__/__/__

Tutorial - 1

(1) Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value on a limiting value.

Types of asymptotic notations:

(a) Big-O-Notation - it represents the upperbound of the running time of an algorithm. It gives the worst-case complexity of an algorithm.

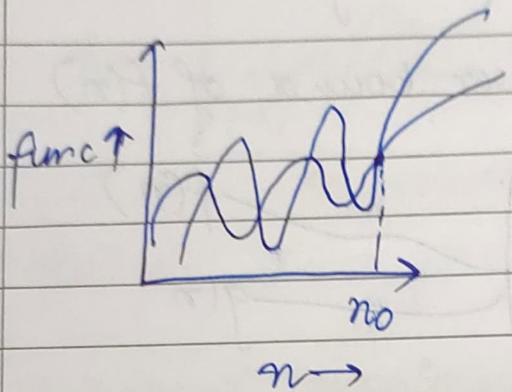
(b) Omega Notation ($\Omega$-notation) - represents the lower bound of the running time of an algorithm. Thus, gives the best case complexity of an algorithm.

(c) Theta-Notation ($\Theta$-Notation): It represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.

(4) Small oh ($o$) - it gives us the upper bound

$$f(n) = o(g(n))$$

$$f(n) < c\,g(n) \quad \forall n > n_0 \ \& \ \forall c > 0$$




$n = o(n^2)$

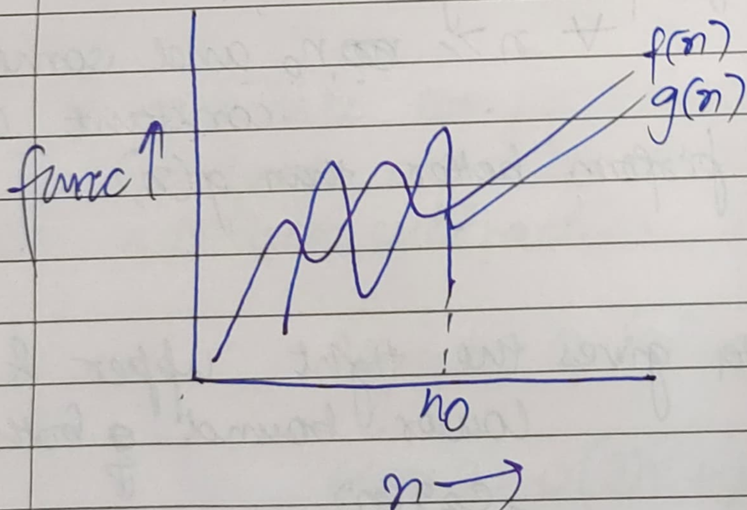
$n < 1n^2$

$2n^2$

$0.5n^2$

$\boxed{n < 0.001n^2}$ $n_0$

(5) Small-omega ($\omega$):-




lower bound

$f(n) = \omega\, g(n)$

$f(n) > c \cdot g(n)$

$\forall n > n_0 \ \& \ \forall c > 0$

$n^2 = \omega(n)$

(2) for ($i=1$ to $n$) { $i = i*2;$ }

$i = 1, 2, 4, 8, 16, 32, \ldots, n$

$= 2^0, 2^1, 2^2, \ldots \ldots, 2^k$

$2^k = n$

$k \log_2 2 = \log_2 n \quad \Rightarrow k = \log n$

$T(n) = O(\log_2 n)$

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

$T(n) = 3T(n-1)$ — ①
$T(n) = aT(n-b) + f(n)$ — ②

comparing ① and ②

$a=3$
$b=1$

since $f(n) = 0$
$k=0$

$T(n) = O\left(n^k \sum^{i=n/b} a^i\right)$

$T(n) = O\left(n^0 \sum^{i=n/b} a^i\right)$

$T(n) = O\left(3^{n/1}\right)$

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

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

$T(n) = 2T(n-1)-1$ — ①
$T(n) = aT(n-b)+f(n)$ — ②

comparing ① and ②:
$a=2$, $b=1$, $f(n)=-1$
$k=-1$

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

Similarly, after k steps:

$= 2^kT(n-k) - 2^{k-1} - 2^{k-2} - \dots - 2^2 - 2^1 - 2^0$

considering $T(1)=1$,
$n-k=1$
$k=n-1$

substituting $k=n-1$
$T(n) = 2^{n-1}T(1) - [2^0+2^1+2^2+\dots+2^{n-3}+2^{n-2}]$

$= 2^{n-1}\times 1 - [2^{n-1}-1]$

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

$T(n) = 1$

time complexity $\Rightarrow O(1)$ Ans

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

loop will work for: $1+2+3+\dots+x <= n$
$\Rightarrow [x*(x+1)]/2 <= n$
$= O(x^2) <= n$
$x = O(\sqrt{n})$ Ans

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

1, 2, 4, 16, 256, --- n

$2^0, 2^1, 2^2, 2^4, 2^{16}, --- 2^{2^k}$

$2^{2^0}, 2^{2^1}, 2^{2^2}, 2^{2^4}, 2^{2^8}, --- 2^{2^k}$

$2^{2^k} = n$

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

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

~~T(n) = O(log n)~~

$k \log_2 2 = \log_2(\log_2 n)$

$k = \log(\log n)$

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

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

$\frac{n}{2} * \log n * \log n$

$\Rightarrow O\left(\frac{n}{2}(\log n)^2\right)$

(8)
```
function (int n)                  ← T(n)
{
    if (n==1)                     // T(1)
        return;
    for (i=1 to n)                // T(n)   } // n²
    {
        for (j=1 to n) {          // T(n)   }
            printf("*");
        }
    }
    function (n-3);               // T(n-3)
}
```

n, n-3, n-6, n-9, .... 1

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

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + ... + 1$

$T(n) = k n^2$

$T(n) \simeq (k-1)/3 * n^2$, so $T(n) = O(n^3)$ Ans

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

$\left(\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \frac{n}{5} + ..... \frac{n}{n}\right)$

$n * \left(1 + \frac{1}{2} + \frac{1}{3} + .... + \frac{1}{n}\right)$

$\Rightarrow O(n \log n)$

(10) n^k and c^n

$k >= 1$ $\quad$ $c > 1$

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

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

$n^k \leq a(c^n)$

$\forall\ n > n_0$ & constant, $a > 0$

for $n_0 = 1$; $c = 2$

$1^k \leq a^2$

$n_0 = 1$ & $c = 2$ Ans