Time Complexity calculations

28 October 2023 18:11

1) Program to calculate factorial of a given number.

```
res = 1, n = 9
                for(i=1; i<=n; i++)
                    res = res * i;
    Print(res);
```

Time Complexity = O(n)

We traverse through the loop n times. As n increases time required
Also increases.
As, Time is proportional to n. Hence, O(n) -> Orderof(n)

---

2) Print 2-D Matrix of size n*n.

n = 3
loop iterates = 9

int arr[3][3] = {11, 22, 33, 44, 55, 66, 77, 88, 99};

```
for( i = 0; i<n; i++)
{
    for(j=0; j<n; j++)
    {
        Print(arr[i][j];
    }
}
```

i	j
0	0, 1, 2, 3
1	0, 1, 2, 3
2	0, 1, 2, 3

11 22 33
44 55 66

$O(n^2)$

Here, there is loop inside loop. For every iteration of outer loop, the innermost loop goes through all the iterations.
So, no. of iterations for inner loop are n*n
As, Time $\alpha\ n^2$. Hence, O(n^2)

---

3) Print the given number in binary format.
Algorithm : Divide the given number by 2 and collect the remainder.

```
While(n > 0){
    Print(n%2)
    N = n/2
    }
```

10 -> 4.
1000 -> 10

For 10 there are only 4 iterations
For 1000 there are 10 iterations

10/2 = 5 -> remainder 0
5/2 = 2 -> remainder 1
2/2 = 1 -> remainder 0
1/2 = 0 -> remainder 1

Going in reverse order the binary of 10 is 1010

Here, Each time we are dividing the number in parts. So we are performing partitioning.
Whenever there is partitioning, the calculation is
$2^i = n,$ *where i is the number of iterations*
2^4
$= 16$ *which is close to* 10, *hence take* 4 *iterations*

10 -> 4 times.

$2^i = n \rightarrow 2^4 = 16$

$2^{ite} = n$

$2^4 = 10$

Take log on both sides
2^itr = n
Log 2^itr = log n

$\log 2^{itr} = \log n$

Itrs = log n / log 2
Time proportional to
Log n / log 2
1/log 2 is constant in theory of proportionality
Hence, 1/log 2 is discarded.
Hence , time = log n

$ite = \frac{\log n}{\log 2}$

$itr = \log n$

as $\frac{1}{\log 2}$ is constant

Hence, 1/log 2 is discarded.
Hence , time = log n

$n = \log n$

As $\frac{1}{\log 2}$ is constant

4) Print table of given number

```
n = 50;
For(I = 1;i<=10;i++)
{
    Print(num * i);
}
```

n → 5
n → 15
n → 50
n → 75

} loop iterates 10 times only.

Here the loop iterates only 10 times irrespective of the value of n.
Hence we can say, the loop iterations are constant.
So the time complexity here is O(1).