Shubham Gupta
F (28)
2017050

Tutorial-1 (DAA)

Ans 1 Asymptotic Notations: These notations are used to tell the complexity of an algorithm with respect to the input size

Different types of notations:

1. Big-O Notation (O): It represents upper bound of an algorithm

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

2. Omega Notation (Ω): It represents lower bond of an algorithm

$$f(n) = \Omega(g(n)) \text{ iff } f(n) \geq c * g(n)$$

3. Theta Notation (θ): It represents upper & lower bond of an algorithm

$$f(n) = \theta(g(n)) \text{ iff } f(n) \geq c_2 g(n) \text{ \& } f(n) \leq c_1 g(n)$$

Ans 2

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

$i=1$
$i=2$
$i=4$
$i=8$
$\vdots$
$i=n$

It is forming a GP

$$a_n = ar^{n-1}$$
$$n = ar^{k-1} = 1 \times (2)^{k-1}$$
$$\log n = \log 2^{k-1}$$
$$\log n = (k-1)\log 2$$
$$k = \log n + 1$$
$$= O(\log n)$$

$$\left(\begin{matrix} a_n = n \\ r = 2 \\ a = 1 \end{matrix} \right)$$

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

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

$T(1) = 3\times 1$

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

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

⋮

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

Ans-4 $T(n) = 2T(n-1)-1$ if $n>0$, otherwise 1

$T(0)=1$

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

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

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

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

⋮

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

Ans-5

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

$i=1$ $s=1$

$i=2$ $s=1+2$

$i=3$ $s=1+2+3$

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

Loop ends when $s > n$

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

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

$$k^2 > n$$

$$k > \sqrt{n}$$

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

Ans-6

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

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

Loop ends when $i * i > n$

$$k * k > n$$

$$k^2 > n$$

$$k > \sqrt{n}$$

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

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 = 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 Nested loop: $j = 1$ to n, $j = j * 2$

$j = 1$
$j = 2$
$j = 4$
$\vdots$
$j = n$

$= O(\log n)$

- 3rd Nested loop: $k = 1$ to n, $k = k * 2$

$k = 1$
$k = 2$
$k = 4$

$= O(\log n)$

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

Ans-8

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

$T(n) = T(n-3) + n^2$ $\quad T(1) = 1$

$T(1) = 1$

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

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

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

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

<u>Ans-9</u> void function (int n)

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

i=1 — j=1 to n
i=2 — j=1 to n
i=3 — j=1 to n
i=4 — j=1 to n

So, for i upto n it will take n^2

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

<u>Ans-10</u> $f(n) = n^k$, $f2(n) = c^n$

$k >= 1$, $c > 1$

Asymptotic relationship between f1 & f2 is Big O

i.e $f_1(n) = O(f2(n)) = O(c^n)$

$n^k \leq a * c^n$ [a is some const]