Assignment

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

Types →

① 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 $O(n)$;

② ~~Notation~~ Big Omega: It is notation used for least case complexity, it provides us with an asymptotic lower bound.

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

③ Theta: It is used for tight bound on the growth rate of runtime of algo.

Ex: Theta of linear search is $\Theta(n)$.

④ Small o → It is used to denote the upper bound (i.e not asymptotically tight).

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

⑤ Small omega: To denote lower bound (that is not asymptotic tight)

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

$\Rightarrow O(\log n)$

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

$T(1) = 1$

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

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

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

$\vdots$

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

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

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

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

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

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

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

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

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

$T(n) = 2^k - 1 \ldots - 2^3 - 2^2 - 2^1 - 2^0$

$= O(1)$

Ans 5.

S	i
1	1
3	2
6	3
10	4

$O(\sqrt{n})$

Ans 6- $i * i = n$

$i^2 = n$

$i = \sqrt{n}$

$O(\sqrt{n})$

Ans 7- $O(n \log^2 n)$

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 $2^2 \leq 2^2$ so c^n is upper limit of n^k.

Ans 11.

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

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

so $O(2^n)$

Ans 12. Space complexity $= O(n)$ as clear call of $f(n-1)$!

$f(n) \longrightarrow 1$

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

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

$\vdots$

2^n

time complexity $= O(2^n)$

Ans13- nlogn

```
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(logn)

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

Ans14. $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + cn^2$

using mast →

$a=2$, $b=2$

$c = 1$

$f(n) > n^a$ & $n^2 > 1$

$O(n^2)$

Ans15. $O(n\sqrt{n})$

An16. $O(\log\log n)$

Ans17. $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right)$

$f(n)$

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

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

$O(\log n)$

Ans18. a) $100 < \log\log n < \log n < \sqrt{n} < n \log(n!) <$
$n\log n < n^2 < 2^n < 2^{2^n} < 4^n < n!$

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

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

Ans19. linear (arr, Key) {
for (int i=0 ; i<n ; i++)
if (arr[i] == Key)
return i;

return -1;
}

Ans20. Insn (arr, n) {
if (n<=1) return;
recursively sor n-1 element
Insert sort (arr, n-1);
Pick las element arr[i] & insert it into
sorted sequence

}

Iteration

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

	stable	Inplace	Online
		✓	X
Bubble sort	✓	✓	X
Select "	X	✓	✓
Insert "	✓		

Ans 2-

	Best	Avg	Worst	space complexity
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	1
Select	$O(n^2)$	$O(n^2)$	$O(n^2)$	1
Insert	$O(n)$	$O(n^2)$	$O(n^2)$	1

Ans 23- Recursion

```
Binary (arr, l, r, 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))
  }
}
```

Iteration

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

Ans 24. $T(n) = T\left(\frac{n}{2}\right) + 1$

Ans 1: $K =$