Asymtotic Notation - These are the mathematical notation used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

- Θ notation - Theta notation bounds a function from above and below, so it defines exact asymtotic behaviours.
  eg. $3n^3 + 6n^2 + 6000 = \Theta n^3$

- Big O notation - The big O notation defines an upper bound of an algorithm it bounds only from above.
  eg Two loops with iteration over n and m terms respectively will have time complexity as $O(n+m)$

Ω notation - Just as Big O notation provides upper bound on a function, Ω (omega) notation provides an asymtotic lower bound.

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

Time complexity for a loop means no. of times the loop has run for ~~loop~~ following value of i

$1, 2^1, 2^2, 2^3, \ldots 2^k$

$[1, 2, 3, \ldots k]$ times

as per program

$2^k = n$

$k \log 2 = \log n$

$k = \log_2 n$

$T(n) = \log_2 n$

$O(\log_2 n)$

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

$T(1) = 3$

$T(n) = 3T(n-1)$ — (1)

Let $n = n-1$

$T(n-1) = 3T(n-2)$ — (2)

From (1) & (2)

$T(n) = 3\{3T(n-2)\} = 3^2(T(n-2))$

$\vdots$

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

$= 3^n$

$\therefore O(3^n)$

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

$T(n) = 2T(n-1) - 1$ — (1)

let $n = n-1$

$T(n-1) = 2T(n-2) - 1$ — (2)

From (1) & (2)

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

$T(n) = 2^2 T(n-2) - 2 - 1$ — (3)

$\vdots$

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

let ~~$k=n$~~ ~~$n-k=1$~~ let $k=n$

~~$T(n) = 2^k T(1) - 2^k + 1$~~ $T(n) = 2^n T(0) - 2^n + 1$

~~$= 2^k c - 2^k + 1$~~ $= 2^n - 2^n + 1$

~~$= 2^k(c-1) + 1$~~ $= 1$

$\therefore$ ~~$O(2^n)$~~ $\Rightarrow O(1)$

Time complexity

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

i	s
1	1 = 1
2	1+2 = 3
3	1+2+3 = 6
⋮	⋮
	n

$T(K) = 1+2+3+ \dots K$

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

for K iteration

$1+2+3+\dots K <= n$

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

$\Rightarrow \frac{K^2+K}{2} <= n$

$\Rightarrow O(K)^2 <= n$

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

$\Rightarrow O(\sqrt{n})$

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

$i * i <= n$

$i <= \sqrt{n}$

$i = 1, 2, 3 \ldots \sqrt{n}$

$\sum_{i=1}^{n} 1 + 2 + 3 + \ldots + \sqrt{n}$

$\Rightarrow T(n) = \frac{\sqrt{n}(\sqrt{n}+1)}{2}$

$T(n) = \frac{n + \sqrt{n}}{2}$

$\Rightarrow O(n)$

```
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++;
}
```

for $k = k * 2$

$k = 1, 2, 4, 8 \ldots n$

$n = \frac{a(r^k - 1)}{r-1}$ $\quad [a = 1 \; r = 2]$

$n = 2^k - 1$

$\log n = k$

⇒

i	j	k
1	$\log n$	$\log n \cdot \log n$
2	$\log n$	$\log n \cdot \log n$
3	⋮	⋮
⋮		
n	$\log n$	$\log n \cdot \log n$

$\therefore \quad O(n \log^2 n)$

```
function (int n)
{  if (n==1) return;
     for (i=1 to n)                    // O(n^2)
     {   for (j=1 to n)                // O(n)
         {   printf("*");
         }
     }
     function (n-3);                   // T(n-3)
}
```

$\therefore \quad T(n) = T(n-3) + n^2 \quad \text{—— (1)}$

let $n = n-3$

$T(n-3) = T(n-6) + n^2 \quad \text{— (2)}$

$T(n) = T(n-6) + n^2 + n^2 \quad \text{— (3)}$

⋮

$T(n) = T(n-3K) + K n^2 \quad —$

let $n - 3k = 1$

$T(n) = T(1) + K n^2$

$\therefore \quad O(n^2)$

For the function, n^k and c^n, what relationship b/w those functions?

Assume that $k >= 1$ and $c > 1$ are the value of c and n_0 for which as given n^k and c^n.

relation bet^n n^k & c^n

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

as $n^k \leq a c^n$

$\forall\ n \geq n_0$ and so

for $n_0 = 1$

$c = 2$

$$\Rightarrow 1^k \leq a\, 2^1$$

$\therefore$ $n_0 = 1$ $\quad c = 2$