***NOTE: Attempt all the questions and write to the point answers.***

**Q. 1 Why space and time complexity must be considered while writing algorithms?** 2

*ANSWER*

Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem. While analyzing an algorithm, we mostly consider time complexity and space complexity. Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Similarly, Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

Time and space complexity depends on lots of things like hardware, operating system, processors, etc. However, we don't consider any of these factors while analyzing the algorithm. We will only consider the execution time of an algorithm.

Lets start with a simple example. Suppose you are given an array AA and an integer xx and you have to find if xx exists in array AA.

Simple solution to this problem is traverse the whole array AA and check if the any element is equal to xx.

```
for i : 1 to length of A
   if A[i] is equal to x
      return TRUE
return FALSE
```

Each of the operation in computer take approximately constant time. Let each operation takes cc time. The number of lines of code executed is actually depends on the value of xx. During analyses of algorithm, mostly we will consider worst case scenario, i.e., when xx is not present in the array AA. In the worst case, the **if** condition will run NN times where NN is the length of the array AA. So in the worst case, total execution time will be (N∗c+c)(N∗c+c). N∗cN∗c for the **if** condition and cc for the **return** statement ( ignoring some operations like assignment of ii ).

As we can see that the total time depends on the length of the array AA. If the length of the array will increase the time of execution will also increase.

**Order of growth** is how the time of execution depends on the length of the input. In the above example, we can clearly see that the time of execution is linearly depends on the length of the array. Order of growth will help us to compute the running time with ease. We will ignore the lower order terms, since the lower order terms are relatively insignificant for large input. We use different notation to describe limiting behavior of a function.

**O-notation:**

To denote asymptotic upper bound, we use OO-notation. For a given function g(n)g(n), we denote by O(g(n))O(g(n)) (pronounced "big-oh of g of n") the set of functions: O(g(n))=O(g(n))= { f(n)f(n) : there exist positive constants cc and n0n0 such that 0≤f(n)≤c∗g(n)0≤f(n)≤c∗g(n) for all n≥n0n≥n0 }

**Ω-notation:**

To denote asymptotic lower bound, we use ΩΩ-notation. For a given function g(n)g(n), we denote by Ω(g(n))Ω(g(n)) (pronounced "big-omega of g of n") the set of functions: Ω(g(n))=Ω(g(n))= { f(n)f(n) : there exist positive constants cc and n0n0 such that 0≤c∗g(n)≤f(n)0≤c∗g(n)≤f(n) for all n≥n0n≥n0 }

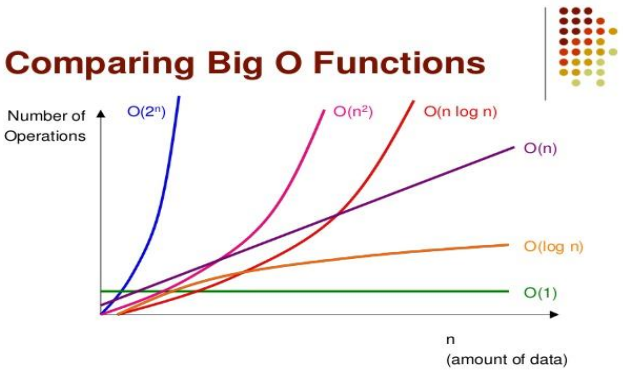
**Θ-notation:**

To denote asymptotic tight bound, we use ΘΘ-notation. For a given function g(n)g(n), we denote by Θ(g(n))Θ(g(n)) (pronounced "big-theta of g of n") the set of functions: Θ(g(n))=Θ(g(n))= { f(n)f(n) : there exist positive constants c1,c2c1,c2 and n0n0 such that 0≤c1∗g(n)≤f(n)≤c2∗g(n)0≤c1∗g(n)≤f(n)≤c2∗g(n) for all n>n0n>n0 }

**Q. 2 Arrange following complexity in their increasing order of growth: $N^2$ , NlogN, N, $2^N$, $N^3$, log N** 2

*ANSWER*

$2^N > N^3 > N^2 > NlogN > N > log\ N$



**Q. 3** **Write algorithm/ program of Binary Search.** 2

*ANSWER*

```
int binarySearch(int low,int high,int key)
{
  while(low<=high)
  {
   int mid=(low+high)/2;
   if(a[mid]<key)
   {
      low=mid+1;
   }
   else if(a[mid]>key)
   {
      high=mid-1;
   }
   else
   {
      return mid;
   }
  }
  return -1;            //key not found
}
```

**Q. 4** **Write a C program to perform 2-D Matrix multiplications and find out its complexity?** 2

```
#include <stdio.h>
#define N 4
void multiply(int mat1[][N], int mat2[][N], int res[][N])
{
   int i, j, k;
   for (i = 0; i < N; i++)
   {
      for (j = 0; j < N; j++)
      {
         res[i][j] = 0;
         for (k = 0; k < N; k++)
            res[i][j] += mat1[i][k]*mat2[k][j];
      }
   }
}
 int main()
{
   int mat1[N][N] = { {1, 1, 1, 1},
                      {2, 2, 2, 2},
                      {3, 3, 3, 3},
                      {4, 4, 4, 4}};

   int mat2[N][N] = { {1, 1, 1, 1},
                      {2, 2, 2, 2},
                      {3, 3, 3, 3},
                      {4, 4, 4, 4}};

   int res[N][N]; // To store result
   int i, j;
   multiply(mat1, mat2, res);

   printf("Result matrix is \n");
   for (i = 0; i < N; i++)
   {
      for (j = 0; j < N; j++)
        printf("%d ", res[i][j]);
      printf("\n");
```

```
        }
        return 0;
}
```

**Q. 5** **An array X [-15…….10, 15……40] requires one byte of storage. If beginning location is 1500 determine the location of X [15][20].** 2

Row Major System:

The address of a location in Row Major System is calculated using the following formula:

Address of A [ I ][ J ] = B + W * [ N * ( I – Lr ) + ( J – Lc ) ]

Column Major System:

The address of a location in Column Major System is calculated using the following formula:

Address of A [ I ][ J ] Column Major Wise = B + W * [( I – Lr ) + M * ( J – Lc )]

Where,
B = Base address
I = Row subscript of element whose address is to be found
J = Column subscript of element whose address is to be found
W = Storage Size of one element stored in the array (in byte)
Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)
Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)
M = Number of row of the given matrix
N = Number of column of the given matrix
As you see here the number of rows and columns are not given in the question. So they are calculated as:

Number or rows say M = (Ur – Lr) + 1 = [10 – (- 15)] +1 = 26
Number or columns say N = (Uc – Lc) + 1 = [40 – 15)] +1 = 26

(i) Column Major Wise Calculation of above equation

The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, M = 26

Address of A [ I ][ J ] = B + W * [ ( I – Lr ) + M * ( J – Lc ) ]

= 1500 + 1 * [(15 – (-15)) + 26 * (20 – 15)] = 1500 + 1 * [30 + 26 * 5] = 1500 + 1 * [160] = 1660 [Ans]

(ii) Row Major Wise Calculation of above equation

The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, N = 26

Address of A [ I ][ J ] = B + W * [ N * ( I – Lr ) + ( J – Lc ) ]

= 1500 + 1* [26 * (15 – (-15))) + (20 – 15)] = 1500 + 1 * [26 * 30 + 5] = 1500 + 1 * [780 + 5] = 1500 + 785
= 2285 [Ans]