #1

## Question #1:

```
#include<iostream>
using namespace std;
class DA
{
public:
    int n;
    void arr()          //The Time complexity for populating the 2D array is  $O(N^2)$ 
    where N is the size of the array ..
    {
        cout << "Enter the size of Array : " << endl;
        cin >> n;
        int** ptr = new int*[n];          //declare 2D dynamic array of size n
        by n
        for (int i = 0; i < n; i++)
        {
            ptr[i] = new int[n];
        }
        cout << "PLZ !!Enter the Elements of the array : " << endl;
        for (int i = 0; i < n; i++) //taking input
        {
            for (int j = 0; j < n; j++)
            {
                cin >> ptr[i][j];
            }
        }
        for (int i = 0; i < n; i++)          // 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  $O(N^4)$ 
            for (int j = 0; j < n - 1; j++)
            {
                int minindex = j;
                for (int k = j + 1; k < n; k++)
                {
                    if (ptr[i][j] < ptr[i][minindex])
                    {
                        minindex = k;
                    }
                }
                if (minindex != j)
                {
                    int temp = ptr[i][j];
                    ptr[i][j] = ptr[i][minindex];
                    ptr[i][minindex] = temp;
                }
            }
        }
        cout << "Array sorted row-wise in ascending order:" << endl;
        for (int i = 0; i < n; i++)          //Displaying
```

```

    {
        for (int j = 0; j < n; j++)
        {
            cout << ptr[i][j] << " ";
        }
        cout << endl;
    }
    // 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  $O(N^4)$ .
    for (int j = 0; j < n; j++)
    {
        for (int i = 0; i < n - 1; i++)
        {
            int maxindex = i;
            for (int k = i + 1; k < n; k++)
            {
                if (ptr[k][j] > ptr[maxindex][j])
                {
                    maxindex = k;
                }
            }
            if (maxindex != i)
            {
                int temp = ptr[i][j];
                ptr[i][j] = ptr[maxindex][j];
                ptr[maxindex][j] = temp;
            }
        }
    }
    cout << "Array sorted column-wise in descending order:" << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << ptr[i][j] << " ";
        }
        cout << endl;
    }
}

```

// 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()
{
    DA obj;
    obj.arr();
    system("pause");
    return 0;
}

```

Output:

```
Enter the size of Array :
4
PLZ !!Enter the Elements of the array :
34
56
20
33
89
35
67
23
67
42
56
76
56
78
14
79
Array sorted row-wise in ascending order:
34 56 20 33
89 35 67 23
67 42 56 76
56 78 14 79
Array sorted column-wise in descending order:
89 78 67 79
67 56 56 76
56 42 20 33
34 35 14 23
Press any key to continue . . .
```

## **Question #2:**

EXPRESSION	DOMINANT TERM(s)	$T(n) = O(?)$
$T(n) = 3n^2 + 1000n + 360$	$3n^2$	$O(n^2)$
$T(n) = 120n + 4n^{1.1} + 3$	$4n^{1.1}$	$O(n^{1.1})$
$T(n) = 50 \log(n) + 0.3 \log(n)^2 + 25$	$0.3 \log(n)^2$	$O(\log(n)^2)$
$T(n) = 30n^2 + 45n! + 66$	$45n!$	$O(n!)$

$T(n) = n \cdot \log(n) + 15 \log(n)^2 + 4$	$n \cdot \log(n)$	$O(n \cdot \log(n))$
---------------------------------------------	-------------------	----------------------

### **Question #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:

$$T(n) = (n-1) * (n-1 + n-2 + \dots + 1) * c = (n-1) * \frac{n}{2} * c = O(n^2) \text{ where } c \text{ is a constant.}$$

- 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:

$$T(n) = 1 + 2 + 4 + \dots + \frac{n}{2} + n = 2n - 1 = O(n)$$

- 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

$$T(n) = n = O(\log n)$$

- 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 #4:**