Time Complexity

28 October 2023 18:11

Asymptotic Analysis.

1) Program to calculate factorial of a given number.

```
res = 1, n = 5
for(i=1; i<=n; i++)
    res = res * i
Print(res)
```

$T \propto n$.

$O(n) \rightarrow$ Order of (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. 3*3

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

$i = 0 \rightarrow j \rightarrow 0$ to $n-1$
$i = 1 \rightarrow j \rightarrow 0$ to $n-1$
$i = 2 \rightarrow j \rightarrow 0$ to $n-1$

$T \propto n^2 \rightarrow O(n^2)$.

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 α 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
    }
```

10/2	=	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

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

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

Take log on both sides
2^itr = 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 , time = log n

Print table of given number

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

iterations = constant = $T = K$

$T \propto O(1)$

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^3)$