Assignment - 1

Q1) Asymptotic notations → These notations are used to tell the complexity of an algorithm when the input is very large.
Asymptotic ~~no~~ means towards infinity.

Different types of Asymptotic notation are :-

(1) Big-On (O) :-

$f(n) = O(g(n))$
$g(n)$ is "tight" upper bound of $f(n)$

$f(n) \leq C \cdot g(n)$
$\forall n \geq n_0, C > 0$

e.g.

$O(n^2 + 3n + 4) \Rightarrow O(n^2 + n)$
$\Rightarrow O(n^2)$

(2) Big-Omega (Ω) :-

$f(n) = \Omega g(n)$
$g(n)$ is "tight" - lower bound of $f(n)$

$f(n) \geq C \cdot g(n)$
$\forall n \geq n_0$ & some constants $C > 0$.

e.g.

$f(n) = 8n^2 + 2n - 3 \geq 8n^2 - 3$

$= 7n^2 + (n^2 - 3) \geq 7n^2 \; (g(n))$

Thus, $k_1 = 7$

(3) Theta (θ) :-

It gives tight upper & lower bound both.

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

if

$C_1 \; g(n) \leq f(n) \leq C_2 \; g(n)$

$\forall n \geq \max(n_1, n_2)$

Some constants C_1 & $C_2 > 0$

e.g.

$3n + 2 = \theta(n)$

as $3n + 2 \geq 3n$

and $3n + 2 \leq 4n$, for n

$k_1 = 3$

$k_2 = 4$

$n_0 = 2$

(4) Small-oh (o) :-

o gives us upper bound.

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

$f(n) < (g(n)) \; \forall \; n > n_0$ & $\forall \; C > 0$

(5) Small-omega (ω):-

ω gives us lower bound.

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

$f(n) \geq C \cdot g(n)$

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

Q2) Time Complexity of

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

Sol)

$i = 1, 2, 4, 8, \ldots\ldots n$

(k steps)

It is G.P.

n^{th} term $= ar^{n-1}$

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

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

$2n = 2^k$

taking $\log_2$ both sides

$\log 2n = k \log_2 2$

$\log_2 2 + \log_2 n = k$

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

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

Using Backward substitution.

Put $n = n-1$ in eq. (1)

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

$T(n-1) = 3T(n-2)$ — (2)

Put value of $T(n-1)$ in eq. (1)

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

$T(n) = 9T(n-2)$ — (3)

Put $n = n-2$ in eq. (1)

$T(n-2) = 3T(n-3)$ — substitute in eq. (3)

$T(n) = 27T(n-3)$

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

$T(0) = 1$

$n-k = 0$

$k = n$

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

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

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

$T(n) = 3^n$

$\therefore T.C. = O(3^n)$.

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

↓ eq. ①

Put $n = n-1$

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

$T(n-1) = 2T(n-2) - 1$ — ②

Put in eq. ①

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

$T(n) = 4T(n-2) - 2 - 1$ — ③

~~$T(n) =$~~

Put $n = n-2$ in eq. ②

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

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

~~$T(n-2) =$~~ $T(n) = 8T(n-3) - 4 - 2 - 1$

~~$T(n)$~~ $T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} \cdots$

$n - k = 0$

$n = k$

$T(n) = 2^n - 2^{n-1} - 2^{n-2} \cdots - 2^{n-k}$

$T(n) = 2^n \left[1 - \frac{1}{2} - \frac{1}{4} \cdots - \frac{1}{n}\right]$

$T(n) = 2^n \times \frac{1((1/2)^n - 1)}{-1/2}$

$T(n) = -2^{n+1} \times \frac{1}{2^n} + 2^n$

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

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

Sol.5)

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

i	s
1	1
2	3
3	6
4	10
5	15
⋮	⋮
k	'

$s = \underbrace{1, 3, 6, 10, \ldots n}_{k \text{ steps}}$

$s = 1, 1+2, 1+2+3, 1+2+3+4, \ldots 1+2+\ldots n$

$n^{th}\text{ term} = \frac{n(n+1)}{2}$

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

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

$\therefore 2n = k^2+k$

$\therefore T(c) = O(n^2).$

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

i = 1, 2, 3, 4 ... = $\sqrt{n}$
i = 1, 4, 9, 16 ... = n

k steps

Let $\sqrt{n} = n$

Series = 1, 2, 3, 4 --- n

A.P.

n^{th} term
$n = 1 + (k-1)1$
$n = 1 + k$
$k = n - 1$
T.C = O(n)

Sol (7) void func (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++
}

k loop :-

$k = 1, 2, 4, 8 \ldots n$

1 × step

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

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

j loop :-

Similar to k loop

$TC = O(\log n)$

i loop :-

$i = \frac{n}{2}, \frac{n}{2}+1, \frac{n}{2}+2 \ldots \frac{n}{2}+n$

$T(c) = O(n)$

$\therefore TC = n(\log(\log n))$

Sol 8)

Time Complexity of -

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

TC of j loop = O(n)
TC of i loop = O(n)

Function calling

n , n-3 , n-6 , n-9 , --- n-(n-

A.P.

$n-(n-1) = n+(k-1)-3$

$n-n+1 = n-3k+3$

$1 = n-3k+3$

$3k = n-2$

$k = \frac{n-2}{3}$

∴ TC = O(n)

∴ Overall TC = $O(n^3)$

Sol.9)

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

i loop

1, 2, 3 --- n

1, 3, 5 --- n

1, 4, 7 --- n

1, 5, 9 --- n

{

$TC = O(n)$

outside i loop = n

$\therefore TC = O(n^2)$

Ans 10) Relation

$O(n^k) < O(a^n)$

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

Let $k=1$ & $a=2$

$O(n) = O(2^{n_0} + C)$

$n = 2^{n_0} + C$

$\log_2 n = n_0 \log_2 2$

$\boxed{n_0 = \log_2 n}$

$C = n - 2^{n_0}$

$C = n - 2^{\log_2 n_0}$

$\boxed{C = n - n_0}$

Ans 11)

```
int j=1, i=0;
while (i<n)
    {i=i+j
    j++ ; }}
```

$i = 0, 1, 3, 6, 10, 15, \ldots$

$j = 1, 2, 3, 4, 5, 6, \ldots$

$i = 0, 1, 1+2, 1+2+3, \ldots$

$0, 1, 2, 3, \ldots$

$$\frac{1+2+\cdots n(n+1)}{2}$$

k steps

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

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

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

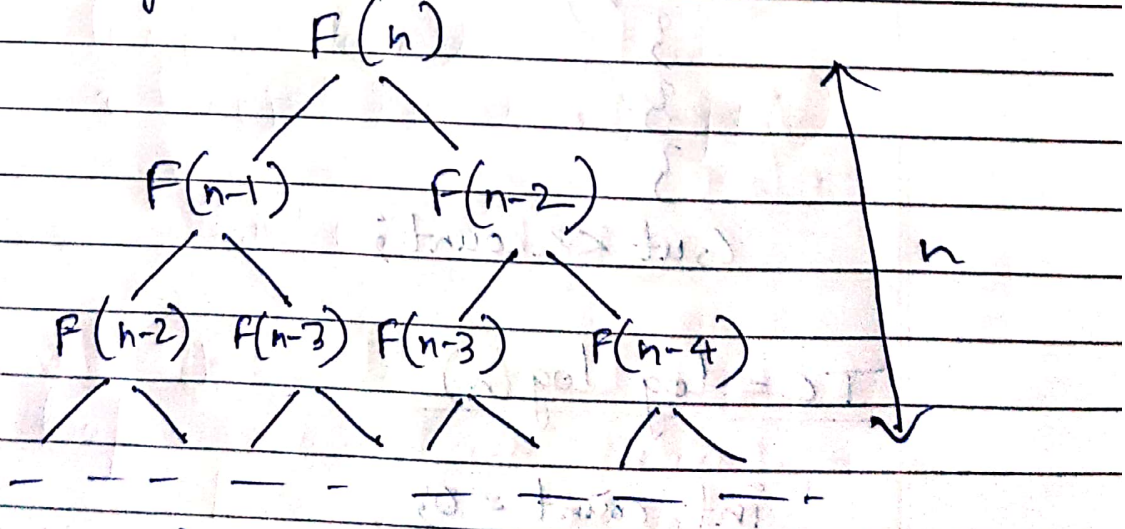
$\therefore TC = O(n^2)$.

Ans 12) $T(n) = T(n-1) + T(n-2)$ — (1)

$T(1) = 1 \quad T(0) = 0$

Put $n = n-1$ in eq. (1)

Using tree method



n height, each height have 2 division

$\therefore TC = O(2^n)$.

Ans 13) TC = n log n

```
for (int i=1; i<=n ; i++)
{
 for (int j=1; j<=n; j=j*2)
  {
   Printf ("Hello");
  }
}
```

TC = n^3

```
int count=0;
for (int i=0; i<n; i++)
{
for (int j=0; j<n; j++)
{
 for ( int k=0; k<n; k++)
   {
    count ++;
   }
}
}
Cout << count;
```

TC = log log (n)

```
int count = 0;
for (int i=1; i<=n ; i=i*2)
{
 for (int j=1; j<=n; j=j*2)
```

```
{
  count++;
  }
}
cout << count;
```

Ans 15)
```
int fun(int n)
{
 for (int i=1; i<=n; i++)
  { for(int j=1; j<n; j=j+1)
   {
     // O(1) task
   }
  }
}
```

$i = 1, 2, 3, \ldots n$

$j = 1, 2, 3 \ldots$

$TC = O(n^2)$.

Ans 16)
```
for(int i=2; i<=n; i= pow(i,k))
  {
     //
  }
```

$i = (2^k)^0, 2^k, (2^k)^k, ((2^k)^k)^k \ldots n$

G.P.

$n = 2.(2^k)^{t-1}$

$\frac{n}{2} = (2^k)^{t-1}$

$$\log n = l-1 \log_2 2^k$$

$$\log n = k(l-1)$$

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

Ans 18) a) 100, $\log \log n$, $\log n$, $\log n!$, $n \log n$, $\sqrt{n}$, n, n^2, $n!$, 2^n, 2^{2n}, 4^n.

b) 1, $\log \log n$, $\sqrt{\log n}$, $\log n$, $\log 2n$, $2 \log n$, $\log n!$, n, $2n$, $4n$, n^2, $n!$, $2(2^n)$

Ans 19) (Linear Search)

```
Int key = input
for (i = 1 to n)
  if arr[i] = key
    Print ("found");
    Exit;
else
    i = i + 1
```

Ans 20) [Insertion Sort]

```
for (i = 1 to array length - 1)
  current = array[i]
  pos = i
while Pos > 0 and Array[pos-1] current
    Array[pos] = Array[pos-1]
    Pos = Position - 1
```

END while
Array [Position] = Current
i = i + 1

Recursive insertion sort

```
if (n<=1)
  Return
Call Insertion Sort (arr, n-1)
  current = arr[n-1]
  Pos = i - 2
  while (Pos >= 0 . AND arr[Pos] > current
      arr[Pos-1] = arr[Pos]
      Pos = Pos - 1
  }
  arr[Pos+1] = current
}
```

Insertion sort is online as we can change the size of array after inserting new values that also the algo will work. Whereas Bubble sort & Insertion sort fails here.

Ans24 Bubble Sort:

Best = $\Omega(n)$
Avg. = $\Theta(n^2)$
Best = $O(n^2)$

Ans 21) Selection sort → $O(n^2)$
Insertion " → $O(n^2)$
Quick " → $\Omega(n \log n)$

Ans 22)

Algo	Stable	Place	Online
Bubble	✓	Inplace	X
Insertion	✓	Inplace	✓
Selection	X	Inplace	X
Quick	X	Outplace	✓
Merge	✓	Outplace	X

Ans 23) Binary Search (Iterative)

```
A ← Sorted array
n ← Size of array
x ← Value to be searches

lowerbound = 0
upperbound = n-1
while lowerbound <= upperbound
  mid = (lowerbound + upperbound) / 2

if A[mid] = x
      Print(found)
      exit
if A[mid] < x
      lowerbound = mid + 1
else
```

upperbound = mid - 1
end while

Recursive

A → Sorted array
n → Size
L → lower bound
U → Upper bound
X → Element to be sarched

Function Binary Search (A, L, U, X)

while L <= U
mid = (L + U)/2
If X == A[mid]
Return mid
else if X > A[mid]
Return Binary Search (A, Mid+1, U, X);
else
Return Binary - Search (A, L, mid-1, X);
End while
End of function

Ans24) Binary Search Recurrence Relation
$T(n) = T(n/2) + 1$