DESIGN AND ANALYSIS OF ALGORITHMS.

TUTORIAL-1

(11/03/2022)

Answer 1: Asymptotic notations are used to find the complexity of an algorithm when input is very large.

- Big Oh (O): $f(n) = O(g(n))$

  iff

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

  for all $n > k$ and

  for some constant $c > 0$

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

- Big Omega (Ω):

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

  iff

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

  $\forall \; n \ge k$

  for some constant $c > 0$.

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

- Big Theta (θ):

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

  iff

  $c_1 g(n) \le f(n) \le c_2 g(n)$

  $\forall \; n \ge \max(n_1, n_2)$

  for some constant $c_1 > 0$ and $c_2 > 0$

  $g(n)$ is both "tight upper bound" and "tight lower bound" of $f(n)$.

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

1, 2, 4, 8, ... n

Let k^{th} term = n

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

taking log on both sides.

$\log n = (k-1) \log_2 2.$

$k = \log n + 1$

$O(1 + \log n)$

$O(\log n)$ = Ans.

Answer 3 : $T(n) = 3T(n-1)$ → (1)

putting $n = n-1$ in eq^n (1)

$T(n-1) = 3T(n-2)$ → (2)

put (2) in (1)

$T(n) = 9T(n-2)$

putting $n = n-2$ in eq^n (1)

$T(n-2) = 3T(n-3)$ → (3)

$T(n) = 27\,T(n-3)$

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

$n - k = 0$

$n = k$

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

$= 3^n T(0)$

$= 3^n$

$O(3^n)$ Ans.

Answer 4:

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

$n = n-1$ in eq^n (1)

$T(n-1) = 2T(n-2) \rightarrow (ii)$

$T(n) = 4T(n-2) \rightarrow (iii)$

putting $n = n-2$ in eq^n (1)

$T(n-2) = 2T(n-3) \rightarrow (iv)$

$T(n) = 8T(n-3)$

$T(n) = 2^k T(n-k)$

$n-k = 0$

$k = n$

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

$= 2^n T(0)$

$= 2^n$

$O(2^n)$ = Ans.

Answer 6:

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

complexity: $O(1 + \sqrt{n} + \sqrt{n} + \sqrt{n})$

$O(1 + 3\sqrt{n})$

$O(3\sqrt{n})$

$O(\sqrt{n})$

$O(n^{1/2})$ = Ans.

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 ++;
}
```

i	j	k
$n/2$	1	1
↙	↙	2
$n/2$	$\log_2 n$	4
		⋮
		$n \searrow \log_2 n$

$O(n/2 \times \log_2 n \times \log_2 n)$

$O(n(\log_2 n)^2)$ = Ans.

Answer 8:

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

n	i	j
	1	1
	⋮	⋮
	2	n
	⋮	⋮
n-3	n	n
	⋮	⋮

$n-6$ $\vdots$ $\vdots$ $\vdots$

complexity:

$1+4+7+\ldots n$

$n = 1+3(k+1)$

$= 3k-2.$

no. of terms

$k = n+2/3$

$= n+2/6\,[2+(n-1/3)\times 3]$

$= [n+2/6\,(n+1)] \times n^2$

$= O\,[(n^2+3n+2)/6 \times n^2]$

$= O(n^4)$ Ans =

Answer 9:

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

n   n2   n2   n2

$O(n+n^2+n^2+n^2)$

$O(3n^2+n)$

$O(n^2)$ Ans =

Answers:

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

$i = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad \ldots$

$s = 1+3+6+10+15+\ldots+n.$

sum of $s = 1+3+6+10+\ldots+n \rightarrow (1)$

also $s = 1+3+6+10+\ldots+n-1+n \rightarrow (2)$

from (1) - (2)

$0 = 1+2+3+4+\ldots \quad n-n$

$T_k = 1+2+3+4+\ldots k$

$T_k = 1/_2\, k(k+1)$

for k, ~~inter~~

$1+2+3+\ldots+k <= n$

$k(k+1)/_2 \leq n$

$(k^2+k)/_2 <= n$

$O(k^2) <= n$

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

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

$O(n^{1/2})$ Ans: