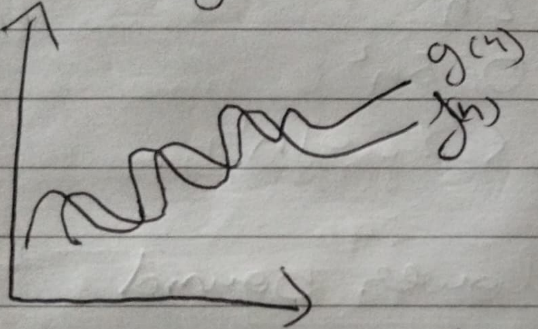
Assignment-1
DAA

→ These notations are used to tell the complexity of an algorithim

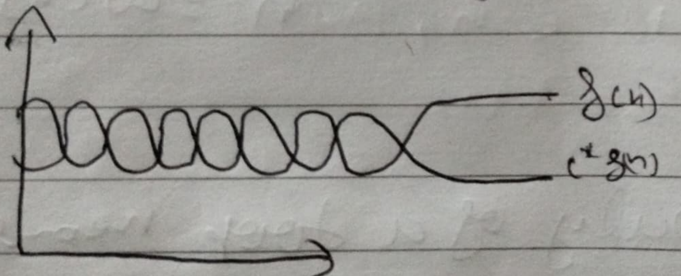
-> It describes the algorithm efficiency & performance in a meaningful way.

1) big oh notation - The function $f(n) = O g(n)$, if & only if there exist a const c (+ve) & $f(n) \leq c\, g(n)$ for all $n, n \geq k$



$f(n) = O g(n)$
if
$f(n) \leq c\, g(n)$
$\forall\ n \geq n_0$
so, constant $c > 0$

2) big omega notation - The function $f(n) = \Omega(g(n))$, if there exist a the constant c & k such that $f(n) \geq c * g(n)$ for $n, n \geq k$

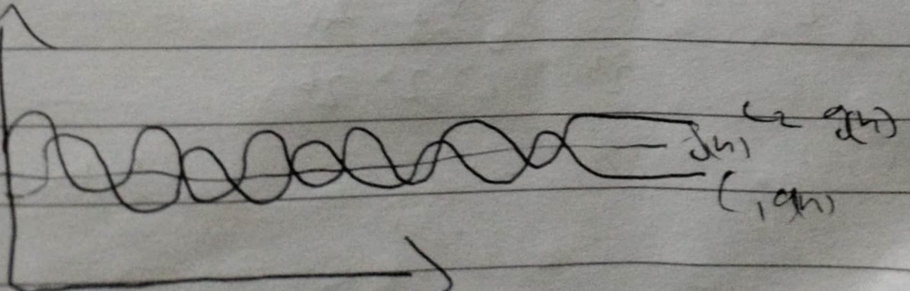


$f(n) = c\, g(n)$
$f(n) \geq c \cdot g(n)$
$\forall\ n \geq n_0$ & $c > 0$

3) big theta notation

Similarly,

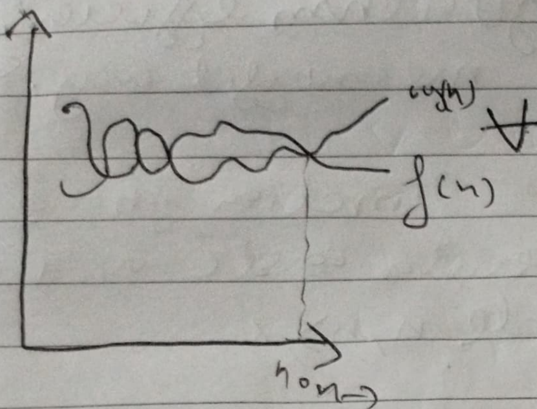
$f(n) = \Theta(g(n))$
if $c_1 g(n) \leq f(n) \leq c_2$
$\forall\ n \geq \max(n_1, n$



4) Small o notation

$f(n) = o(g(n))$: $g(n)$ is upper bound of $f(n)$ if & only if

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



$\forall\ n > n_0$ and for all constraints

$c > 0$

$n = o(n^2)$

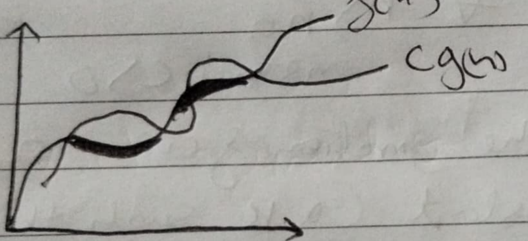
$n < 1n^2$

$2n^2$

$0.5n^2$

$n < 0.001n^2\ n_0$

5) Small omega



lower bound

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

$f(n) > c.g(n)$

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

$n^2 = \omega(n)$

Q2) for (i=1 to n)

```
{   i = i*2
}
```

; time complexity of a loop means the no of time it was to run

i 1   2   4   8   16   32 ---- 2^k

2^1 2^2 2^3 2^4 2^5 2^6 —— n

$i = 1, 3, 4, 8, 16, 32, \ldots, 2^k$ this means

k

i.e $2^k = n$ $k = \log n$

$k \log_2 2 = \log_2 n \Rightarrow \log_2 2 = 1$

$TC = O(\log n)$

Q3) $T(n) = \begin{cases} 3(T(n-1), & n>0 \\ 1 \end{cases}$

by forward Sustitution

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

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

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

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

$= 3 * 3 = 9$

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

$3T(2) = 3 * \cdot 3^2 = 3^3$

So, $T(n) = 3^n$

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

Q4) $T(n) = \begin{cases} 2T(n-1) - 1, & n>0 \\ 1 \end{cases}$

by forward substitution

$T(0) = 1$

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

$= 2^1 - 1$

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

$= 2^2 - 2^1 - 1$

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

$= 2^3 - 2^2 - 2^1 - 1$

$2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots 2^3 - 2^2$

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

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

$\boxed{TC = O(1)}$

5)
```
int i, s=1;
while (s<=n)
{   i++;
    s = s+i;
    printf("#");
}
```

The value of i increases for each ite-ration. The value contained in 's' at the ith iteration is the sum of the first 'i' +ve integers. If k is the total no of iterations taken by any prog.

$1+2+3+ \text{------} +k$

$[k(k+1)/2] > n$

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

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

(6)
```
void function (int n)
{
    int i, count=0
    for (i=1; i<=n; i++)
    {   count++;
    }
```

$\boxed{O(n)\ .\ T.C}$

(7)
```
void function (int n)
{
    int j, k, i, 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++;
}}}
```

TC = $\log n * \log n$
= $O(n \log n)$
TC = $O(n \log n)$

Q8)
```
function (int n)
{
        if (n == 1)
          { return n;
        for (i = 1 to n)
          {
            for (j = 1 to n)
              { printf ("*");
              }
          }
        function (n-3);
}
```
$\boxed{T.C = O(n^2)}$

Q9)
```
void function (int n)
{
      for (i = 1 to n) {
    for (j = 1; j <= n; j = j+1)
      { o (n)
         printf ("*");
      }
    }
```
$\boxed{TC = O(n^2)}$

Q10) for the function n^k a c^n, what is asymptotic motations b/w these func

assume, that $k>=1$ & $c>1$ are const

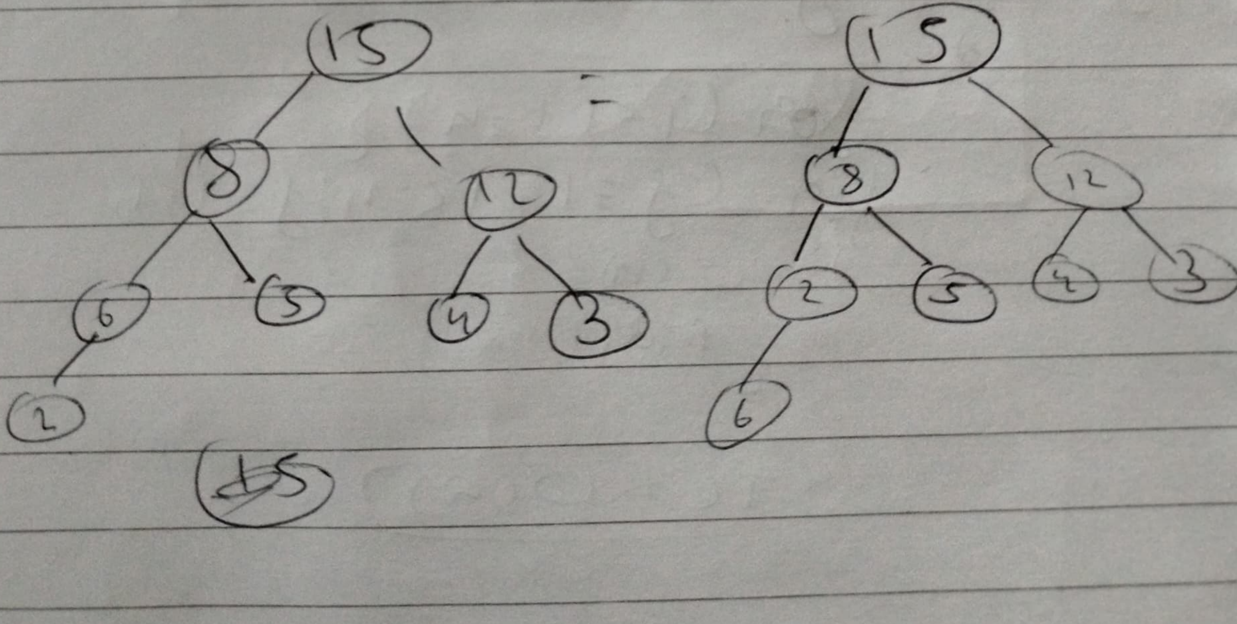
find out the val of c & n which holds

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

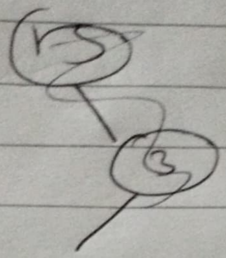
Q11)

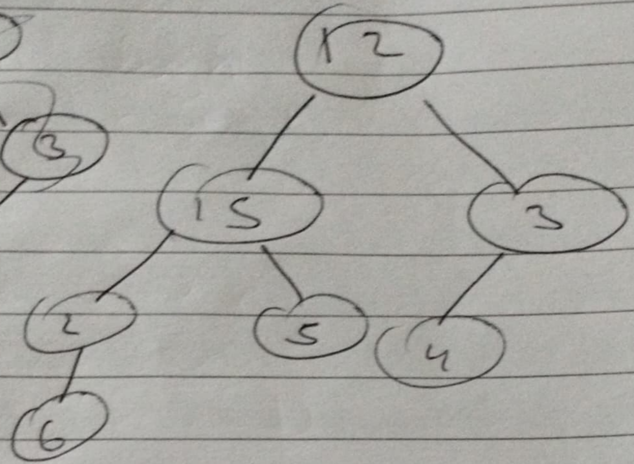
→ The time complexity of 'extractmin()' would depend on the underlying implementation of the data structure. if its a binary heap, the time complexity would be O(1) as the minimum element is always at the top. if its a different data structure the complexity may vary.

Q12) find max heap & del

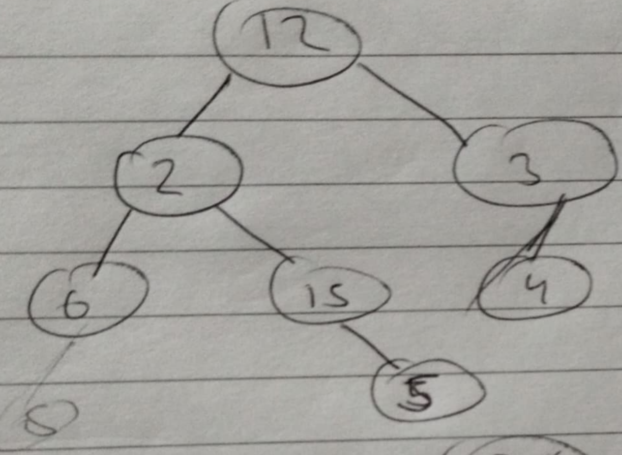


=>

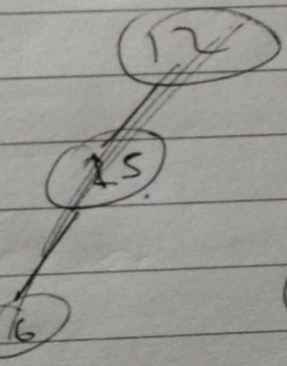


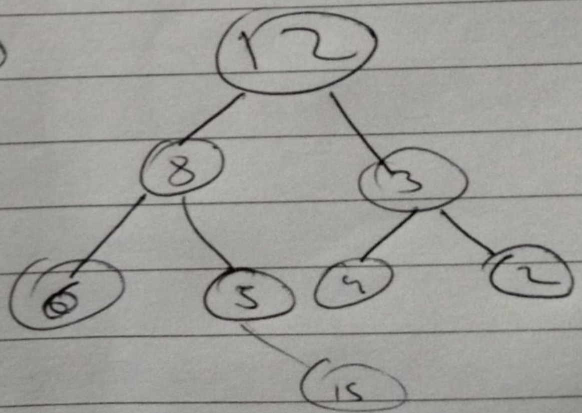


=>



=>





∴ 15 is max so,