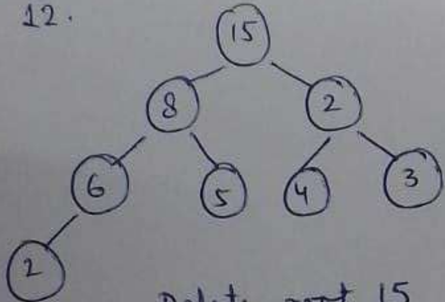
11. Extract Min :-

```
int extract_min( vector<int> &heap)
{ if (heap.empty())
    { return -1;      → O(1)
    }
  swap (heap(0), heap.back()); → O(1)
int minelement = heap.back();
heap.popback(); → O(1)
heapify (heap); → O(log n)
return min_element;
}
```

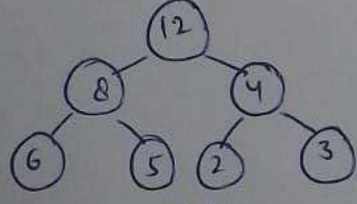
$T(n) = O(\log(n))$

12.

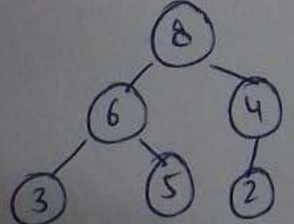


Delete root 15

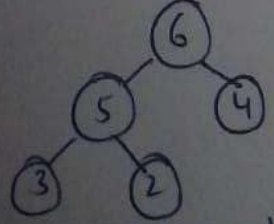
~~Delete root~~ 4



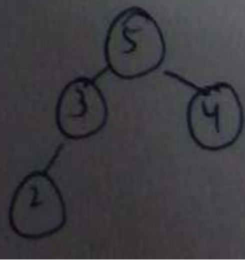
Delete root 12



Delete root 8



Delete root 6



8.
```
fun^n (int n)
{  if (n==1)
     return;
   for (i=1 to n)  → n
   { for (j=1 to n)  → n
     { printf("*");
     }
   }
   fun(n-3)
```
$T(n) = n * n * [T(n-3)] = n^2 + T(n-3)$
$= O(n^2)$

9.
```
void fun^n (int n)
{  for (i=1 to n)
   { for (j=1; j>=n; j=j+1)
     { print("*");
     }
} }
```
$i=1,\ j=1,2,3,4 \ldots n$

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

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

$n + \frac{n}{2} + \frac{n}{3} \ldots \frac{n}{n} = \log n$

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

10. n^k and c^n

$n=1$

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

$n=2$

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

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

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

$n^k \geq c^n$

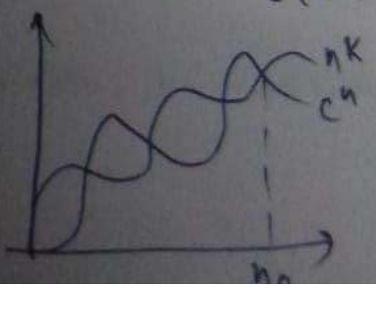
let, $n^k = f(n).\ c^n = \log(n)$

$f(x) \geq \log(n)$

$c_0 > 0,\ n_0 \geq n_0$

$\therefore f(x) = O(\log(n))$

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



5. while (s <= n)

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

$i = 1 = i++,\ i = 2$

$s = 3$

$i = 3$

$s = 6$

$i = 4$

$s = 10$

$i = 5$

$s = 15$

$i = 2 \quad 3 \quad 4 \quad 5 \longrightarrow$

$s+1+2 \quad s+1+2+3 \quad s+1+2+3+4$

$s = s+1+2+3+4 - - - k$

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

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

$k^2 \leq n \Rightarrow k \leq \sqrt{n}$

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

6. void funⁿ (int n)

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

$i = 1 \quad 2 \quad 3 \quad 4 \; - \; -$

$i^2 = 1 \quad 4 \quad 9 \quad 16 \; - \; - \; k^2$

$k^2 \leq n \Rightarrow \boxed{T(n) = O(\sqrt{n})}$

7. Void funⁿ (int n)

```
{ int i, j, k, cnt =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++;
}
```

$T(n) = T(n/2) + \log n + \log n$

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

$= O(n \log n)$

2. T.C for → P ($i = 1$ to n)

$i = i + 2$

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

$2^0 \quad 2^1 \quad 2^2 \quad 2^3 \; --- \; 2^k$

G.P = $a r^{k-1}$

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

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

$\log 2n = k \log 2$

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

$\boxed{\log n = k}$

$\therefore \; T(n) = O(\log(x))$

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

$T(0) = 1$

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

$= 3$

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

$= 9$

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

$T(n) = 3^n$

$= O(3^n)$

4. $T(n) = 2T(n-1) - 1, \; n > 0$ otherwise —— 1

Let $n = n-1$

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

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

put $T(n-1)$ in 1

$T(n) = 4T(n-2) - 3$ —— 2

put $n = n-2$

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

out in 2

$T(n) = 8T(n-3) - 5$

$(n-k) = 1$

$k = n-1$

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

$\boxed{T(n) = O(2^n)}$

Daksh Jain

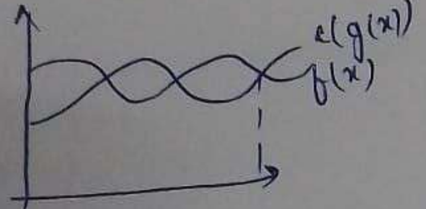
Sec - DS1, Roll no. → 40

1. Asymptotic notations are used to represent the complexities for algorithms for asymptotic analysis.

→ These notations are mathematical tools to represent complexities.

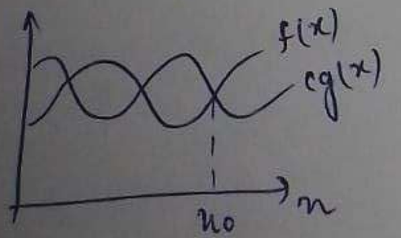
<u>Big O notation</u> → Gives the upper bound for a f(x) with a const. factor.

$$f(x) = O(g(x))$$
$$\text{if } f(x) \leq O(g(x))$$
$$\text{for } c > 0 \text{ and } n \geq n_0.$$



<u>Big Omega Notation</u> → Gives the lower bound for a f^n p(x) within a const. factor.

$$f(x) = \Omega(g(x))$$
$$\text{if } f(x) \geq g(x)$$
$$f(x) \text{ for } c > 0 \text{ and } n \geq n_0$$



<u>Big Theta Notation</u> → Gives bound for a fun^n f(x) within a const factor.

$$f(x) = O(g(x))$$
$$\text{if } c_1 g(x) = f(x) \leq c_2 g(x)$$
$$c_1 > c_2 > 0$$
$$\text{and } n \geq n_0$$