Tutorial - 1

① Lakshay Sharma
AI and DS
10

1) Asymptotic Notations :-

- Asymptotic Notations are the mathematical notations 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 notations:-
  1. Big - O notation
  2. Omega notation.
  3. Theta notation

(I) Big - O Notation :-

- Gives upper bound of the running time of an algorithm
- Gives worst-case complexity of an algorithm given two ~~fun~~ function $f(n)$ and $g(n)$

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

iff $f(n) \leq Cg(n)$ for all $n \geq n_0$ $C > 0$

example :- In bubble sort,
when the input array is in reverse condition, the algorithm takes the maximum time (n^2) to sort the elements i.e. the worst case.

②

(II) Omega Notation (Ω) :-

- represents the lower bound of the running time of an alogorithm.
- It provides the best case complexity of an algorithm

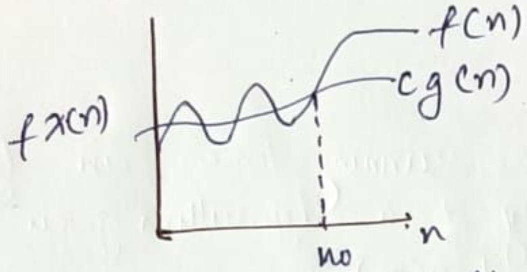
given two functions

$$f(n) \text{ and } g(n)$$

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

iff

$$f(n) \geq c\, g(n) \quad \text{for all } n \geq n_0 \quad c > 0$$



example :- In bubble sort, when the input is already sorted, the time taken by the algorithm is linear, i.e., best case.

(III) Theta Notation (Θ) :-

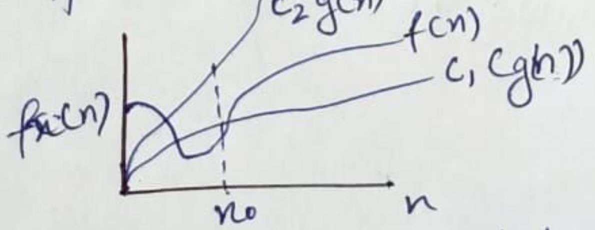
- Represents the upper and lower bound of the running time of an algorithm.
- provides the average-case complexity of an algorithm

given two of f(x) f(n) and g(n)

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

$$\text{iff} \quad c_1 g(n) \leq f(n) \leq c_2 (g(n))$$

$$\text{for all } n \geq n_0 \,;\; c_1\, c_2 > 0$$



example :- In bubble sort, when the input array is neither sorted nor in reverse order. Then it takes average-time.

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

$i = 1, 2, 4, 8, 16, 32 \ldots 2^k$

$n = ar^{k-1}$

$n = 1.2^{k-1} = 2^{k-1}$

$n = \frac{2^k}{2} \Rightarrow \boxed{2n = 2^k}$ $\quad \log(2n) = k \log 2$

$k = \log(2n)$

$\Rightarrow$ Complexity $= O(\log n)$

3> $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$

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

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

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

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

$\vdots$

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

$= 3^n \; [\because T(0) = 1]$

complexity $\Rightarrow O(3^n)$

4> $T(n) = \{2T(n-1) - 1$ if $n>0$; otherwise $1\}$

$T(n) = 2T(n-1) - 1$ —① $\quad T(1) = 2T(1-1) - 1$

$= 2T(0) - 1$

$T(1) = 2 - 1 = 1$ —②

puting $n = n-1$

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

$T(n) + 1 = 2(2T(n-2) - 1)$ {from ①}

$T(n) = \cancel{2^2} \; 2^2 T(n-2) - 2 - 1$

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

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

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

$\vdots$

$\vdots$

$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - \dots 2^2 - 2^1 - 2^0$

$T(1) = 1$ (from (2))

$n-k = 1$

$T(n) = 2^{n-1} T(1) - [2^0 + 2^1 + 2^2 + \dots + 2^{n-3} + 2^{n-2}]$

$= 2^{n-1} \times 1 - [2^{n-1} - 1]$

$= 2^{n-1} - 2^{n-1} + 1$

$T(n) = 1$

Complexity $\Rightarrow$ $O(1)$

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

s: 3, 8, 10, 15, 21, $\vdots$, n — k times

$S_k = 3+6+10+10+15+21+ \dots T_k$

$S_{k-1} = 3+6+10+15+ \dots T_{k-1}$

$S_k - S_{k-1} = 3+3+4+5+6+ \dots + T_k - T_{k-1}$

$T_k = 3 + [3+4+5+6 \dots + (k-1) \text{ times}]$

$= 3 + \frac{(k-1)}{2} (6+(k-2).)$

$= 3 + \frac{(k-2)}{2}(k+4)$

(T_k) k^{th} term.

for last iteration k^{th} term is n.

$T_k = n$

$\frac{(k-2)(k+4)+3}{2} = n.$

For time complexity removing lower order terms

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

6> void function (int n)

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

i	times
1	$\sqrt{1}$
2	$\sqrt{2}$
3	$\sqrt{3}$
4	$\sqrt{4}$
⋮	⋮
n	$\sqrt{n}$

complexity → $O(\sqrt{n})$

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	1	1
n-1	2	2
⋮	4	⋮
n/2	⋮	⋮
$\frac{n}{2}$ times	log n	log n

$= O\left(\left(\frac{n}{2}\right)(\log n)(\log n)\right)$

complexity → $O(n(\log n)^2)$

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

n	i	j
n	1	1 - - - n
	⋮	1 - - - n
	n	⋮ n
n-3	1	1 - n
	⋮	⋮
n-6	- -	- -
⋮	⋮	⋮

$= n + n-3 + n-6 - - - - + 7 + 4 + 1$

(or) $1+4+7+ - - - + n-3+n$

$n = 1+3(k-1)$ $\{ \because n^{th} \Rightarrow a+(n-1)d \}$

$k = \frac{n+2}{3}$

$\Rightarrow$ Time complexity of recursion $= O(n)$

i	j
1	1
⋮	⋮
n	n

$\Rightarrow n^2$

Total time complexity $= O(n^3)$

(6)

9)
```
void function (int n)
{ for (i=1 to n)
  { for (j=1 ; j<=n ; j=j+2)
        printf(" * ")
  }}
```

i	j
1	1 ⋮ n
2	1 3 ⋮ n
n	n

$= n^2$

Total complexity → $O(n^2)$