Ans2

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

values of i: 1, 2, 4, 8, ... 2^k

which forms a G.P

$a=1, r=2$

$\therefore t_k = a \cdot r^{k-1}$

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

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

$2n = 2^k$

$k \cdot \log_2 2 = \log_2(2n)$

$k \cdot \log_2 2 = \log_2(2) + \log_2(n)$

$k = 1 + \log_2(n)$

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

$T(0) = 1$

Using forward substitution,

$T(1) = 3T(0) = 3$

$T(2) = 3T(1) = 3\times3 = 9$

$T(3) = 3T(2) = 3\times9 = 27$

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

$= O(3^n)$

Ans4 $T(n) = \{2T(n-1)-1$ if $n>0$, otherwise $1\}$

$T(0) = 1$

Using backward substitution,

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

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

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

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

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

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

$= 2^k \cdot T(n-k) - \sum_{i=0}^{k-1}(2^i)$

Put $k=n$

$\therefore T(n) = 2^n \cdot T(0) - \sum_{i=0}^{n-1}(2^i)$

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

$= 2^n - 2^n + 1$

$= 1$

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

Ans5)
```
i=1, s=1;
while(s<=n){
    i++; s+=i;
    printf("#");
}
```

Values of i and s will change as :-

i 1 2 3 4 5 . . .

×11 ×11

s 1 3 6 10 15 . . .

$\therefore S_i = S_{i-1} + i$
where $i = 1, 2, 3, \ldots K$

Sum of A.P is $\frac{n(n+1)}{2} = \frac{K(K+1)}{2}$

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

Ans 6

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

values of i will be: $1, 2, \ldots, K$
where $K = \sqrt{n}$
$\therefore T(n) = O(\sqrt{n})$

Ans 7)

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

values of i will change as: $n/2, \left(\frac{n}{2}\right)+1, \left(\frac{n}{2}\right)+2, \ldots n$

$= \left(\frac{n}{2}\right) + 1$ times $\Rightarrow O(n)$

values of j and k change as:
1, 2, 4, 8, 16, ...
Time complexity of these loops will be
$O(\log n)$

$\therefore T(n) = O(\log n) \cdot O(\log n) \cdot O(n)$
$= O(n \cdot (\log n)^2)$

Ans8)
```
function (int n){
    if(n==1) return;
    for(i=1 to n){
        for(j=1 to n)
            printf("*");
    }
    function(n-3);
}
```
values of n will be: n, n-3, n-6, ..., 1
$\Rightarrow O(n)$
complexity of i loop = $O(n)$
complexity of j loop = $O(n)$
$\therefore T(n) = O(n^3)$.

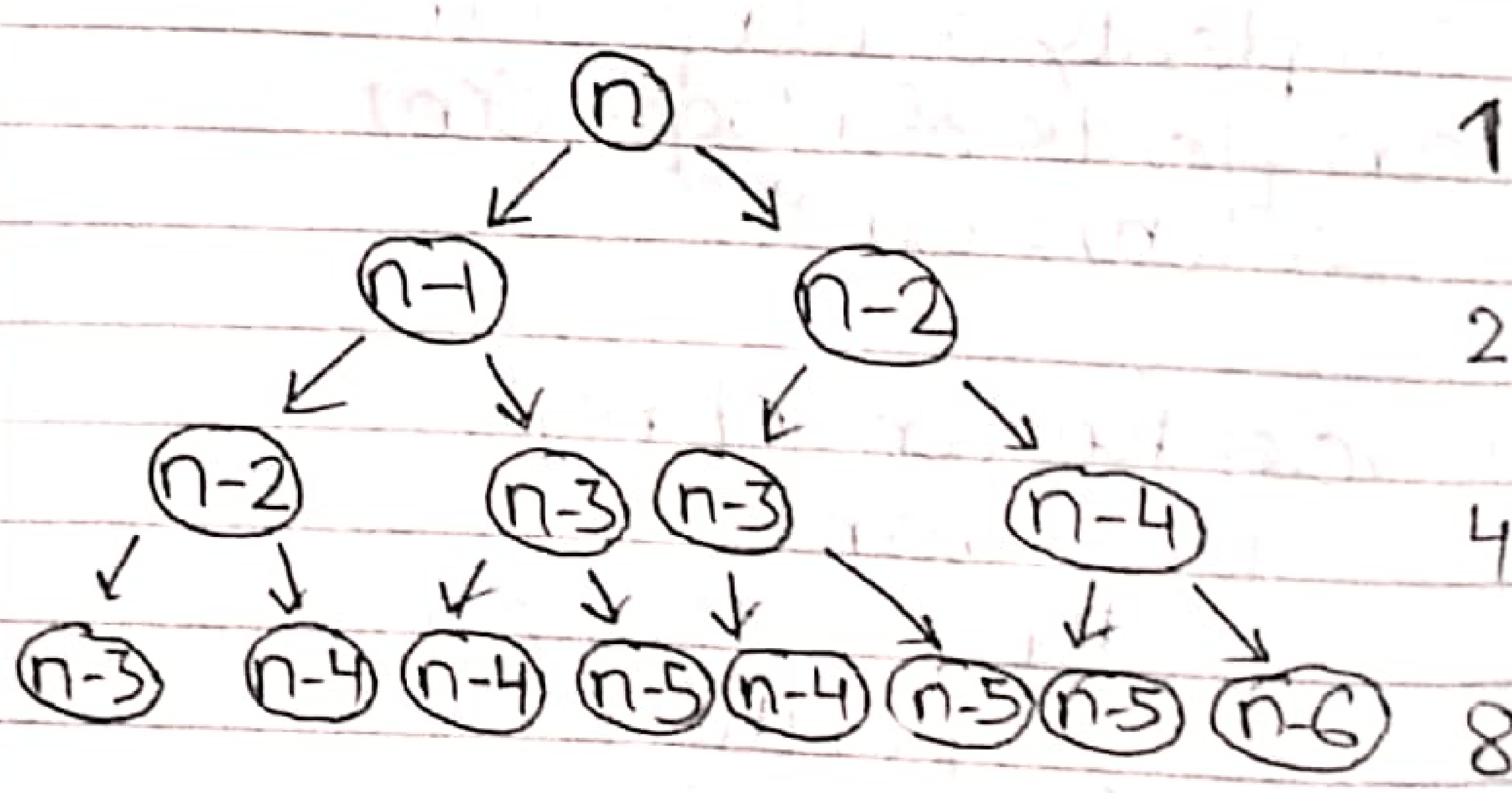
Ans9)
```
void function(int n){
    for(i=1 to n){
        for(j=1; j<=n; j+=i)
            printf("*");
    }
}
```

complexity of loop $i = O(n)$
complexity of loop $j = O(\sqrt{n})$

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

Ans 11)
```
void fun(int n) {
    int j=1, i=0;
    while(i<n){
        i+=j;
        j++;
    }
}
```
complexity $\Rightarrow O(\sqrt{n})$

Ans 12)
```
int fib(int n){
    if(n<=1) return n;
    return fib(n-1) + fib(n-2);
}
```
$T(n) = T(n-1) + T(n-2) + 1$



$T(n) = 1 + 2 + 4 + \ldots + 2^n$ $\quad 2^n$

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

$$\Rightarrow O(2^n)$$

Ans 13) I) $n(\log n)$

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

II) n^3

```
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++){
        for(int k=1; k<=n; k++)
            count++;
    }
}
```

III $\log(\log n)$

```
for(int i=n; i>=1; i=√i){
    count++;
}
```

Ans1) # Asymptotic Notations :-

Asymptotic → towards infinity

* while defining the complexity of our algorithms, we will use asymptotic notati assuming our input size is very large

* Time Complexity	Space Complexity
Number of instruction to carry out an algorithm.	Extra space required by an algorithm except input.

1) Big-oh (O):

$f(n) = O(g(n))$

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

$\therefore f(n) <= c\,g(n)$

$\forall\ n \geq n_0$, some constant $c > 0$

* while calculating complexities :

- constants are ignored
- Lower order terms are ignored in addition or subtraction ∴ Take the highest order term

2) Big Omega (Ω):

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

g(n) is "tight" lower bound of f(n)

$\therefore f(n) \geq c.g(n)$

$\forall\ n \geq n_0$ and $c>0$

3) Theta (Θ):

Theta gives the "tight" upper & lower bound both.

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

if $c_1.g(n) \leq f(n) \leq c_2 g(n)$

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

for some constant c_1 & $c_2 > 0$

$\Rightarrow f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$

4) Small-oh (o):

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

$f(n) < c\ g(n)$

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

5) Small-omega (ω):

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

$f(n) > c\ g(n)$

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

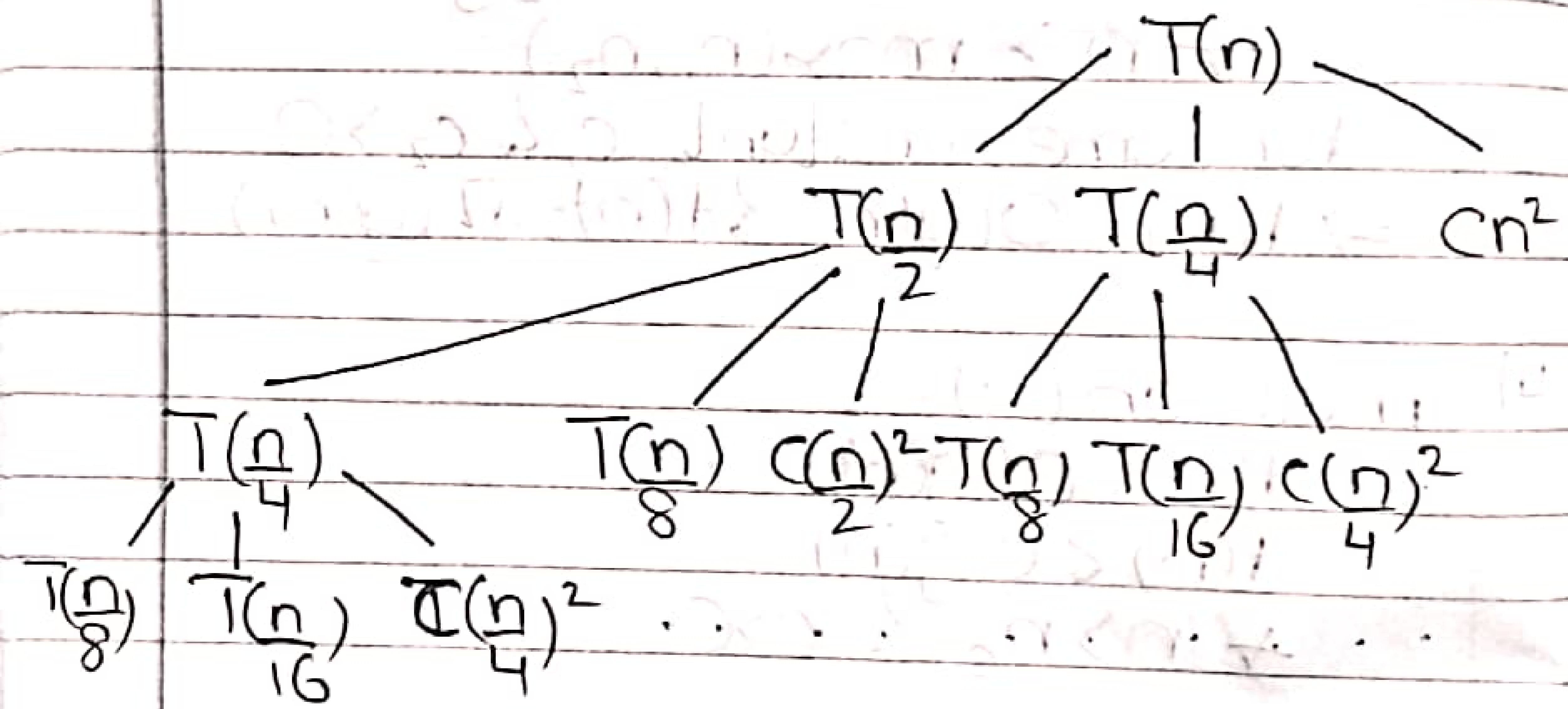
Notes:-

* $f(n) = O(g(n)) \rightarrow g(n) = \Omega(f(n))$
* $f(n) = o(g(n)) \rightarrow g(n) = \omega(f(n))$
* $f(n) = \Theta(g(n)) = O(g(n))$ & $f(n) = \Omega(g(n))$

	Reflexive	Symmetric	Transitive
O	✓	✗	✓
Ω	✓	✗	✓
Θ	✓	✓	✓
o	✗	✗	✓
ω	✗	✗	✓

Ans 14) $T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + cn^2$

Recursion Tree



No. of nodes per level will be 1, 3, 6, 12 ...

$\therefore$ Total no. of nodes $= \frac{1+3(2^{n/2}-1)}{(2-1)}$

$= 1 + 3 \cdot (2^{n/2} - 1)$

$= 2^n$

$= O(2^n)$

Ans 15) $O(\sqrt{n})$, same as ans 5 & 11

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

values of i will be as follow :

$2, 2^{k}, 2^{k^2}, 2^{k^3}, \ldots, 2^{k^c}$

As $2^{k^c} <= n$

$k^c \cdot \log_2 2 = \log_2 n$

$k^c = \log n$

$c \cdot \log_k k = \log_k - \log(n)$

$c = \log_k (\log n)$

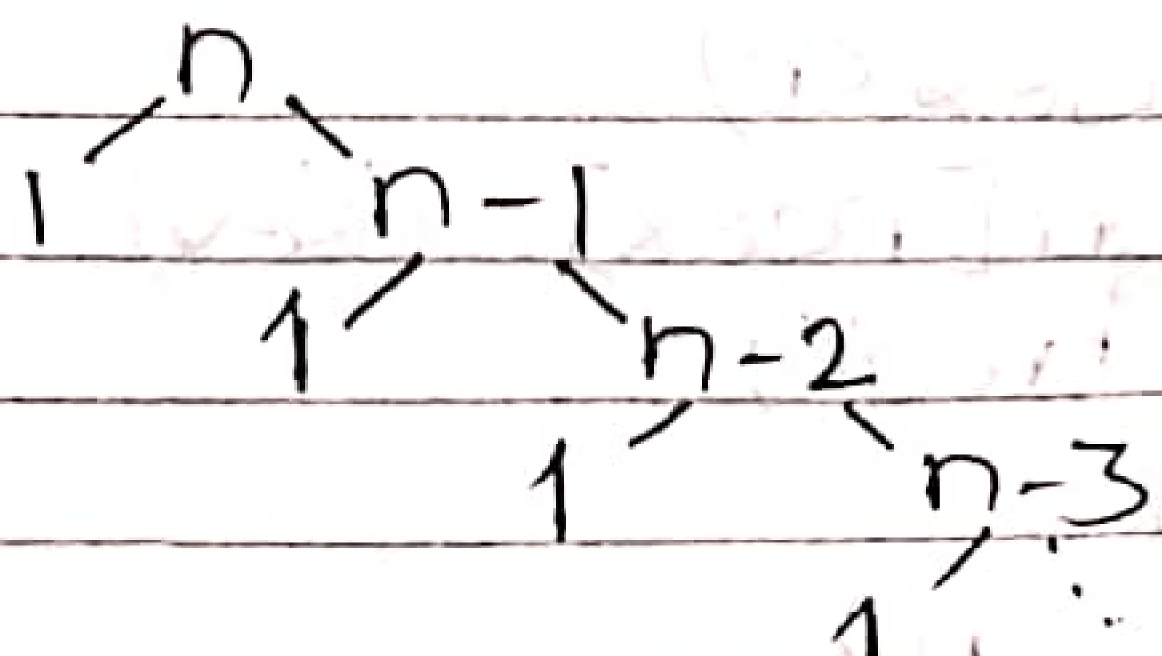
$\therefore T(n) = O(\log_k (\log n))$

Ans 17) Quick sort will repeatedly divide the array into two parts of 99% and 1%. i.e, when the pivot chosen by the partition is always either the smallest or the largest element in the array.

$\therefore T(n) = T(n-1) + 1$

$T(1) = 1$

Recursion Tree :



Using forward substitution :

$T(1) = 1$
$T(2) = 2$
$T(3) = 3$
$\vdots$
$T(n) = n$

$\therefore$ Time complexity $\Rightarrow 1+2+3+ \dots +n$

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

$$= O(n^2)$$

Ans 18) a) $100 < \log(\log n) < \text{root}(n) = \log(n) < n < \log(n!) < n\log(n) < 2^n = n^2 < 2^{2n} = 4^n < n!$

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

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

Ans 19)
```
index = 0
while(index<n)
        if(arr[index]==key)
            break;
        index;
if(index==n)
    // not found
else
    // found
```

Ans 20) Iterative Insertion sort:-

```
void insertion_sort(int[],arr, int n){
  for(int i=0; i<n; i++){
      int temp = arr[i];
      int j = i-1;
      while (j>=0 && arr[j]>temp){
          arr[j+1] = arr[j];
          j--)
      }
      arr[j+1] = temp;
   }
}
```

Recursive Insertion sort:-

```
void insertion_sort(int arr[], int n){
    if (n<=1)
       return;
    insertion_sort(arr, n-1);
    int last = arr[n-1];
    int j = n-2;
    while (j>=0 && arr[j]>last){
         arr[j+1] = arr[j];
         j--;
    }
    arr[j+1] = last;
}
```

Insertion sort is called online sorting because we can add new elements to the array's end while the sorting is being executed. At any given iteration, say iteration 'K', only first K elements of array participate in sorting. Therefore we can add new elements to the array during the sort.
No, other algorithms discussed in class are not online sorting.

Ans 21) Complexity of all algorithms discussed in class:-

	Best	Average	Worst
• Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
• Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
• Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
• Merge	$O(n\log n)$	$O(n.\log n)$	$O(n.\log n)$

Ans 22)
- Bubble sort : inplace, stable, offline
- Selection sort : inplace, unstable, offline
- Insertion sort: inplace, stable, offline

Ans 23,24) Recursive Binary Search:-

```
int binary_search(int* arr, int l, int r){
    if(l<=r){
        m= l+(r-l)/2;
        if(arr[m] == key)
            return m;
```

```
else if(arr[m] < Key)
    return binary_search(arr, l, m-
}
}
```

- Recurrence relation for recursive binary search:-
  $T(n) = T(n/2) + 1$
- Time complexity of recursive binary search:
  $T(n) = O(\log n)$
- Time complexity of iterative binary search
  $T(n) = O(\log n)$
- Time complexity of Linear search:
  $T(n) = O(n)$
- Space complexity in all cases $= O(1)$