Rudraksh Agarwal
48
ML

# Assignment-1

**Q-1** What do you understand by Asymptotic Notations. Define different Asymptotic notation with examples.

**Ans 2** Asymptotic notation is a mathematical tool used in computer science to describe the behaviour of functions as their inputs grow towards infinity. It is particularly used in analyzing the performance and efficiency of algorithms. There are 3 primary asymptotic notations:

1. Big O notation (O)
2. Omega notation ($\Omega$)
3. Theta Notation ($\Theta$)
4. Small O Notation (o)
5. Small Omega ($\omega$)

## 1. Big O notation (O):

Big O notation represents the upper bound of a func$^n$'s growth rate. It provides as asymptotic upper bound for the func$^n$, indicating the worst case scenario.

$f(n) = O(g(n))$

$g(n)$ is 'tight' upper bound of $f(x)$

$f(n) = O(g(n))$
iff $f(n) \leq g(n) \times C$
$\forall n \geq$ no, & for some constant $C > 0$

Ex→ If we have an algorithm with a time complexity $O(n^2)$, it means the algorithm execution time grows as the input size ($n$) increases. A simple nested loop where each loop iterates upto $n$.

2) Big omega Notation / Omega Notation ($\Omega$):

Omega Notation represents the lower bound of a func$^n$'s growth rate. It provides an asymptotic lower bound for the func$^n$, indicating the best case scenario.

$f(n) = \Omega(g(n))$
$g(n)$ is tight lower bound of $f(n)$

$f(n) = \Omega(g(n))$
iff $f(n) \geq c\,g(n)$
$\forall\ n \geq n_0$ and for some constant $c > 0$.

Ex→ A simple iteration through an array.

3) Theta Notation ($\Theta$):

It represents the tight bound of a func$^n$'s growth rate. It provides both upper & lower bounds, indicating the exact behaviour of the func$^n$.

$f(n) = \Theta(g(n))$
$f(n) = \Theta(g(n))$ & $f(n) = \Omega g(n)$
iff $c_2 g(n) \leq f(n) \leq c_1 g(n)$
$\forall\ n > \max(n_1, n_2)$ & some constants $c_1 > 0$ & $c_2 > 0$.

Ex→ linear search algorithm.

## 4) Small o notation (o):

It represents a stricter upper bound that Big O notation. It denotes func$^n$ that grow strictly slower than another func$^n$.

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

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

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

iff $f(n) < g(n)$

$\forall\ n > n_0$

Ex → merge sort

## 5) Small Omega Notation (ω):

Small omega notation represents a stricter lower bound than Omega notation. It denotes func$^n$'s that grow strictly faster than another func$^n$.

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

$f(n)$ is lower bound of $f(n)$

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

iff $f(n) > c(g(n))$

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

Ex → worst case of bubble sort.

Q-2) What should be the time complexity of

```
for (i=0 to n)
{
    i = i*2;
}
```

Sol^n → Let no. of iterations required for 'i' to exceed 'n' as 'k'.

$2^k > n$

Taking $\log_2$ both sides :-

$k = \log_2(n)$

∴ Time complexity is $O(\log_2(n))$

Q-3) $T(n) = \begin{cases} 3\,T(n-1), & \text{if } n>0 \\ 1, & \text{otherwise} \end{cases}$

Sol^n → $T(n) = \begin{cases} 3\,T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

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

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

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

⋮

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

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

$n=0,\ T(0)=1$

Now,

$T(n) = 3 \cdot 3T(n-2) = 3^2 T(n-2)$

$T(n) = 3^2 \cdot 3T(n-3) = 3^3 T(n-3)$

generally,

$T(n) = 3^{n+1} \cdot 3T(1) = 3^n T(1)$

here let $T(1) = c$

$T(n) = 3^n \cdot c$

$\therefore$ Time complexity is $T(n) = O(3^n)$.

**Q.4)** $T(n) = \begin{cases} 2T(n-1) - 1 & , \text{if } n > 0 \\ 1 & , \text{otherwise} \end{cases}$

Sol$^n$ → $T(n) = 2T(n-1) - 1$

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

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

$\vdots$

$T(1) = 2T(0) - 1 \quad \{\because T(0) = 1\}$

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

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

Similarly,

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

Say $k = n$, we get

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

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

$\therefore$ Time complexity is

$T(n) = O(2^n)$ Ans

Q-5. What should be the time complexity of—

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

Ans → The loop will execute approximately $O(\sqrt{n})$ times

∴ Time complexity is

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

Q-6 Time complexity of—

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

Ans → Since, $i^2 \leq n$

$\therefore i \leq \sqrt{n}$

∴ the loop iterates $\sqrt{n}$ times.

∴ Time complexity is $O(\sqrt{n})$.

Q7. Time Complexity of —

```
void func^n (int n) {
  int i, j, k, count = 0;
  for (i = n/2 ; i <= n ; i++)          ——> n/2 times
      for (j = 1 ; j <= n ; j = j * 2)   ——> log2(n)
          for (k = 1 ; k <= n ; k = k * 2)   ——> log2(n)
              count++;
}
```

Ans → ∴ The total no. of iterations

$\frac{n}{2} \times \log_2(n) \times \log_2(n)$

Time Complexity is: $O(n \log^2(n))$.

Q8. Time complexity of —

```
func^n (int n) {
  if (n == 1) return;
  for (i = 1 to n) {          ——> O(n)
      for (j = 1 to n) {      ——> O(n)
          printf("*");
      }
  }
  func^n (n-3);               ——> n/3 times
}
```

Ans → Total Time complexity $= O(n) \times O(n) \times \left(\frac{n}{3}\right)$

Time Complexity $= O\left(\frac{n^3}{3}\right)$

⇒ Time Complexity is $O(n^3)$. ☆

Q→⑨ Time complexity of—

```
void func^n (int n) {
    for(i=1 to n) {                    → O(n)
      for(j=1 ; j<=n; j=j+i) {         → O(n log n)
        printf("*");
      }
    }
}
```

Ans → ∴ Total Time complexity ⇒ O(n) × O(log n)
⇒ O(n log n). Ans

Q→⑩ For the func^n's $n^k$ & $c^n$ what is asymptotic relation between these func^n's ?
Assume that k>=1 & c>1 are constants. Find out the value of c & n₀ for which relation holds.

Ans. • $n^k = c^n$

Taking log both the sides

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

$\frac{\log n}{n} = \frac{\log c}{k}$

Let $x = \frac{\log n}{n}$

$y = \frac{\log c}{k}$

$x = y$

∴ This determines that n₀ is a point where $n^k$ & $c^n$ intersect.