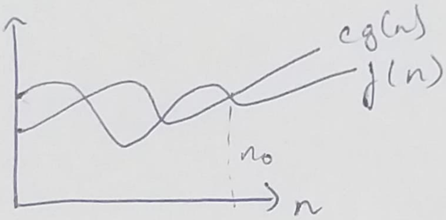
Assignment - 1

1. Asymptotic notations are used to represent the complexities of algorithms for asymptotic analysis.
These notations are mathematical tools to represent complexities

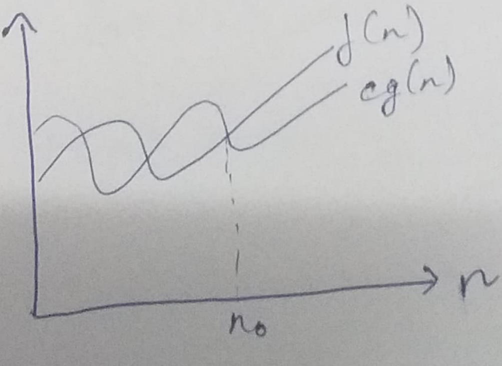
- Big oh notation
  Gives an upper bound for a function $f(n)$ within a constant factor

$$f(n) = O(g(n))$$
$$\text{iff } f(n) < c\,g(n)$$
$$\text{for } c > 0 \ \& \ n \geq n_0$$



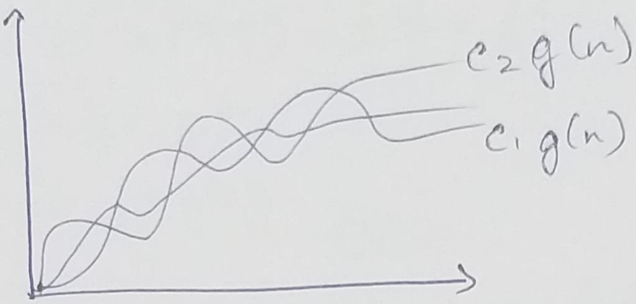
- Big Omega Notation
  Gives Lower bound for a function $f(n)$ within a constant factor c

$$f(n) = \Omega(g(n))$$
$$\text{iff } f(n) \geq c\,g(n)$$
$$f(n) \text{ for } c > 0 \ \& \ n \geq n_0$$
$$c\,g(n)$$



- Big Theta Notation
Gives bound for a function $f(n)$ within a constant factor

$$f(n) = \Theta(g(n))$$
$$\text{iff } c_1 g(n) \le f(n) \le c_2 g(n)$$
$$\text{for } c_1, c_2 > 0 \ \& \ n \geq n_0$$



2) Time Complexity for

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

$i = 1 \quad 2 \quad 4 \quad 8 \quad n$

$2 \quad 2^1 \quad 2^2 \quad 2^3 \quad 2^k$

G.P. ar^{k-1}

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

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

$\log 2n = k \log 2$

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

$\log n = k$

$\Rightarrow T(n) = O(\log(n))$ Ans

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

$T(0) = 1$

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

$= 3$

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

$= 9 = 3^2$

$T(3) = 3T(2) = 27 = 3^3$

$T(n) = 3^n$

$\Rightarrow O(3^n)$ Ans

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

$T(n) = 4T(n-2) - 3$ — ②

Put $n = n-2$

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

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

Put in ②

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

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

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

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

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

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

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

5.) while (s <= n)

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

$i = 1 \Rightarrow i{+}{+},\ i = 2$

$s = 3$

$i = 3$

$s = 6$

$i = 4$

$s = 10$

$i = 5$

$s = 15$

$i =$	2	3	4	5
$s =$	$s+1+2$	$s+1+2+3$	$s+1+2+3+4$	$s+1+2+3+4+5$

$s = s + 1 + 2 + 3 + 4 + \cdots k$

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

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

$k^2 \leq n$

$k \leq \sqrt{n}$

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

6.) void fn (int n)

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

$k^2 \leq n$

$k \leq \sqrt{n}$

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

Ans

$i =$	1	2	3	4 ...
$i^2 =$	1	4	9	16 ... k^2

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++;

}

}

}

$$T(n) = T(n/2) * Log(n) * Log(n)$$

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

$$= O(n \log^2 n) \text{ Ans}$$

8.) fun (int n)

{

if (n == 1)

return;

for (i = 1 to n) → n

{

for (j = 1 to n) → n

{

printf("*");

}

}

fun (n-3);

}

$$T(n) = n * n * [T(n-3)]$$

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

$$= O(n^2) \text{ Ans}$$

9.) void fun(int n)

```
{
  for (i=1 to n) → n
  {
    for (j=1; j<=n; j=j+i)
    {
      print("*");
    }
  }
}
```

$i=1, j=1,2,3,4, \ldots n \rightarrow n/1$

$i=2, j=1,3,5,7, \ldots n \rightarrow n/2$

$i=3\ j=1,4,7,11, \ldots n \rightarrow n/3$

$i=n-1, j=1, n \quad \longrightarrow n/n=1$

$$n+\frac{n}{2}+\frac{n}{3}+\cdots\frac{n}{n} = \log(n)$$

$$T(n) = n * \log n = O(n \log n) \text{ Ans}$$

10.) $n^k \qquad c^n$

$n=1,$

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

$n=2$

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

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

$\therefore$ we can say that for any value of $n>0$

$n^k \geq c^n$

let $n^k = f(n)$, $c^n = c_0 g(n)$

$\therefore f(n) \geq c_0 g(n)$

$c_0 > 0, n_0 \geq n_0$

$\therefore f(n) = O(c_0 g(n))$

$\therefore n^k = O(c^n)$ Ans

11) Extract Min ⇒

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

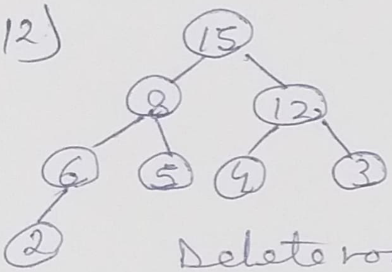
No. of comp ⇒

$$\left[\left(1 * \frac{n}{4}\right) + 2 * \frac{n}{8} + 3 * \frac{n}{16} + (n-1) * 1 \right.$$

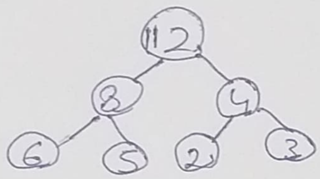
$= \log(n)$ Ans

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

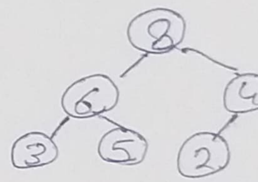
12)



Delete root 15 ⇒

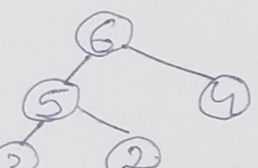


Delete root 12 ⇒

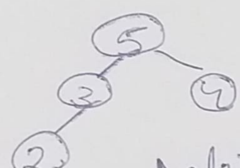


Delete root 8

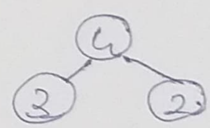
⇒



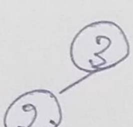
Delete root 6 ⇒



Delete root 5 ⇒



Delete root 4



Delete root 3 ⇒

Delete root 2

⇒ heap is empty.