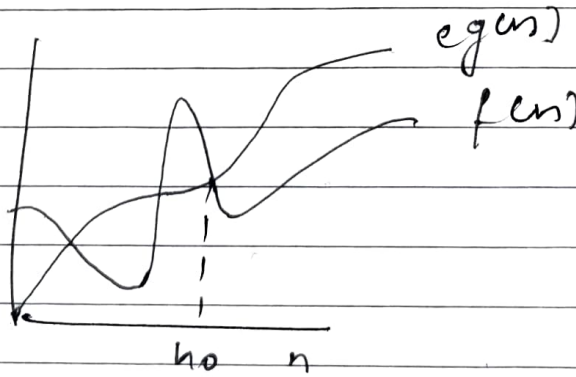
Date.
Page No.

Assignment 1.

A1. Asymptotic Notations are used to represent the complexities of algorithms for asymptotic analysis. These notations are mathematical tools to represent the complexities. There are three notations that are commonly used.

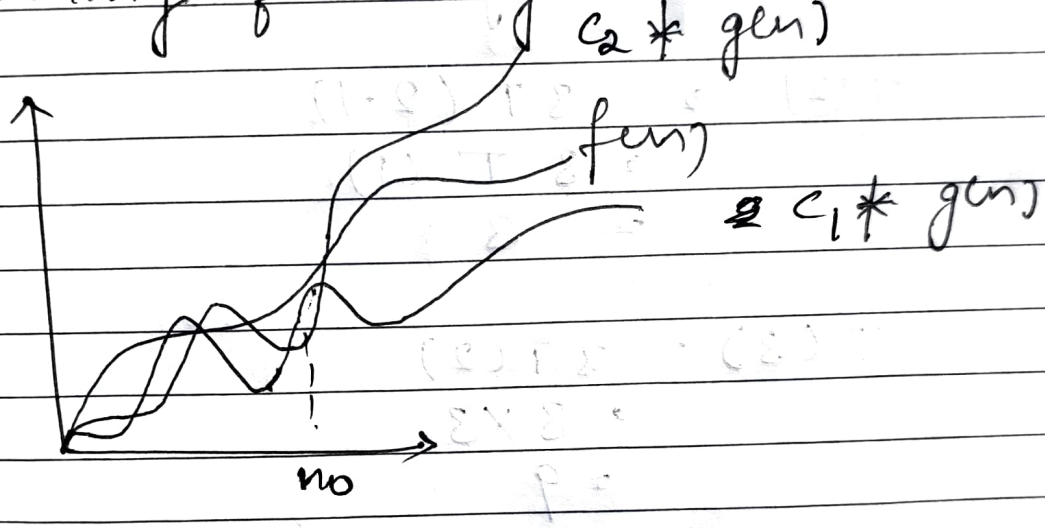
(i) Big Oh Notation : It gives an upper bound for a function f_n to within a constant factor. Therefore it gives the worst case complexity of an algorithm

If $f(n)$ describes the running time of an algorithm, $f(n)$ is $O(g(n))$ if there exist a positive constant c and n_0 such that $0 \leq f(n) \leq c\,g(n)$ for $n \geq n_0$



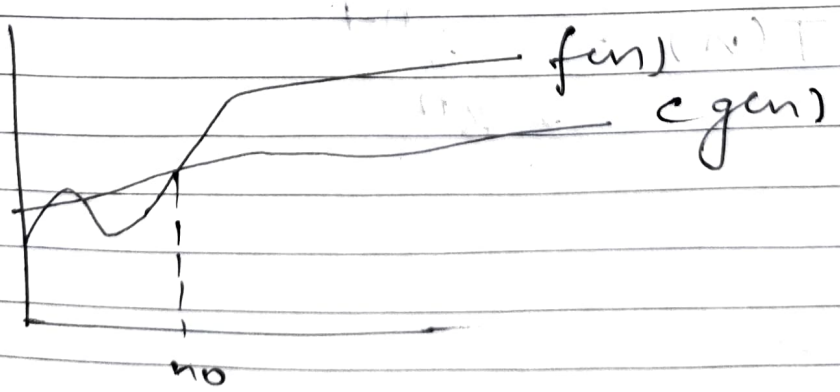
Theta Notation (θ-Notation)

Theta Notation encloses the function from above and below. Since it represents the upper and lower bound of the running time of an algorithm it is used for analyzing the average case complexity of an algorithm



3. Omega Notation (Ω-Notation)

→ represents the lower bound of running time of an algorithm. Thus, it provides the best case complexity of an Algorithm



A2. for (i=1 to n) { i = i * 2; }

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

A3. $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

using forward Substitution

$T(1) = 1$ (given)

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

$T(3) = 3T(2)$
$= 3 \times 3$
$= 9$

$T(4) = 3T(3)$
$= 3 \times 9$
$= 27$

$T(n) = 3^{n-1}$
$= 3^n$

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

$T(1) = 1$

$n = 2$

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

$= 2 - 1$

$= 1$

putting $n = 3$

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

$= 2 - 1$

$= 1$

putting $n = 4$

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

$= 1$

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

Date.
Page No.

A50.

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

After 1st iteration

$s = s+1$

After 2nd iteration

$s = s+1+2$

After m iteration

$s = s+2+2+\cdots m$

$= \frac{m(m+1)}{2} <= n$

$\frac{m^2+m}{2} <= n$

$m^2 <= n$

$m <= \sqrt{n}$

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

A6. void function (int n)
{
int i, count = 0;
for (i = 1; i*i <= n; i++)
count++;
}

A: When i = 1 i*i = 1 <= 5 c = 1
i = 2 i*i = 4 <= 5 c = 2
i = 3 i*i = 9 ≰ 5

$i^2 <= n$
$i <= \sqrt{n}$
$TC = O(\sqrt{n})$

A7. void function (int n)
{
int i, j, k, 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++;
}
}
}
}

Date.
Page No.

Time complexity of innermost loop

$k=1$ to n, $k = k*2$

1, 2, 4, 8, 16 ... k^{th} term

$$k^{th} \text{ term} = 2^{k-1}$$

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

$$2n = 2^k$$

taking log both sides

$$\log 2n = \log 2^k$$

$$\log 2n = k$$

$$\log_2 2 + \log_2 n = k$$

$$k = 1 + \log_2 n$$

Time complexity of middle most loop.

$j=1$ to n, $j = j*2$

1, 2, 4, 8, 16 ... k^{th} term

$= 1 + \log_2 n$

Time complexity of outermost loop

$i = n/2$ to n, $i++$

$n/2$, $\frac{n}{2}+1$, $\frac{n}{2}+2$, ... k^{th} term

$$k^{th} \text{ term} = \frac{n}{2} + k$$

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

$$k = n/2$$

TC $\frac{n}{2} * (1+\log_2 n) * (1+\log_2 n)$

$= O(n \log_2 n)^2$

A8. function (int n)

```
{ if (n==1)                    — T(n)
      return;
   for (i=1 to n)
   {                           → T(n+1)
      for (j=1 to n)
      {  printf("*");
      }
   }
   function(n-3);
}
```

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

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

$TC = O(n^2)$

A9. void function (int n)

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

Date.
Page No.

for $i=1$ loop turns n times
$i=2$ " " $n/2$ times
$i=3$ " " $n/3$ times

$TC = n[n + n/2 + n/3 + \cdots]$
$TC = n[1 + 1/2 + 1/3 + \cdots]$
$TC = n \log n$
$TC = O(n \log n)$.

A10.