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

→ $T(n) = 3T(n-1)$ — ①

put $n = n-1$

$T(n-1) = 3T(n-2)$ — ②

put in eq. ①

$T(n) = 3 \times 3T(n-2)$

$T(n) = 9T(n-2)$ — ③

put $n = n-2$ in eq -①

$T(n-2) = 3T(n-3)$ — ④

put in eq ③

$T(n) = 27T(n-3)$ — ⑤

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

$T(0) = 1$

put $n-k=0$

$n = k$

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

$T(n) = 3^n$

$\boxed{T(n) = O(3^n)}$

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

→ $T(n) = 2T(n-1) - 1$ — ①

put $n = n-1$

$T(n-1) = 2T(n-2) - 1$ — ②

put in eq ①

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

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

Scanned by TapScanner

$T(n) = 2^2\, T(n-2) - 3$ — ③

put $n = n-2$

$T(n-2) = 2T(n-3) - 1$ — ④

put in eq ③

$T(n) = 4\left[2T(n-3) - 1\right] - 3$

$T(n) = 8T(n-3) - 4 - 2 - 2^0$ — ⑤

$T(n) = 2^k\, T(n-k) - 2^{k-1} - 2^{k-2} \ldots 2^0$

put $n-k = 0$

$n = k$

$T(n) = 2^n\, T(n-n) - 2^{n-1} - 2^{n-2} \ldots 2^0$

$= 2^n\, T(0) - 2^{n-1} - 2^{n-2} \ldots 2^0$

$= 2^n - \left[1 + 2^1 + 2^2 + \ldots + 2^{n-1}\right]$

$T(n) = 2^n - \left(\dfrac{1 \times (2^n - 1)}{2-1}\right)$

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

$= 1$

$T(n) = O(1)$

6- Time coplexity of - void function(int n)

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

→ for $n=1$, $i=1$ time
for $n=2$, $i=1$ time
⋮

for $n=4$ $i=2$ times
⋮

for $n=3$ $i=3$ times
⋮

for $n=n$ $=\sqrt{n}$ times

$\sum_{i=1}^{n} 1+\cancel{1}+1+2+\dots+\sqrt{n}$ times

$= O(\sqrt{n})$

$T(n) = O(\sqrt{n})$ or $O(n^{1/2})$

7– Time complexity of:

```
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)
      count++;
}
```

→ for $n=2$, $i=2$ times
⋮

for $n=16$, $i=3$ times
⋮

for outer loop, $(n/2+1)$ times

for both inner loops = $(\log_2 n)$ times

$$T(n) = O(i * j * k)$$

$$= O\left(\left(\frac{n}{2}+1\right) * (\log_2 n) * (\log_2 n)\right)$$

$$T(n) = O\left[n(\log_2 n)^2\right]$$

2- Times complexity of :

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

→ for $n=1$, i will be from 1 to 1 ⇒ 1 times

for $n=2$, i will be 3 times.

upto n time → $(\log_2 n+1)$ times

$$\Rightarrow O(\log_2 n+1)$$

1- What is Asymptotic notations. Define different Asymptotic notation with examples:

→ They help us to find the complexity of an algorithm when input is very large.

1- **Big O(O):**

$f(n) = O(g(n))$

$f(n) = O(g(n))$

iff

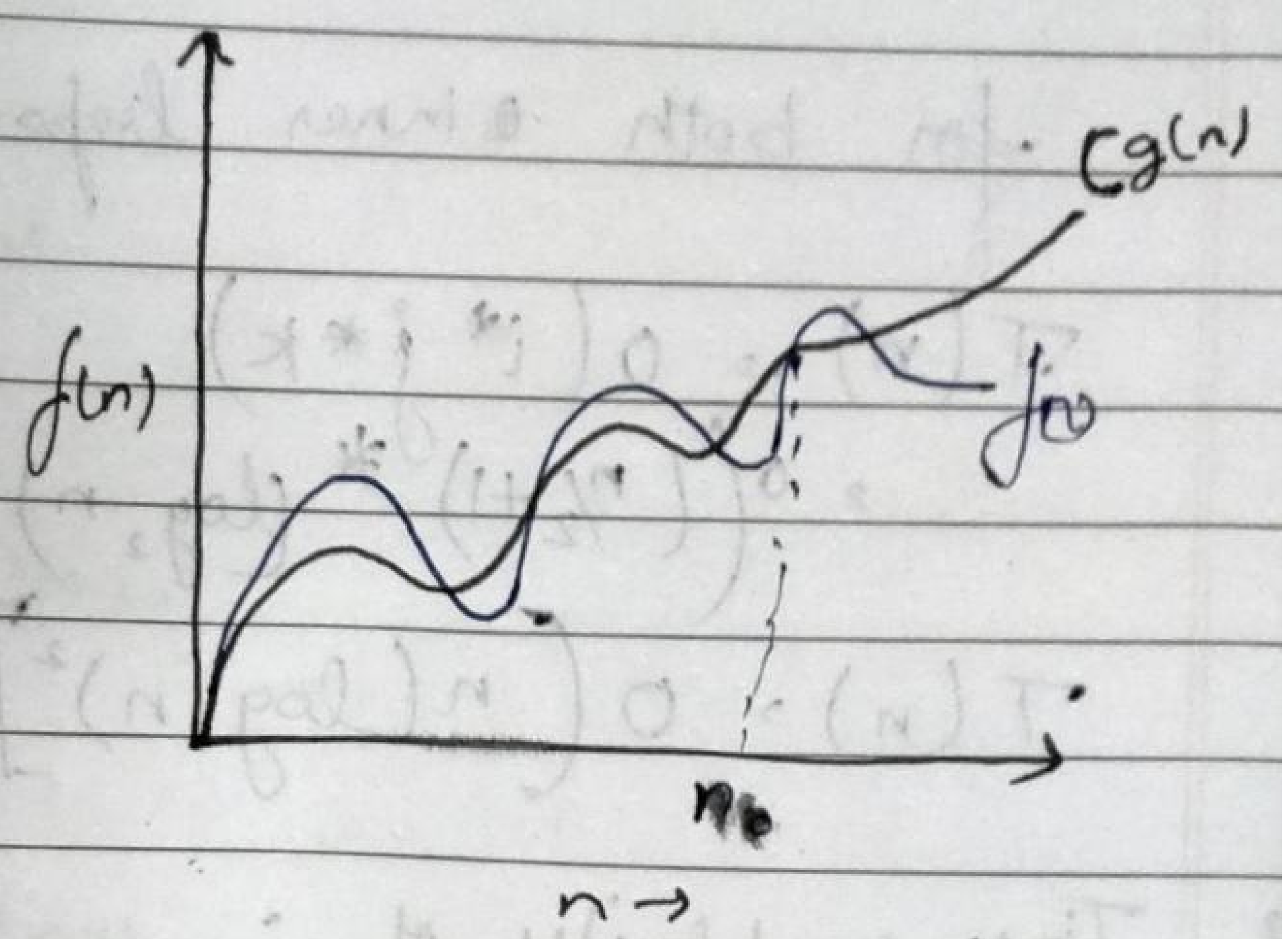
$f(n) \le g(n)$

$\forall$

$n \ge n_0$

for some constant, $C > 0$

$g(n)$ is 'tight' upper bond of $f(n)$.



2. **Big Omeg(Ω):** $f(n) = \Omega(g(n))$

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

iff

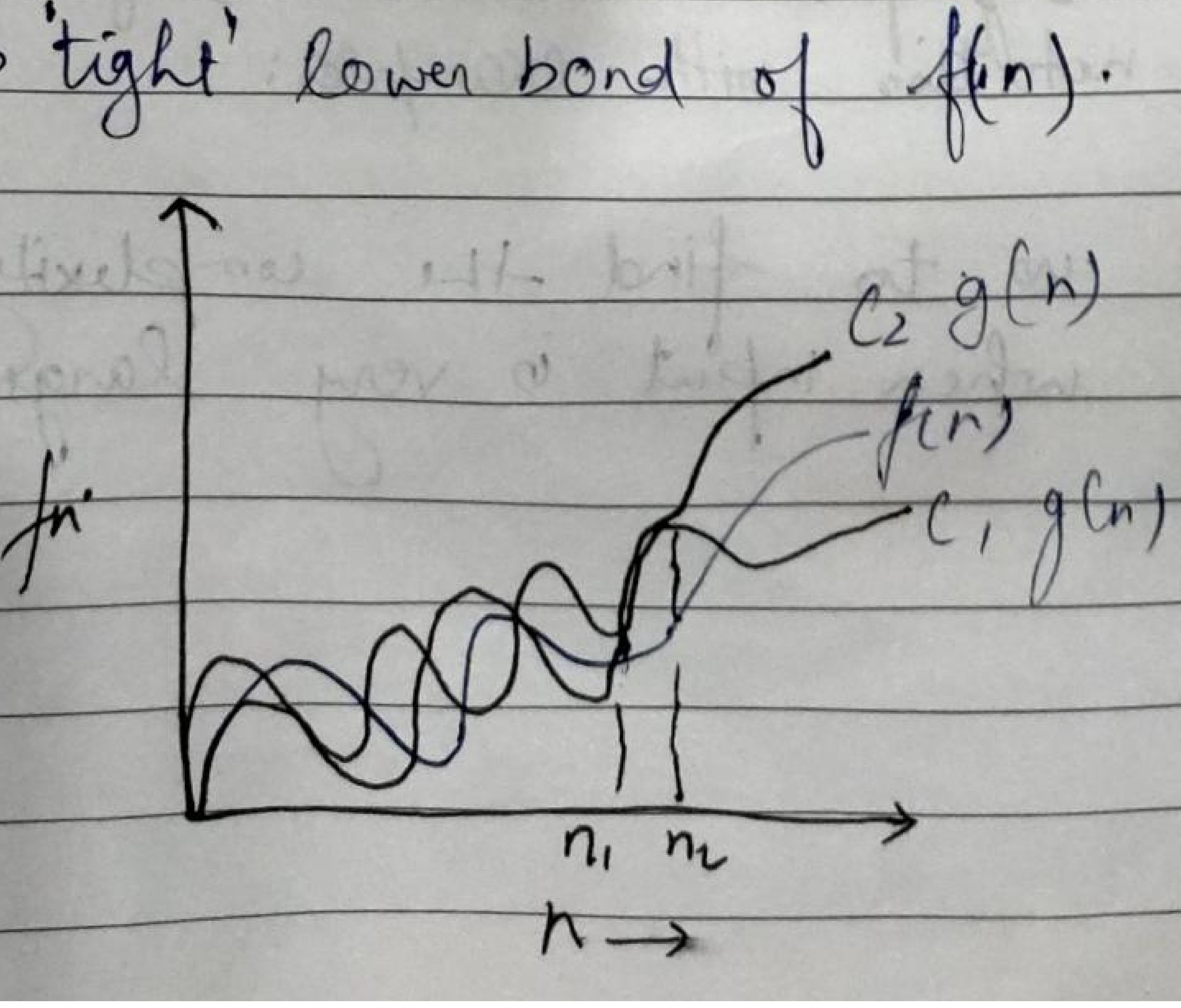
$f(n) \ge c\,g(n)$

$\forall$

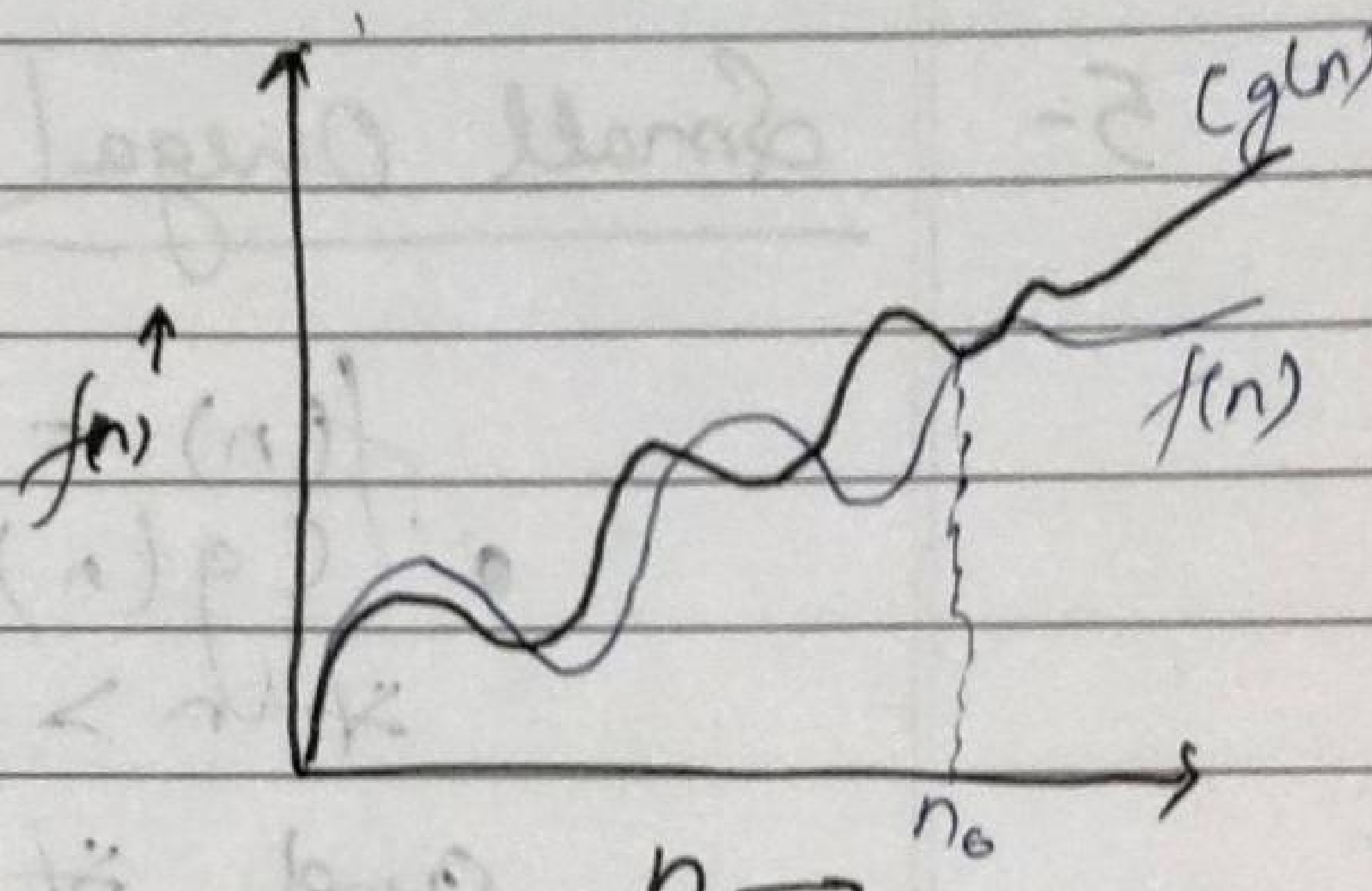
$n \ge n_0$

for some constant, $c > 0$

$g(n)$ is 'tight' lower bond of $f(n)$.



3- <u>Theta(θ)</u> → $f(n) = \theta(g(n))$



$f(n) = \theta(g(n))$.

iff

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

$\forall n \ge \max(n_1, n_2)$

for some constant $c_1 > 0$ and $c_2 > 0$.

g(n) is both 'tight' upper and lower bond of function f(n).

4- <u>Small θ(θ)</u>: $f(n) = \theta(g(n))$

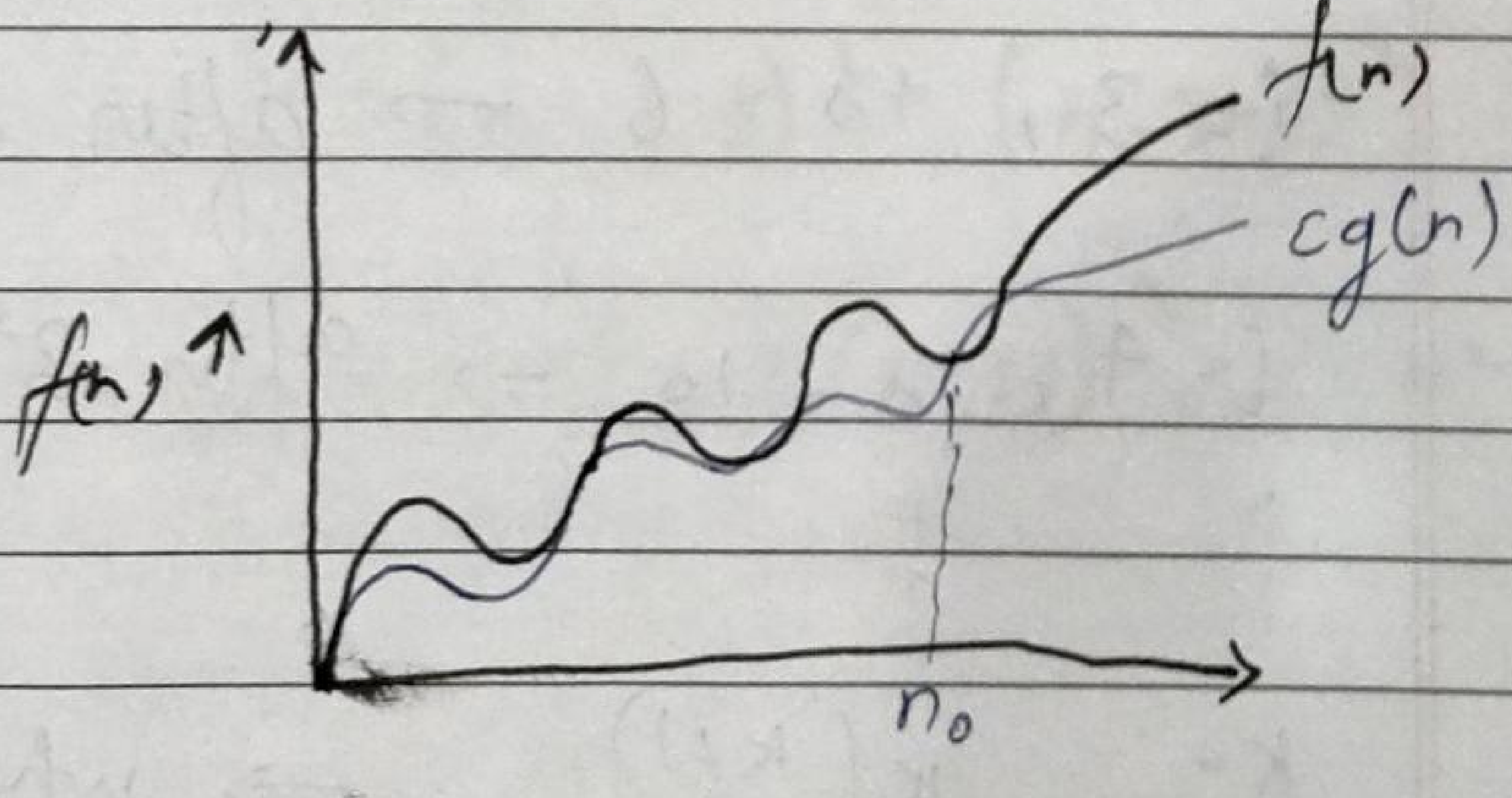
$f(n) = \theta(g(n))$

When

$f(n) < g(n)$

$\forall n > n_0$ and $\forall$ constant ; $c > 0$.

g(n) is upper bond of function f(n).



5- Small Omega (ω): $f(n) = \omega(g(n))$

$f(n) = \omega(g(n))$, when
$c\,g(n) < f(n)$
$\forall n > n_0$
and $\forall$ constants, $c > 0$.
$g(n)$ is lower bond of function $f(n)$.

5 - Time complexity :

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

→ for $i = 2$, $s = 3$ → after 1st iteration

$i = 3$, $s = 6$ → after 2nd iteration

$i = 4$, $s = 10$ → after 3rd iteration

$\vdots \qquad \vdots$

$k \cdot \quad \frac{k(k+1)}{2}$ → when $i = k$,

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

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

Scanned by TapScanner

$\therefore$ $O(k^2) \leq n$

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

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

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

$T(n) = T(n-3) + n^2$ — ①

$T(1) = 1$

put $n = n-3$ in ①

$T(n-3) = T(n-6) + (n-3)^2$ — ②

put $n = n-6$ in ①

$T(n-6) = T(n-9) + (n-6)^2$ — ③

puting ② in ③

$T(n) = T(n-6) + (n-3)^2 + n^2$ — ④

putting $T(n-6)$ in ④

$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n^2$

$= T(n-3\cdot3) + (n-3\cdot2)^2 + (n-3\cdot1)^2 + (n-3\cdot0)^2$

$T(n) = T(n-3k) + (n-3(k-1))^2 + (n-3(k-2))^2 \ldots n^2$

putting $n-3 = 1$; $n = 3k+1$

$k = \frac{n-1}{3}$

$T(n) = T(1) + (1+3)^2 + (1+6)^2 \ldots n^2$

$= 1 + 4^2 + \ldots + n^2$

$T(n) = O(n^2)$

9- Time complexity of:

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

→ for $i=1$, $j=1,2,3, \ldots n$

for $i=2$, $j=1,3,5, \ldots n/2$

for $i=3$, $j=1,4,7, \ldots n/3$

$i=n$, $j=1+1+\ldots 1$

$\frac{n}{1}+\frac{n}{2}+\frac{n}{3}+\ldots+\frac{n}{n-1}-\log(n-1)$

$= n\left\{1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n+1}\right\}-\log(n-1)$

$= n\log(n-1)-\log(n-1)$

$= n\log(n-1)$

$T(n) = n\log n$