Assignment : 1

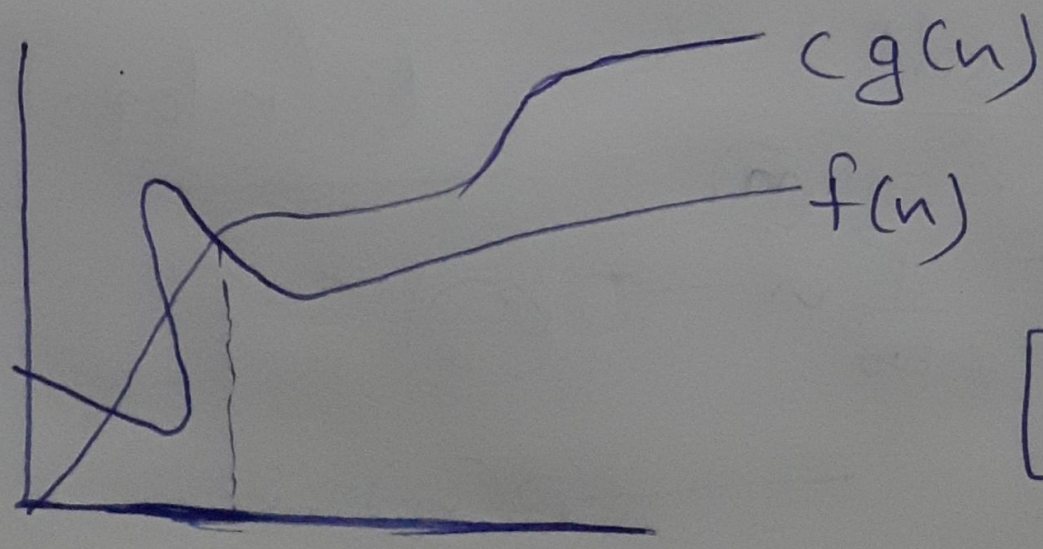
Ans-1) **Asymptotic Notations :**

It is a way to express the running time or space complexity of an algorithm in terms of its input size. It's a shorthand that gives a quick measure of a function's behavior as the input size grows large.

Three types →
- Big-O Notation (O-notation)
- Omega Notation (Ω-notation)
- Theta Notation (Θ-notation)

* **Big O Notation** - Represents the upper bound of the running time of an algorithm. It gives the Worst case complexity of an algorithm.

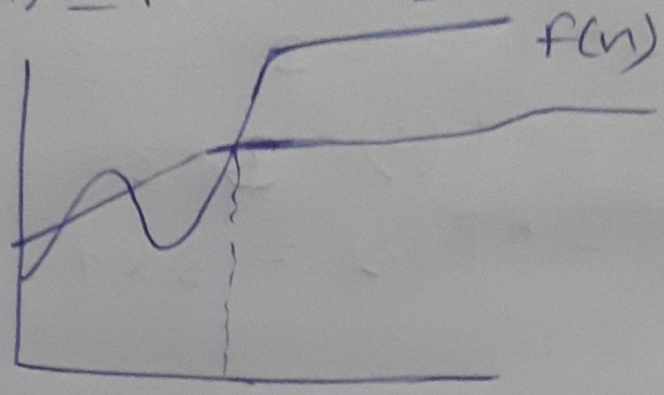
$$0 \leq f(n) \leq cg(n) \quad \forall \; n \geq n_0$$



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

* Omega Notation (Ω-Notation): Represents the lower bound of running time of an Algorithm. It provides the best case complexity of an algorithm.

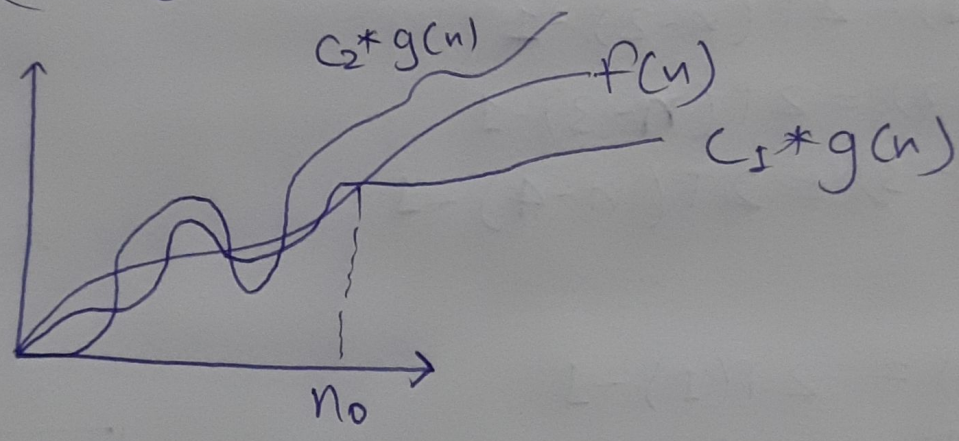
$C * g(n) \leq fn$ if $n \geq n0$



[f(n) = Omega(g(n))]

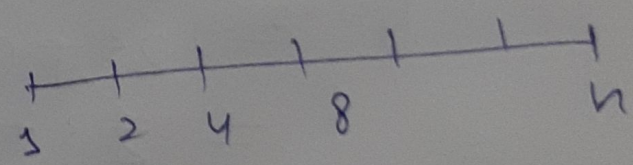
* Theta Notation (Θ-Notation): Encloses the function from above and below. Represents the upper and the lower bound of the running time of an algorithm. for analyzing Average case complexity of an algorithm.

$(C_1 * g(n) \leq f(n) \leq C_2 * g(n)$ if $n \geq n_0)$



Ans → (2)

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



← k elements.

GP: $n = a \times r^{k-1}$

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

$k = \log_2 n$

Time complexity: O(log n)

Ans-③ When $n>0$; $T(n) = 3*T(n-1)$

$n=0$; $T(n)=1$

If $n=1$; $T(1)= 3*T(0)= 3*1=3$

$n=2$; $T(2)= 3*T(1)= 3*3=9$

$n=3$; $T(3)=3*T(2) = 3*9=27$

Time complexity = $O(3^n)$

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

$T(0)=1$.

Substitute $T(n-1)$:

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

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

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

⋮

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

$T(1)= 2T(0)-1$

$\boxed{T(n)=1}$

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

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

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

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

$=$ - — —

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

$= 1$

Ans- 5)

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

5 times

's' is incremented by 'i' in each iteration, where 'i' is incremented linearly.
So, the sum of first 'i' integers is essentially $O(i^2/2)$

∴ Time complexity of loop $O(\sqrt{n})$, as the loop iterates until the sum of the first 'i' integers exceeds 'n' and 'i' grows as the square root of 'n'.

Ans → 6) Time complexity is $O(\sqrt{n})$.
loop iterates until i*i is less than or equal to n, which means it iterates roughly sqrt(n) times.

```
Void function (int n)
{
   int i, count=0;
   for (int i=1; i*i<=n; i++)  → √n+1 times
      count ++;  ——————  1 time
}
```

Time complexity = $O(\sqrt{n})$

Ans → 7) void function (int n)

```
{
  int i, j, k, count = 0;
  for (int i = n/2 ; i <= n ; i++) ⟶ n/2 + 1 times
    for (j = 1 ; j <= n ; j = j * 2) ⟶ log2 n times
      for (k = 1 ; k <= n ; k = k * 2) ⟶ log2 n times
        count ++ ⟶ 1 time
}
```

$= \frac{n}{2} (\log_2 n)(\log_2 n)$

$= \frac{n}{2} (\log_2 n)^2$

Time complexity $= O(\frac{n}{2}(\log n)^2$

$= O(n + \log^2 n)$

Ans - 8)

```
function (int n)
{
  if (n == 1)
    return;
  for (i = 1 to n)
  {
    for (j = 1 to n)
    {
      printf ("*");  ⟶ n times → O(n)
    }
  }
  function (n - 3);  ——— n/3 times
}
```

$O(n^2)$

Geometric series: $T(n) = O(n^2) + T(n-3) + T(n-6) + \cdots + T(1)$

* Recursive calls reduces by 3 each time.
let no of Recursive calls = K

$n, n-3, n-6 - - - - 1$

Arithmetic seq: (d=3)

$\text{Sum} = \left(\frac{k}{2}\right) * (n+1)$

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

$\approx \frac{k^2}{2}$ (if $k >>$)

Sum $\approx n$

$\therefore k \approx n$

$k \propto n$

Time complexity $= O(n^2) * O(n)$

$= O(n^3)$

$\therefore$ Time complexity

Ans-9)

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

n times

n times (for i = 1)

n/2 times (for i = 2)

Inner loop iterations: $1 + 2 + 3 + \cdots + n$ of n^2

Time complexity $= O(n^2)$

[As the Dominant factor comes from nested loops]

Ans→10) n^k grows polynomially with base n
* c^n grows exponentially with exponent n.

∵ $k \geq 1$ and $c > 1$, c^n will eventually grows faster than any polynomial function n^k.

Asymptotic relationship -

$c^n \rightarrow O(c^n)$ (c^n grows faster than n^k)

$n^k \rightarrow O(n^k)$

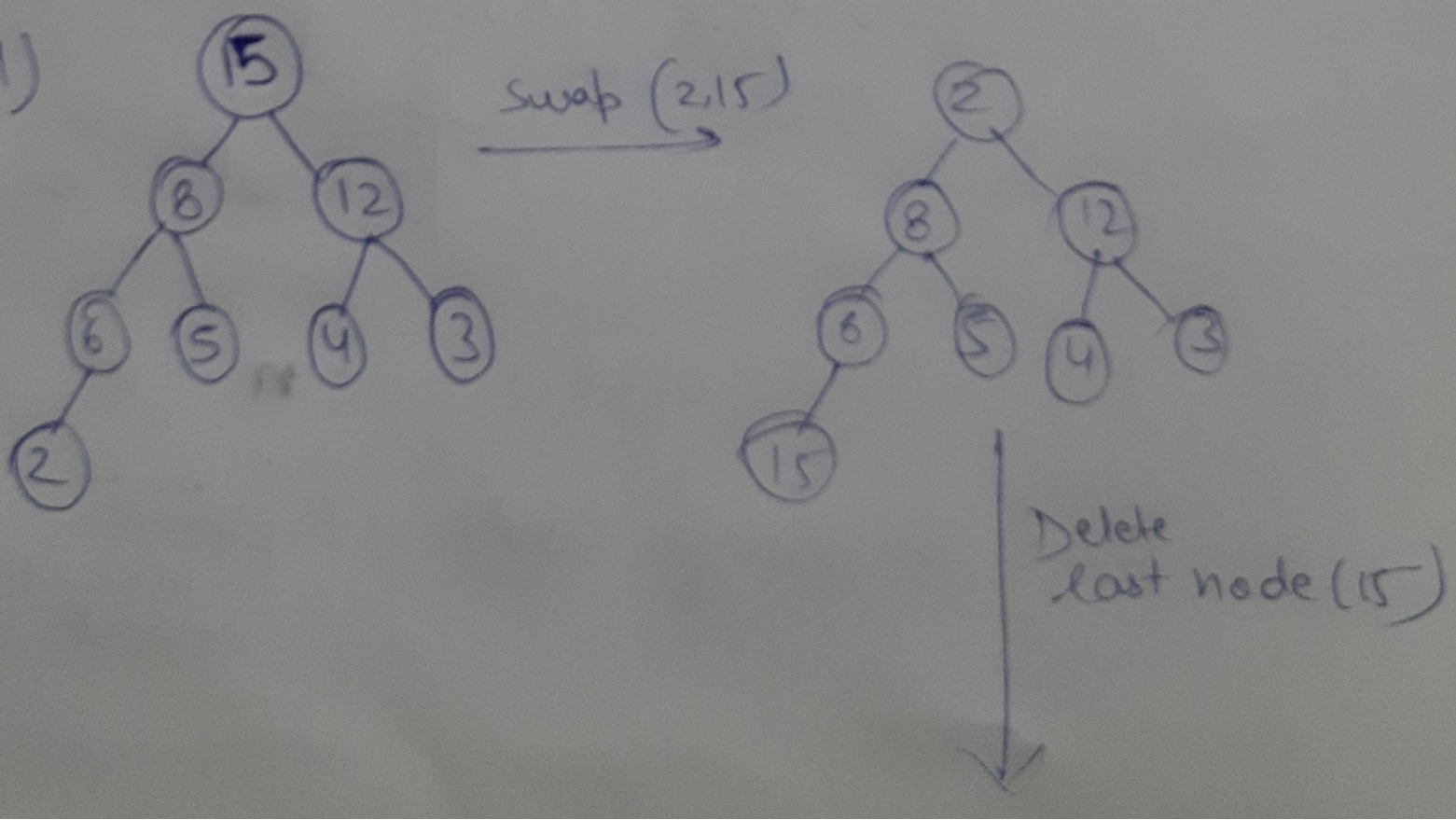
$\Rightarrow \lim_{n \to \infty} \frac{c^n}{n^k} = \infty$

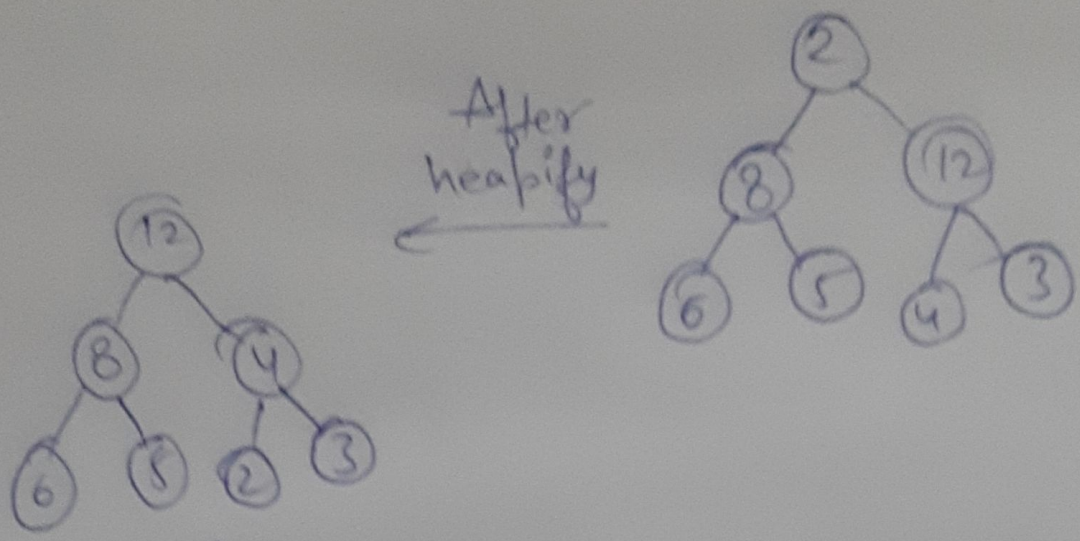
∴ $c > 1$, c^n will surpass any polynomial function n^k as $n >>$.

Ans→11) The Complexity of Extract Min:
when it returns only 1 minimum element from heap.

Time Complexity = $O(1)$ | As it only delete root's node

Ans-12)




After heapify
swap(12,6) & delete last node
heapify
heapify
Swap (8,5) & delete 8
heapify
heapify
heapify
Delete last element
All nodes Deleted