Date: / /

ASSIGNMENT-1

Ujjwal Panwar ML 87

Ans 2

```
for (i=0; i<=n; i=i*2){
    //some code
}
```

Values of i : 1, 2, 4, 8, ---- 2^k

This is a geometric progression.

$a = 1$, $r = 2$

$\therefore t_k = ar^{k-1}$

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

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

$2n = 2^k$

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

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

$k = 1 + \log_2 (n)$

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

Date: / /

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

$T(0) = 1$

Using forward substitution,

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

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

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

$T(n) = 3^n$

$= O(3^n)$

Ans 4 $T(n) = \{2T(n-1)-1$ if $n>0$, else $1\}$

$T(0) = 1$

Using backward substitution,

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

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

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

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

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

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

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

Put $k = n$

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

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

Date: / /

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

$= 1$

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

Ans 5

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

values of s and i changes as follows :

i	1	2	3	4	5
s	1	3	6	10	16

$\therefore s_i = s_{i-1} + i$

where $i = 1, 2, 3, \ldots k$

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

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

Date: / /

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, ---, 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 : $n/2, (n/2)+1, \frac{n}{2}+2, \ldots n$

$\Rightarrow (\frac{n}{2}+1)$ times $\approx O(n)$

Date: / /

Values of j and k change as :
1, 2, 4, 8, 16, ...

Time complexity of these loops will be $O(\log n)$

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

Ans 1

```
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 : $n, n-3, n-6, \ldots, 1$
$\Rightarrow O(n)$

Complexity of i loop $\Rightarrow O(n)$
Complexity of j loop $\Rightarrow O(n)$

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

Date: / /

Ans 9

```
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++;
    }
}
```

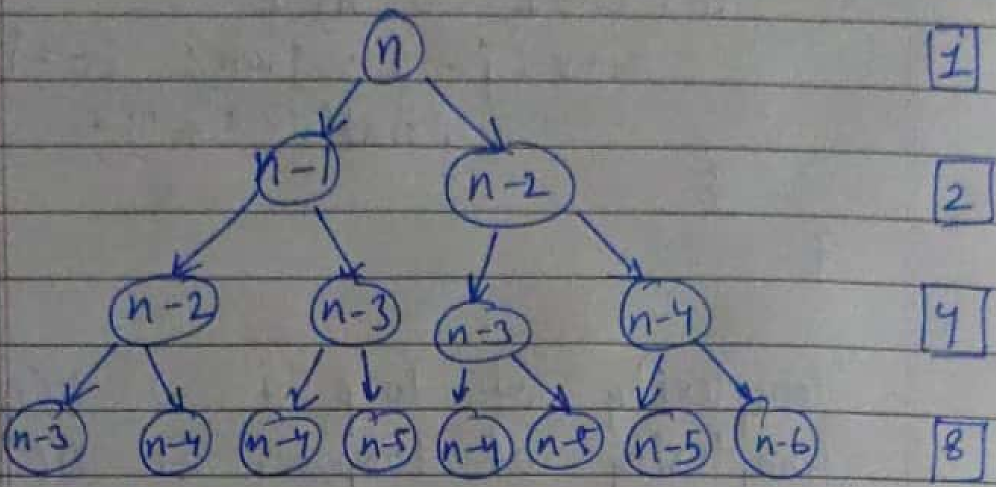
Complexity => $O(\sqrt{n})$

Ans 12

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

Date: / /

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



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

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

$$= O(2^n)$$

Ans 13 I) n (logn)

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

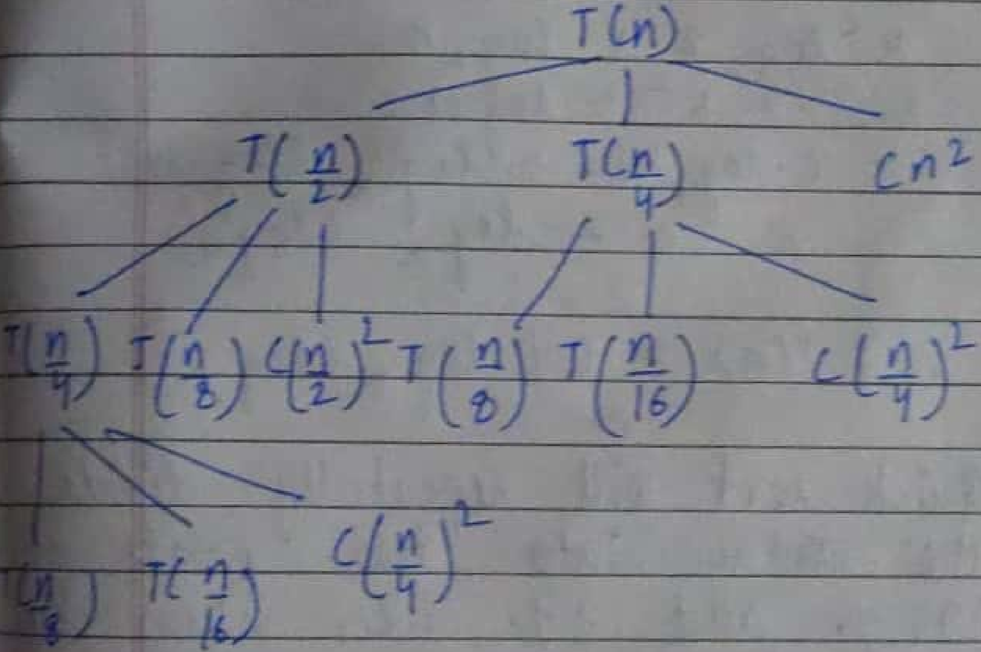
Date: / /

(III) log(log n)

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

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

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

$= 2^n$

$= O(2^n)$

Date: / /

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

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

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

$k^c = \log n$

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

$c = \log_k (\log n)$

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

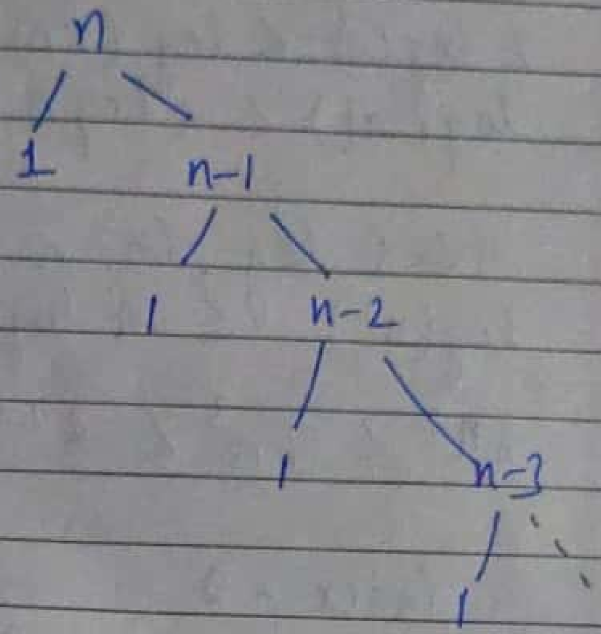
Ans 17 Quick sort will repeatedly divide the array into two parts of 99% and 1%, i.e, when the pivot chosen by the partition is always either the smallest or the largest element is in the array.

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

$T(1) = 1$

Date: / /

Recursion Tree :



Using forward substitution :

$T(1) = 1$

$T(2) = 2$

$T(3) = 3$

$\vdots$

$T(n) = n$

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

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

$\Rightarrow O(n^2)$

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

Date: / /

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

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

Interative 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 lost = 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 ~~sorting~~ array during the sort.

Date: / /

No, other algorithms discussed in class are not online sorting.

Ans 21 Complexity of all algorithms discussed in class :-

	Best	Avg	Worst
• Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
• ~~Sorting~~ Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
• Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
• Merge	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$

Ans 22
- Bubble sort : inplace, stable, offline
- Selection sort : inplace, unstable, offline
- Instration sort : inplace, stable, online
- Merge sort : not 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, m+1, r);
      else
            return binary-search (arr, l, m-1);
    }
}
```

Date: / /

- Recurrence relation for recursive binary search :-

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

- 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 = $O(1)$

Date: / /

Ans 1 Asymptotic Notations :-

Asymptotic → towards infinity

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

*

Time complexity	Space complexity
Number of instructions to carry out an algorithm.	Extra space required by an algorithm except input.

i) 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 substraction.

$\therefore$ Take the highest order term.

Date: / /

2) Big Omega (Ω):

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

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

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

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

3) Theta (θ):

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

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

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

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

for some constant c_1 & $c_2 > 0$

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

4) Small-oh (o):

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

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

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

5) Small-omega (ω):

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

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

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

Date: / /

Notes :-

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

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

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