ASSIGNMENT

Ans 1 Asymptotic notations are languages that allow us to analyze an algorithm's running time by identifying its behaviour as the input size of algorithm.

Types :-

(a) Big O : It is commonly used for worst case, and gives us upper bound for the growth rate of runtime of algorithm.

Example : Big O notation for linear search is $O(n)$.

(b) Big Omega : It is notation used for best case complexity, it provides us with an asymptotic lower bound.

Example : Big Omega of linear search is $\Omega(1)$.

(c). Theta : It is used for tight bound on the growth rate of runtime of an algo.

Example : Theta of linear search is $\theta(n)$

(d) Small o : It is used to denote the upper bound (i.e. not asymptotically tight).

$$f(n) = o(g(n)) \quad \forall \quad f(n) < c(g(n))$$
$$c > 0$$

(e) Small omega : To denote lower bound (that is not asymptotically tight).

Ans 2

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

Time Complexity — $O(\log n)$

Ans 3

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

$T(1) = 1$

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

$T(3) = 3T(\cancel{n}-2) = 9$

$T(4) = 3T(3) = 27$

$\vdots$

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

Time Complexity $\rightarrow$ $O(3^n)$

Ans 4

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

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

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

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

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

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

$T(n) = 16T(n-4)-8-4-2-1$

$T(n) = 2^k \text{ - - - - - - } 2^3 - 2^2 - 2^1 - 2^0$

Time Complexity $\rightarrow$ $O(1)$

Ans 5

S	i
1	1
3	2
6	3
10	4

Time Complexity − $O(\sqrt{n})$

Ans 6

$i * i = n$

$i^2 = n$

$i = \sqrt{n}$

Time Complexity − $O(\sqrt{n})$

Ans 7

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

Ans 8

Ans 9 Total $TC = O(n \log n)$

Ans 10 n^k is $O(c^n)$ as for example

If we take $n = 2$ $K = 2$, $c = 2$

Then $2^2 \leqslant 2^2$

So, c^n is upper limit of n^k.

Ans 11

$j = 1$	$i = 0$
1	1
2	3
3	6
4	10

The series of i is nearly dependent on i as 2^i

So, Time Complexity $= O(2^n)$

Ans 12

Space complexity $= O(n)$ as clear call of $f(n-1)$

$f(n) \longrightarrow 1$

$f(n-1)$ $\quad$ $f(n-2) \longrightarrow 2$

$f(n-2)$ $\quad$ $f(n-2)$ $\quad$ $f(n-3)$ $\quad$ $f(n-4) \longrightarrow 2^2$

$\equiv$

2^n

Time Complexity $= O(2^n)$

Ans 13

* $n \log n$

```
for (i=0 ; i<n ; i++)
    for( j=0 ; j<n ; j= j*2)
        c++ ;
```

* n^3

```
for (i=0; i<n; i++)
    for( j=0; j<n; j++)
        for( k=0 ; k<n; k++)
            c++
```

* log (log n)

```
int func ( int n
{ if (n = =1)
      return n ;
  else
  return func (√n )+ func (√n ) ;
}
```

Ans 14

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + Cn^2$$

using master method →

$a=2$, $b=2$

$c=1$

if $f(n) > n^a$ $\quad f(n^2) > 1$

Time Complexity – $O(n^2)$

Ans 15 $O(n\sqrt{n})$

Ans 16 $O(\log \log n)$

Ans 17 $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right)$

$f(n)$

$f\left(\frac{99n}{100}\right)$ $\quad$ $f\left(\frac{n}{100}\right)$

$f\left(\frac{99n \times 99}{100^2}\right)$ $\quad$ $f\left(\frac{99n}{100}\right)$

Time Complexity – $O(\log n)$

Ans 18 a) $100 < \log(\log(n)) < \log(n) < \sqrt{n} < n < \log(n!) <$ ~~$n \log n <$~~ $n \log n < n^2 < 2^n < 2^{2n} < 4^n <$ ~~2~~ $n!$

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

c) $96 < \log_2 n < \log_8 n < \log 5n < \log n! < n \log_6(n) < n \log_2(n) < 8n^2 < 7n^3 < 8^{2n} < n!$

Ans 19

```
linear (arr, key)
{
 for(int i=0; i<n ; i++)
  if (arr [i]}== key)
     return i;
return -1;
}
```

Ans 20

```
Insertion (arr, n)
{
  if (n<=1) return;
 recursing for n-1 element
     Insertion sor(arr, n-1);

Pick last element arr[i] and inserts i into sorted sequence.
}
```

Iteration

```
Insertion (arr, n)
{
 for(i=1 to i<n; i++)
{
 Pick arr[i] & insert into arr [0, ------, i-1]
}
```

	Stable	Inplace	Online
Bubble Sort	✓	✓	×
Selection Sort	×	✓	×
Insertion sort	✓	✓	✓

Ans 21

	Best	Avg	Worst	Space Complexity
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	1
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	1
Insertion Sort	$O(n)$	$O(n)$	$O(n^2)$	1

Ans 23 Recursive

```
Binary (arr, l, h, key)
{
 if( l<r)
 {
  mid = l+(r-l)/2 ;
  if (arr[mid] == key)
    return 1 ;
  if (key < arr[mid])
    Binary( l, mid-1, key);
  else
    Binary (mid +1, r, key);
 }
}
```

Iterative

```
~~wit~~ while (l<r)
{
  mid= l+(r-1)/2
 if (arr [mid] == key) return 1 ;
 ~~x~~ if ( key < arr[mid] )
      r= mid -1
 else
      l= mid + 1;
}
```

Ans 24

$$T(n) = T\left(\frac{n}{2}\right) + 1$$