NAME - Abhishek Kumar Section - DS1 Roll.no - 5

Q1 Asymptotic notations are used to represent the complexities of algorithms for asymptotic analysis.

→ These notations are mathematical tools to rep complexities

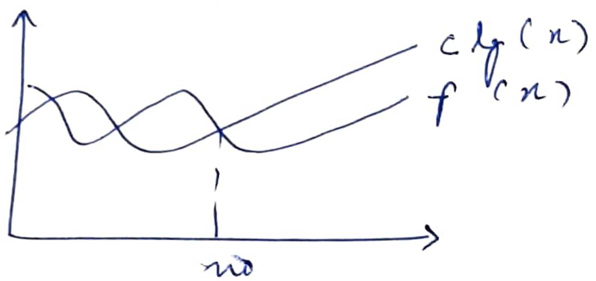
Big Oh notation:-

Gives upper bound for a f^n f(n) within a constant factor.

$f(x) = O(g(x))$

if $f(n) \leq c g(n)$

for $c > 0$ & $n \geq n_0$.



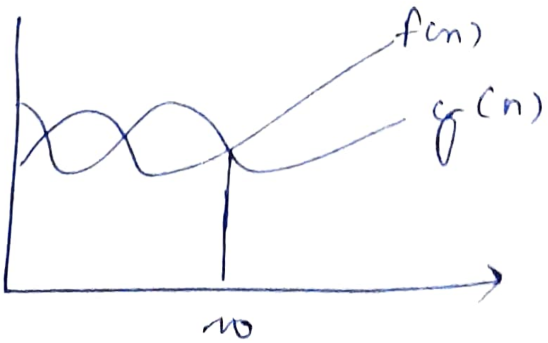
Big Omega Notation.

Gives lower bound for a f^n f(x) within a constant factor

$f(n) = \Omega(g(x))$

if $f(x) \geq c g(x)$

f(n) for $c > 0$ & $n \geq n_0$



Big Theta Notation

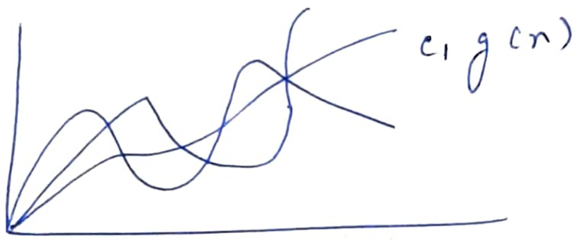
Gives Bound for a fn f(n) with const factors

$f(n) = \Theta(g(n))$

if $c_1 g(n) \le f(n) \le c_2 g(n)$

$c_1 > c_2 > 0$

$\forall n \geq n_0$



2. TC for →

for (i = 1 to n)

i = i * 2

i = 1   2   4   8   - -   n

2^0 2^1 2^2 2^3 . . . 2^n

$GP = ar^{k-1}$

$n = 1 \cdot 2^{k-1}$

$n = \frac{2^k}{2}$

$2n = 2^k$

$\log 2n = k \log 2$

$\log 2 + \log n = k \log 2$

$\boxed{\log n = k}$

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

3. $T(n) = 3T(n-1)$, $n > 0$, otherwise

$T(0) = 1$

$T(1) = 3T(0)$

$= 3$

$T(2) = 3(T)(1))$

$= 9 = 3^2$

$T(n) = 3^n$

$= O(3^n)$

4. $T(n) = 2T(n-1) - 1$ — ① , $n > 0$, otherwise 1

Let $n = n-1$

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

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

put $T(n-1)$ in ①

$T(n) = T(n-2) - 3$ — ②

put $n = n-2$

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

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

Out in ②

$T(n) = 4(2T(n-3) - 1) - 1$

$= 8T(n-3) - 4 - 1$

$= 8T(n-3) - 5 = 2^k T(n-k) \cdot -1$

$$\boxed{\begin{matrix}(n-k) = 1 \\ k = (n-1)\end{matrix}}$$

$T(n) = 2^{n-1} T(n-n+1) - 5$

$= 2^{n-1} T(1) - 5$

$= \frac{2^n}{2} = 2^n = O(2^n)$

5. while (s <= n)

```
{ i++;
  s = s+i;
  printf("#");
}
```

$i = 1 = i++, i = 2$

$S = 3$

$i = 3$

$S = 6$

$i = 4$

$S = 10$

$i = 5$

$S = 15$

$i = 2 \quad 3 \quad 4 \quad 5 \longrightarrow$

$S+i+2 \quad S+1+2+3 \quad S+1+2+3+4 \qquad S+1+2+3+4+5$

$S = S+1+2+3+4 - - k$

$S(k) = K(K+1)/2 \leq n$

$K^2 + \frac{K}{2} \leq n$

$K^2 \leq n$

$K \leq \sqrt{n}$

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

6. void fn (int n)

```
{ int i, count = 0;
  for (i=1, i*i<=n, i++)
  {
    count++;
  }
}
```

$i = 1 \quad 2 \quad 3 \quad 4 \ldots$

$i^2 = 1 \quad 4 \quad 9 \quad 16 \ldots k^2$

$K^2 \leq n$

$K \leq \sqrt{n}$

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

7. void fn (int n)

```
{ int i, j, k, count = 0;
 for (i = n/2 , i <= n ; i++)      -> T(n/2)
 {
  for (j = 1, j < n , j = j * 2)     -> log(n)
   for (k = 1 ; k <= n ; k = k * 2)   -> log (n)
    count ++;
   }
 }
}
```

$T(n) = T(n/2) * \log(n) * \log(n)$

$= n/2 * \log n^2$

$= O(n \log^2 n)$