Name - Riya Chaudhary
Section - I
Roll No - 7

Date
Page No. 1

Tutorial - 1

Asymptotic notations used to analyse an algorithm when input is large.

1. Big O Notation - It represents the upper bound of the running time of an algorithm. It gives the worst case complexity of an algorithm.

Ex $f(n) = O(g(n))$ if $0 \leq f(n) \leq g(g(n))$ if $n \geq n_0$ for $g(n)$

2. Small o notation - It is denoted by o. It is used to describe an upper bound that cannot be tight.

Ex - $f(n) = o(g(n))$ if $f(n) < g(n) \forall n > n_0$ for $\forall c > 0$.

3. Theta Notation - It represents the upper and lower bound of running time of an algorithm. It is used for analyzing the average-case complexity of an algorithm.

Ex - $f(n) = \theta(g(n))$ if $c_1(g(n)) \leq f(n) \leq c_2(g(n))$ $\forall n \geq \max(n_1, n_2)$ and some constant c_1 & $c_2 > 0$

4. Big Omega Notation - It represents the lower bound of the running time of an algorithm. It provides best case complexity.

Ex - $f(n) = \Omega(g(n))$ if $f(n) \geq (g(n)) \geq 0 \forall n \geq n_0$ & $c > 0$.

5. Small omega Notation - It is used to describe a loose bound of $f(n)$ and it is denoted by ω.
Ex- $f(n) = \omega(g(n))$ if $f(n) > g(n)) \forall n > n_0 \& c > 0$

2. for (i=1 to n) (i=i*2;)

for (i = 1; i <= n; i *= 2)

$1, 2, 4, \ldots n$

$n = ar^{k-1}$

$n = 1(2)^{k-1}$

$2^k = 2n$

$\log_2 2^k = \log_2 2n$

$K = \log_2 2 + \log_2 n$

$K = 1 + \log_2 n$

$\boxed{K = O(\log n)}$

3. $T(n) = \{ 3T(n-1)$ if $n > 0$, otherwise $1 \}$

$T(n) = 3T(n-1)$

$= 3(3T(n-1)-1)$

$= 3^2 T(n-2)$

$= 3^2(3T(n-2)-1)$

$= 3^3 T(n-3)$

$= 3^n T(n-n)$

$= 3^n$

1. $T(n) = \{ 2T(n-1) - 1 \text{ if } n > 0, \text{ otherwise } 1 \}$

$T(n) = 2T(n-1) - 1$

$= 2(2T(n-1-1) - 1)$

$= 2^2 T(n-2) - 2 - 1$

$= 2^3 T(n-3) - 2^2 - 2^1 - 2^0$

$= 2^n T(n-n) - (2^n - 1)$

$= 2^n - 2^n + 1 = 1$

$T = O(1)$

5.
```
int i=1, s=1;
while(s<=n)
{ i++; s=s+i;
  print("#");
}
```

$1 \le n$

$S = 1+2$, $S = 3+3$, $S = 6+4$

$S = 3$, $S = 6$, $S = 10$

$1, 3, 6, 10, \ldots$

$1 + 2 + 3 + \cdots K$

$\frac{K(K+1)}{2} = n$

$K^2 + K - 2n = 0$

$K = \frac{-1 \pm \sqrt{1+8n}}{2} \Rightarrow K = \frac{1}{2}\sqrt{1+8n}$

$K = \sqrt{n}$

$K = O(\sqrt{n})$

$\boxed{T = O(\sqrt{n})}$

6. void function (int n)
{ int i, count = 0.
for (i = 1; i * i <= n; i++)
count++;
}

i = 1 $1 \leq n$
i = 2 $4 \leq n$
i = 3 $9 \leq n$

$K^2 <= n$
$K <= \sqrt{n}$
$T = O(\sqrt{n})$

7. void function (int n)
{ int i, j, k, count = 0;
for (i = n/2; i <= n; i++)
for (j = 1; j <= n; j = j * 2)
for (k = 1; k <= n; k = k * 2)
count++

}

$\frac{1}{2}, \frac{2}{2}, \frac{3}{2} \cdots \frac{n}{}$

$n = \frac{1}{2} + (K-1) \times \frac{1}{2}$

$n = \frac{1}{2} + \frac{K}{2} - \frac{1}{2}$

$K = n$
$T = O(n)$

$1, 2, 4, 8 \cdots n$
$n = 1\,(2)^{K-1}$
$2^K = 2n$

$K = 1 + \log n$

$T = O(\log n)$

1, 2, 4, 8, -- n

$n = 1\,(2)^{K-1}$

$2^K = 2n$

$K = 1 + \log n$

$T = O(\log n)$

8.

```
function(int n)
{ if (n==1) return;
    for (i=1 to n)
     { for (j=1 to n)
        { printf(" * ");
        }
     }
  function(n-3);
}
```

1, 2, 3, -- n

$n = 1 + (K-1) \times 1$

$n = K$ $\quad T = O(n)$

1, 2, 3, -- n

$n = 1 + (K-1)n$

$n = K$ $\quad T = O(n)$

$T = O(n^2)$

9.
```
void function (int n)
{ for (i=1 to n)
  { for(j = 1; j<=n ; j = j+i)
    printf("*");}
}
```

1, 2, 3, -- n

$n = 1 + (k-1)1$

$n = k$

$T = O(n)$

1, 2, 3, -- n

2, 3, -- n

$T = O(n)$

$T = O(n^2)$

10.

$n^k \; c^n$

$k \geq 1 \; c > 1$

$n^k = O(c^n)$

$n \geq n_0$

$c > 0$