Tutorial-1 ①
DAA

Name – Ayushman Bhatt
Section – AI & DS
Class Roll No – 21
University Roll No – ~~2002172~~ 2017640

Answer 1 **Asymptotic Notation**: Asymptotic Notations are the mathematical notations used to describe the running time of an algorithm.

Different types of Asymptotic Notation :-

① **Big-O Notation (O)** – It represents upper bound of algorithm.

$$f(n) = O(g(n)) \text{ if } f(n) \leq c * g(n)$$

② **Omega Notation (Ω)** – It represents lower bound of algorithm.

$$f(n) = \Omega(g(n)) \text{ if } f(n) \geq C * g(n)$$

③ **Theta Notation (Θ)** – It represents upper and lower bound, that is average bound of algorithm.

$$f(n) = \Theta(g(n)) \text{ if } C_1 * (g(n)) \leq f(n) \leq C_2 * (g(n))$$

Answer 2

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

i = 1
i = 2
~~i = 3~~
i = 4
i = 8
i = 16
⋮
i = n

It is forming a GP.

$a_n = ar^{n-1}$

$n = ar^{k-1}$

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

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

$\log n = (k-1)\log 2$

$\boxed{k = \log n + 1}$

$\therefore$ Time Complexity $= O(\log n)$

Answer 3 $T(n) = 3T(n-1)$ if $n > 0$, otherwise 1

$T(1) = 3T(0)$ $\quad [T(0) = 1]$

$T(1) = 3 \times 1$

$T(2) = 3T(1) = 3 \times 3 \times 1$

$T(3) = 3 \times T(2) = 3 \times 3 \times 3$

$\vdots$

$T(n) = 3 \times 3 \times 3 \times \ldots$ n times

$= 3^n = O(3^n)$

Hence, time complexity $= O(3^n)$

Answer 4 $T(n) = 2T(n-1) - 1$ if $n \geq 0$, otherwise 1

$T(0) = 1$

$T(1) = 2T(0) - 1$

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

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

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

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

$= 2 - 1 = 1$

$\vdots$

$T(n) = 1$ Time Complexity $= O(1)$

Answer 5

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

$i = 1$ $S = 1$

$i = 2$ $S = 1+2$

$i = 3$ $S = 1+2+3$

$i = 4$ $S = 1+2+3+4$

$\vdots$

Loop ends when $s > n$

$1+2+3+4+\ldots+k > n$

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

$k^2 > n$

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

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

Answer 6

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

$i=1$
$i=2$
$i=3$
$i=4$
$\vdots$
$i=k$

Loop ends when $i*i>n$

$k*k>n$

$k^2>n$

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

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

Answer 7

```
void function (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++;
}
```

- 1st Loop → $i=n/2$ to n, i++
  $= O(n/2) = O(n)$
- 2nd Loop (Nested) → $j=1$ to n, $j=j*2$
  $j=1$
  $j=2$
  $j=4$
  $\vdots$
  $j=n$
  $= O(\log n)$
- 3rd Loop (Nested) → $k=1$ to n, $k=k*2$
  $k=1$
  $k=2$
  $k=4$
  $\vdots$
  $k=n$
  $= O(\log n)$

Total Time Complexity $= O(n \times \log n \times \log n) = O(n\log^2 n)$

Answer 8

```
function(int n)
{ if (n==1) return; —— 1
  for(int i=1 to n)
  { for(int j=1 to n) —— n²
    { printf("*"); }
  } function(n-3); —— T(n-3)
}
```

Recurrence Relation

$\boxed{T(n) = T(n-3) + n^2}$

$\therefore \rightarrow T(1) = 1$

$\rightarrow T(4) = T(4-3) + 4^2 = T(1) + 4^2 = 1^2 + 4^2$

$\rightarrow T(7) = T(7-3) + 7^2 = 1^2 + 4^2 + 7^2$

$\rightarrow T(10) = T(10-3) + 10^2 = 1^2 + 4^2 + 7^2 + 10^2$

So, $T(n) = 1^2 + 4^2 + 7^2 + 10^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6} = O(n^3)$

also for terms like $T(2), T(3), T(5)$

So, Time Complexity $= O(n^3)$

Answer 9

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

$i=1 \rightarrow j=1$ to n
$i=2 \rightarrow j=1$ to n
$i=3 \rightarrow j=1$ to n
$i=4 \rightarrow j=1$ to n

$\therefore$ So, for i upto n it will take n^2

So, $T(n) = O(n^2)$

Time Complexity $= O(n^2)$

Answer 10 $f_1(n) = n^k$, $f_2(n) = C^n$

Asymptotic relationship between f_1 and f_2 is Big-O.

i.e., $f_1(n) = O(f_2(n)) = O(C^n)$

as $n^k \leq G * C^n$ [G is some constant]