Tutorial - 1

Q1.

Ans→ Asymptotic Notation are the mathematical notation used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

There are mainly three asymptotic notation:

→ Big-O notation
→ Omega notation
→ Theta notation

Big-O-Notation

Big-O notation represent the upper bound of the running time of an algorithm. Thus, it gives the worst case complexity of an algorithm.

Omega-Notation

Omega notation represent the lower bound of the running time of an algorithm. Thus it provides the best case complexity of an algorithm.

Theta Notation

Theta Notation encloses the function from above and below. Since, it represent the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average case complexity of an algorithm.

Saraswat Shah

Page (1)

Q2.

See: for (int i = 1; i < n; i = i * 2)

$T(1) = 1$

$i = 1, 2, 4, 8, \text{----} 2^{k-1}$

$T(n) = 1 + 2 + 4 + 8 \text{---} 2^{k-1}$

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

$n = 2^k$

$\log n = k \log 2.$

$k = \log_2 n.$

So, the time complexity is $O(\log_2 n)$

Q3. $T(n) = 3T(n-1)$ —— (1)

$T(1) = 1$

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

$T(n-1) = 3T(n-2)$ —— (2)

Put eq (2) in eq (1)

$T(n) = 3(3T(n-2))$ —— (3)

Put $n = n-2$ —— in eq (2)

$T(n-2) = 3T(n-3)$ —— (4)

Put eq (4) in eq (3)

$T(n) = 3 \cdot 3 \cdot (3T(n-3))$ —— (5)

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

Saraswat Shah

$T(n) = 3^n T(0)$

Clearly, above the complexity of this function is $O(3^n)$.

Q4. $T(n) = 2T(n-1) - 1$ — (1)

Put $n = n-1$ in eq (1).

$T(n-1) = 2T(n-2) - 1$ — (2)

Put eq (2) in eq (1).

$T(n) = 2(2T(n-2) - 1) - 1$ — (3).

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

Put $n = n-2$ in eq (1).

$T(n-2) = 2T(n-3) - 1$ — (4).

Put eq (4) in (3).

$T(n) = 2^3 T(n-3) - 2^2 - 2^1 - 2^0$ — (5)

$T(n) = 2^{n-k} T(n-k) - 2^{n-k+1} - 2^{n-k+2} - \cdots - 2^{n-1}$ — (6)

$2^{n-k} = 1$

$(n-k)\log 2 = 0$

$k = n$ ← Put in eq (6)

$T(n) = T(1) - (2^1 + 2^2 + 2^3 + \cdots 2^{n-1})$

$= 1 - 2(2^{n-1} - 1)$

$T(n) = 2^n + 3$

$O(2^n)$ Ans

Sraswat Shah

Q5.

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


We can define the term 's' according to relation $S_i = S_{i-1} + i$. The value of 'i' increases by one for each iteration. The value contained in 's' at the i^{th} iteration is the sum of the first 'i' positive integers. If k is total number of iteration taken by the program, then while loop terminates if: $1 + 2 + 3 + \dots + k = \frac{k(k+1)}{2} > n$

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

Time complexity is $O(\sqrt{n})$.

Q6.)

$1^2 + 2^2 + 3^2 + \dots + (n-x)^2$.

$1 + 4 + 9 + \dots + (n-k)^2$.

$T(n) \quad \frac{(n)(n+1)(n+2)}{6}$

Ans → $O(n^3)$ → Time Complexity

Q7.)

```
for (k=1 ; k<=n ; k=k*2)
```
↳ O(log n)

```
for (j=1 ; j<=n ; j=2*j)
```
↳ O(log n)

```
for (int i= n/2 ; i<=n ; i++)
```
↳ O(n)

Time complexity $O(n \log^2 n)$.

Q8.

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

Time complexity $O(n*n*n) = O(n^3)$.

Q9.

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

for inner loop
$O(n)$

for outer loop.
$n + n + - - - - - + n$ times.

$= n^2$

$O(n^2) \rightarrow$ times complexity.

i (10.) for the function, n^k and c^n, what is the asymptotic relationship between these function?
$k >= 1$ & $c > 1$ are constant.

It should be n^k is $O(c^n)$