ASSIGNMENT-1

1. Asymptotic notation are notations used to tell the complexity of an algorithm when the input is very big. Asymptotic notations are as follows:

i) Big O Notation: used to define the tight upper bound.

$$f(n) = O(g(n))$$
iff $f(n) \leq c\, g(n)\ \forall n \geq n_0$ & some const $c > 0$

Eg.: for Merge sort:
$$O(n\log n)$$

ii) ~~The~~ Big Ω Notation: used to define the tight lower bound.

$$f(n) = \Omega(g(n))$$
iff $f(n) \geq c\, g(n)\ \forall n \geq n_0$ & some const $c>0$

Eg.: for Merge Sort $\Omega(n\log n)$.

iii) Theta Θ Notation: gives both tight lower and upper bound.

if $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$
Then
$$f(n) = \Theta(g(n)).$$
iff $c_2\, g(n) \leq f(n) \leq c_1\, g(n)\ \forall n \geq \max(n_1, n_2)$ & $c_1, c_2 > 0$

iv) Small o Notation: used to give the upper bound.

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

if $f(n) < c\,g(n)\ \forall\, n > n_0$ & $\forall$ const. $c > 0$

v) Small Omega ω Notation: used to give lower bound.

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

if $f(n) > c\,g(n)\ \forall\, n > n_0$ and $\forall\, c > 0$.

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

$i =$ 1 2 4 8 16 32 ------ n

Let i's final value be 2^k
then

$$2^k = n.$$

taking log on both sides.

$$K \log_2 2 = \log_2 n.$$

$$\Rightarrow K = \log_2 n.$$

$\therefore$ T.C. $= O(\log_2 n)$ //

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

$\therefore\ T(0) = 1$. — (a)

$$T(n) = 3T(n-1) \text{ — } (1)$$

Let $n = n-1$.

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

then, $T(n) = 3 \times 3T(n-2)$ — (2)

Let $n = n-2$.

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

then, $T(n) = 3 \times 3 \times 3T(n-3)$ — (3)

Generalizing (3)

$$T(n) = 3^k T(n-k) \text{ — } (4)$$

$$1 = n-k \Rightarrow \boxed{n=k}$$

putting $n=k$ in (4)

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

$$\Rightarrow T(n) = 3^n T(0) \quad \{T(0)=1, \text{from (a)}\}$$

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

$$\therefore\ f(n) = O(3^n)$$

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

$\therefore\ T(0) = 1$ — (1)

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

Let $n=1$

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

$$= 2T(0) - 1 \Rightarrow 2-1 \ \{\text{from (1)}\}$$

$$T(1) = 1 \text{ — } (2)$$

Let $n=2$.

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

Date
Page

$= 2T(1) - 1 \Rightarrow 2 - 1$ {from ②}

$T(2) = 1$

Let $n = 3$.

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

$= 2T(2) - 1$

$= 2 - 1$

$T(3) = 1$ — ③

From the above examples, with increasing values of n, the time complexity stays 1.

$\therefore \quad T(n) = O(1)$

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

$i = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \ ---$

$s = 1\ 3\ 6\ 10\ 15\ 21\ 28\ 36 \ ---$

Let total no. of terms be k.

$S = 1\ 3\ 6\ 10\ 15 \ --- \ --- T_{k-1}\ T_k$

$S = \quad 1\ 3\ 6\ 10\ 15 \ --- \ --- T_{k-2}\ T_{k-1}\ T_k$

Subtracting:

$0 = 1\ 2\ 3\ 4\ 5 \ --- \ --- k$

$\Rightarrow T_n = \frac{n(n+1)}{2}$

$$k \le \frac{n(n+1)}{2}$$

$$2k \le n^2 + n.$$

$$k \approx \sqrt{n}$$

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

6.

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

loop runs till $i^2 < n$.
or till i reaches $\sqrt{n}$.

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

7.

```
void function(int n)
{
 int i,j,k, count=0;
 for(i=n/2; i<=n; i++)    — n/2
 {
   for(j=1; j<=n; j=j*2)    — log n
   {
     for(k=1; k<=n; k=k*2)    — log n
     { count++;
     }
   }
}}
```

Date
Page

for(j=1; j<=n; j=j*2)

j's final value = 2^k.

$2^k = n$.

taking log on both sides,

$K \log_2 2 = \log_2 n$.

$\Rightarrow K = \log_2 n$

$\therefore$ for j's loop T.C. = $O(\log_2 n)$

Similarly for k,

T.C. = $O(\log_2 n)$

$\therefore f(n) = O\left(\frac{n}{2} \times \log_2 n \times \log_2 n\right)$

T.C. = $O(n \log^2 n)$ //

8. ~~Find~~

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

T.C. = $O(n \times n)$

$= O(n^2)$

Date
Page

9.

```
void function (int n)
  for (i = 1 to n)                    — n times
      for (j = 1; j <= n; j = j+i)    — ~~n times~~
          printf (" * ");
```

$i = 1 \quad 2 \quad 3 \quad 4 \quad 5 \; --- \; n$

$j = 1$

no of times inner loop runs $= n \quad n/2 \quad n/3 \quad n/4 \quad n/5 \; --- \; 1$

for inner loop iterations

$$= n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \text{------} + 1.$$

$$= \log n.$$

$\therefore$ T.C. $= O(n \log n)$

10. $f(n) = n^k \quad , k >= 1$

$f(c) = c^n. \quad , c > 1$

Let $k = 1$ & $c = 2$.

c^n grows faster than n^k. In other words c^n has greater growth rate than n^k.

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

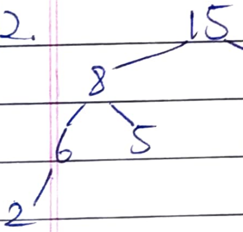
It can be said that c^n is the upper bound of n^k.

11. time taken to extract minimum element $= O(1)$ ~~since~~ minimum element is always at the top of a minHeap.

time taken to Heapify the remaining heap.

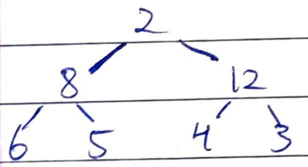
$= O(\log n)$

$\therefore$ T.C. $= O(\log n$

12. 

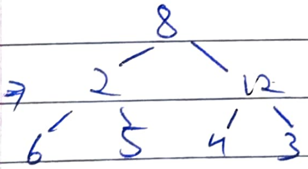
delete 15 and swap 15 with 2.

15	8	12	6	5	4	3	2
2	8	12	6	5	4	3	~~15~~



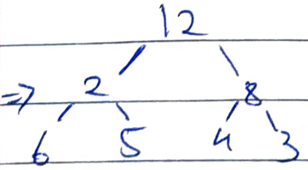
Heapify

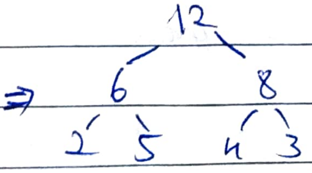
2	8	12	6	5	4	3



Heapify

8	2	12	6	5	4	3





12	2	8	6	5	4	3

12	6	8	2	5	4	3