Design and Analysis of Algorithms
Assignment - 1

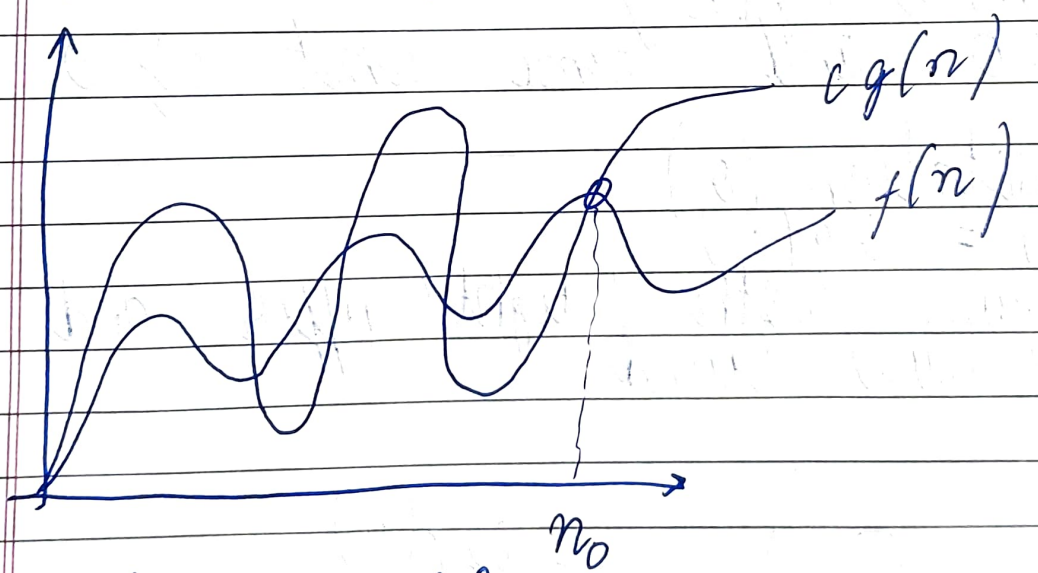
(1) Asymptotic notations are used to tell the complexity of an algorithm when the input is very large.

Different asymptotic notations are :-

1) Big O notation

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

$g(n)$ is 'tight' upper bound of $f(n)$



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

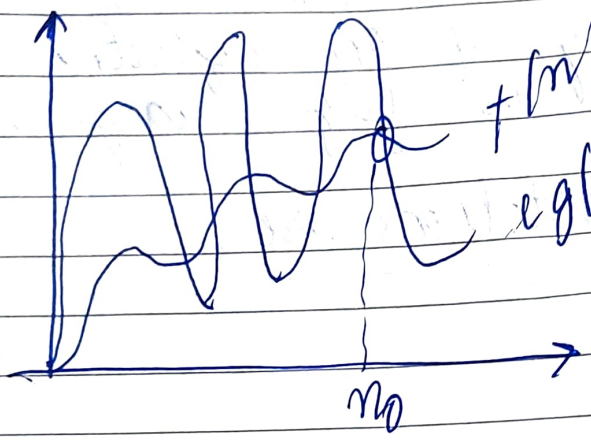
iff $f(n) \leq g(n)$

$\forall\ n \geq n_0$ and for some constants $c > 0$.

(ii) Big Omega Notation (Ω)

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

$g(n)$ is tight lower bound of $f(n)$.



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

iff

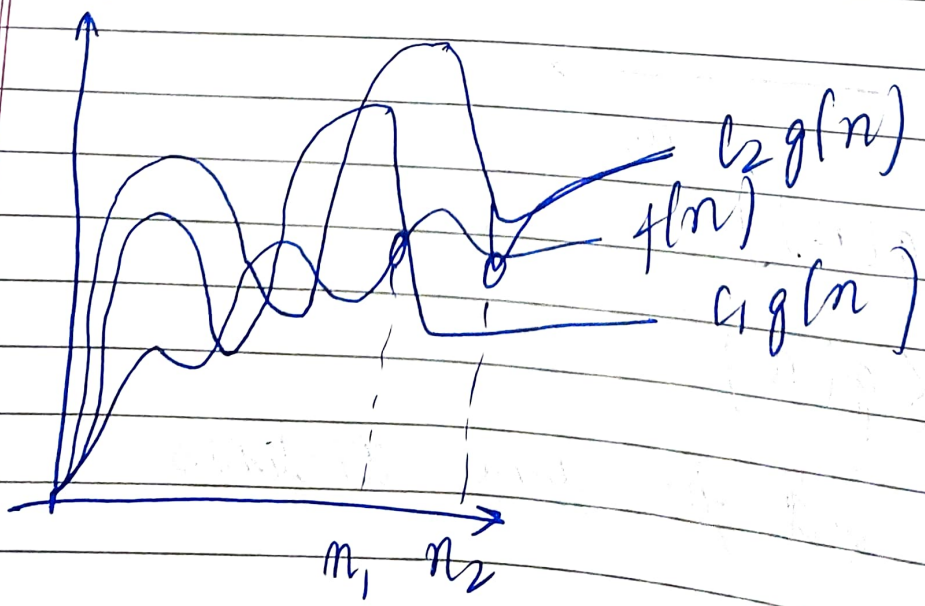
$f(n) \geq cg(n)$

$\forall\ n \geq 0$ and some const $c > 0$

(iii) Theta Notation (Θ)

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

theta gives both 'tight' upper and 'tight' lower bound.



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

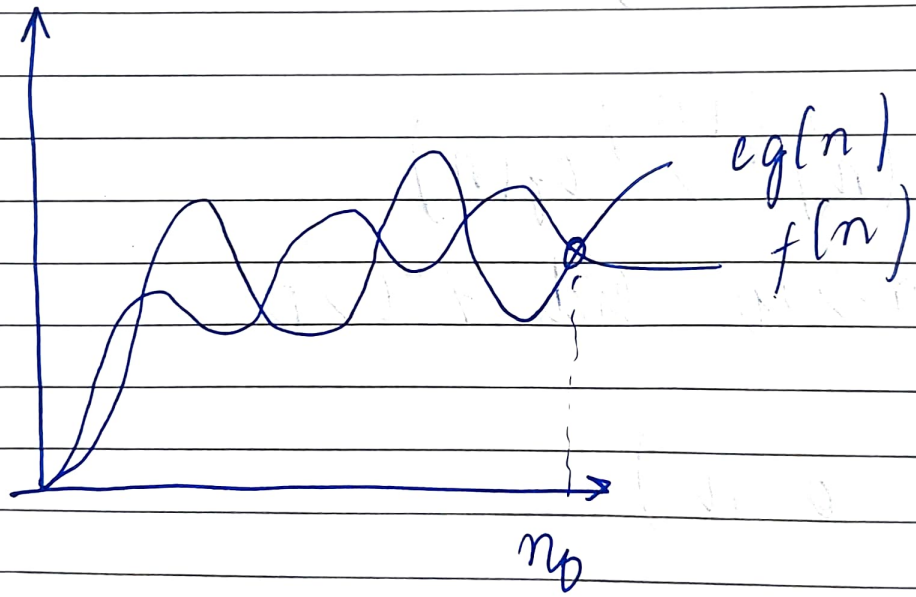
iff $c_1 g(n) \leq f(n) \leq c_2 g(n)$

$\forall\ n \geq \max(n_1, n_2)$ and for some constant $c_1, c_2 > 0$

(iv) Small o notation (o)

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

$g(n)$ is upper bound of $f(n)$



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

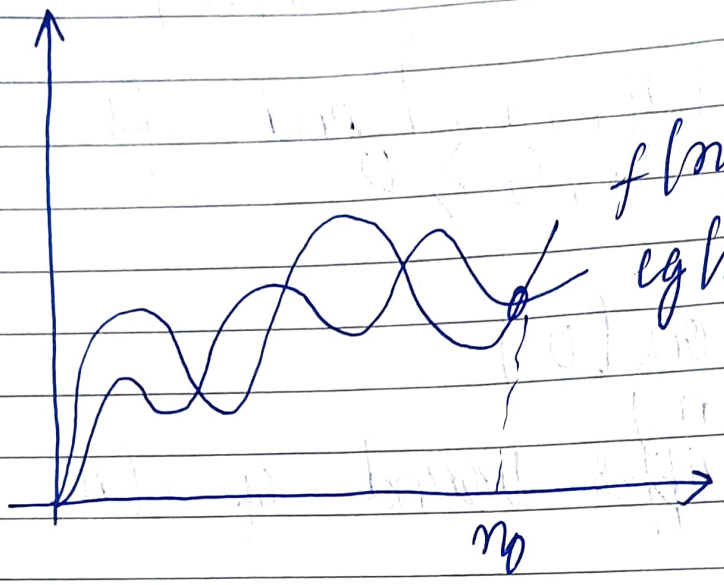
iff $f(n) < cg(n)$

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

(v) small omega notation → (ω)

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

$g(n)$ is lower bound of $f(n)$



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

iff $f(n) > cg(n)$

$\forall \quad n > n_0$ and $\forall \; c > 0$

(2)

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

1 2 4 8 ---- n

2^0 2^1 2^2 2^4

$t_k = ar^{n-1}$

$n = 1 * 2^{k-1}$

$n = 2^{k-1}$

$\log_2 n = k-1$

$\log_2 n + 1 = k$

$k = \log_2 n + 1$

$O(\log_2 n)$

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

$T(1) = 1$

~~let $T(n) = T(n-1) + n$ — (1)~~

let $T(n) = 3T(n-1)$ — (1)

put $n = n-1$ in (1)

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

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

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

Now

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

$T(n) = 9T(n-2)$ — (2)

Now,

put $n = n-2$ in (1)

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

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

v)

Now in ②

$T(n) = 9\{3T(n-3)\}$

$= 27\,T(n-3)$

Now,

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

Let $n-k=1$

$n = k+1$

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

$T(n) = 3^k T(1)$

$T(n) = 3^k$

→ $O(3^n)$ //

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

$T(n) = 2T(n-1) - 1$ - ①

② put $n = n-1$ in ①

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

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

put $T(n-1)$ in ①

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

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

$T(n) = 4T(n-2) - 3$ — (2)

now put $n = n-2$ in (1)

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

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

Now in (2)

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

$= 16T(n-4) - 12 - 3$

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

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

~~put~~ Now, $n-k = 1$

$n = k+1$

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

$= 2^k - 2^k + 1$

$= 1$

Time complexity

→ $O(1)$

(5) i = 1, s = 1

i	1	2	3	4	5
s	1	3	6	10	15

```
while (s <= n)
{
    i++;
    s = s + i;
    print ("#");
}
```

→ $O(\sqrt{n})$

(6)

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

loop iterates from i = 1 to $i * i <= n$ which means it runs approximately $\sqrt{n}$ times

→ $O(\sqrt{n})$

(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++;
        }
    }
}
```

}

outer loop runs $n/2$ times
middle loop ~~runs~~ doubles j in each iteration (j = j * 2)
it runs $\log_2 n$ times
innermost loop is similar to middle loop resulting in $\log_2 n$ iterations.

$O(n/2) * O(\log_2 n) * O(\log_2 n)$

$O(n * \log_2 (n)^2)$

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

→ $O(n^2)$

(9)

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

→ $O(n^2)$

(10)

$n^k \quad c^n$

$n=1$

$n^k = 1^k \quad c^n = c$

$n=2$

$n^k = 2^k, \quad c^n = c^2$

$n=k, \quad n^k = k^k, \quad c^n = c^k$

∴ we can say that for any value of $n \geq 0$

$n^k \geq cn$

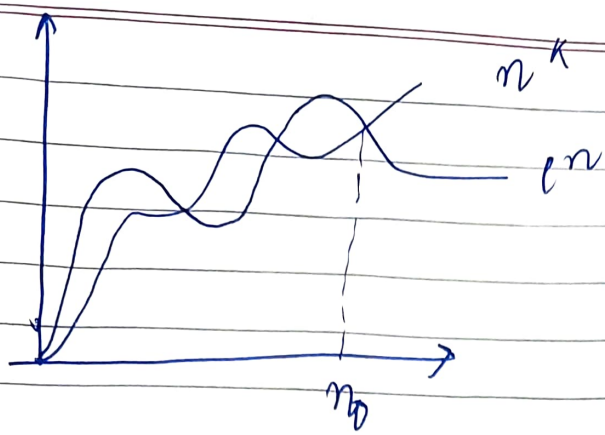
let $n^k = f(x)$. $g^n = c_0 g(n)$

∴ $f(n) \geq c_0 g(n)$

$c_0 > 0, \quad n \geq n_0$

∴ $f(n) = O(c_0 g(n))$

∴ $n^k = O(c^n)$



(11) extract min =>

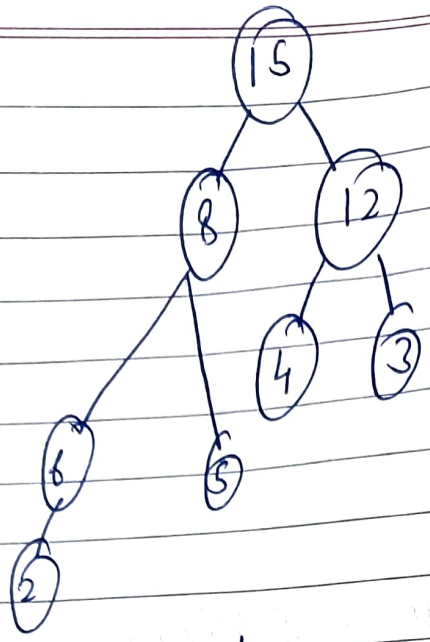
```
int extractMin ( vector<int> & hlop)
{
    if ( hlop. empty ( ) )
    {
        return -1 ;        → O(1)
    }
    swap ( hlop [0], hlop . back ( ) ) ; → O(1)
    int minElement = hlop. back ( ) ;
    hlop. pop_back ( ) ;   → O(1)
    heapify ( hlop, 0 ) ;  → O(log n)
    return min element ;
}
```

$T(n) = O(\log(n))$ // ans

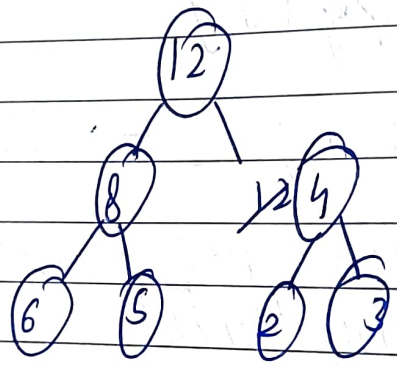
$$\left[\text{No. of } (n-1) * \left(1 * \frac{n}{2} \right) + 2 * \frac{n}{8} + 3 * \frac{n}{16} + \right.$$

$= \log(n)$ // harmonic mean

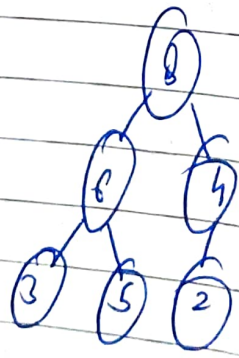
12



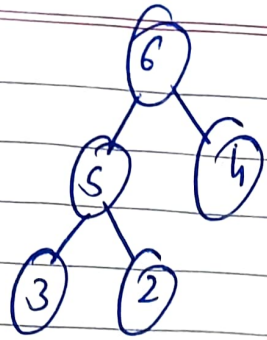
delete root 15



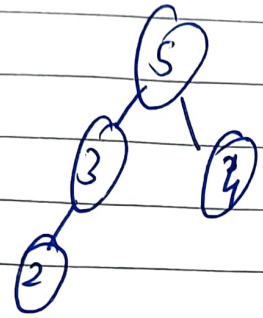
delete root 12



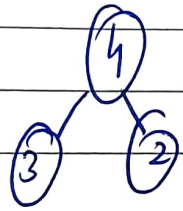
delete root 8



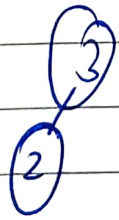
delete root 6



delete root 5

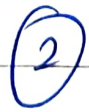


delete root 4



delete root 3

delete root 2



delete root 2

Heap is empty.