Prakhar Pande
Section -D
Roll No. -28 Univ No.: 2016906

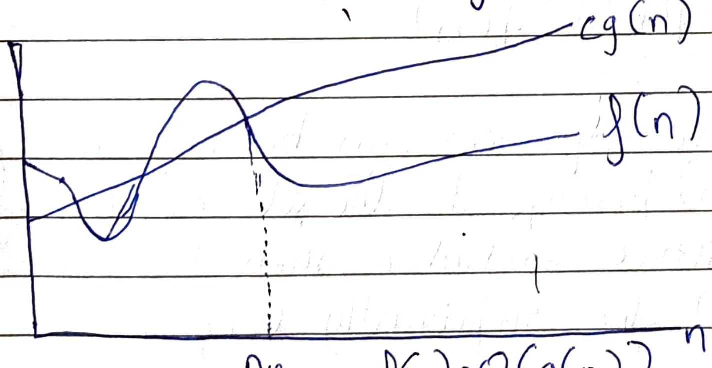
classmate
Date
Page

Design and Analysis of Algorithm

Tutorial - 1

1) Asymptotic Notation is used to describe the running time of an algorithm, how much time an algorithm takes with a given input, n. There are mainly three asymptotic notations:
- Big-O notation.
- Omega notation.
- Theta notation.

Big - O Notation (O-notation):-
Big-O notation represents the upper bound of the running time of an algorithm. Thus it gives the worst-case complexity of an algorithm.



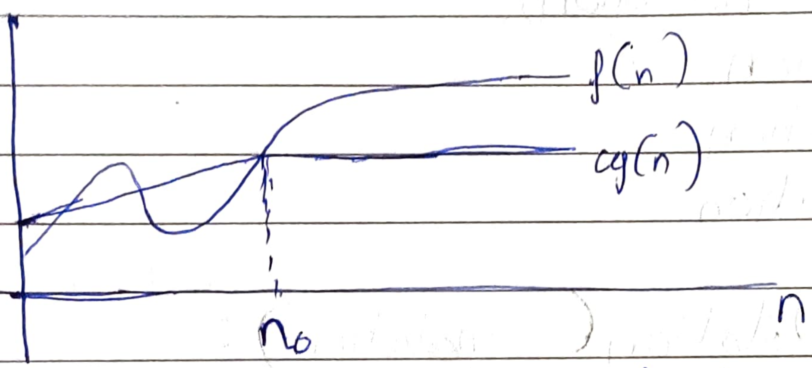
$f(n) = O(g(n))$,

$O(g(n)) = \{ f(n):$ there exist positive such that
$0 \leq f(n) \leq cg(n)$.

The above expression can be described as a function $f(n)$ belongs to the set $O(g(n))$ if there exists a positive constant c such that it lies between 0 & $cg(n)$, for sufficiently large n. For any value of n, the running time of an algorithm does not cross the time provided by $O(g(n))$.

- Omega Notation (Ω-notation)

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



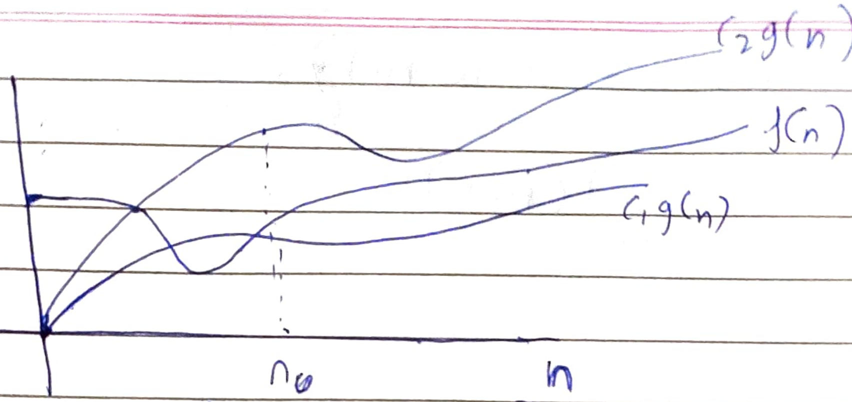
Omega gives the lower bound of a function.

$\Omega(g(n)) = \{ f(n)$: there exist positive such that
$0 \leq cg(n) \leq f(n)$.

The above expression can be described as a function $f(n)$ belongs to the set $\Omega(g(n))$ if there exists a positive constant c; such that it lies above $cg(n)$, for sufficiently large n.

- Theta Notation (Θ-notation).

Theta notation encloses the function from above and below. Since it represents the upper & the lower bound of running time of an algoritm, it is used for analyzing the average-case complexity of an algorithm.



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

Theta bounds the function with constants factors.

$\Theta(g(n)) = \{ f(n) :$ there exist positive such. that $0 \le c_1 g(n) \le f(n)$.

The above expression can be described as a function $f(n)$ belongs to the set $\Theta(g(n))$ if there exist positive constants c_1 & c_2 such that it can sandwiched between $c_1 g(n)$ & $c_2 g(n)$, for sufficiently large n.

Q2 for ($i=1$ to n) { $i = i * 2$; }

for ($i = 1$ to n) // $i = 1, 2, 4, 8 \dots n$
{ $i = i * 2$ } // $O(1)$

$\Rightarrow \sum_{i=1} 1 + 2 + 4 + 8 + \dots + n$

K^{th} term of GP $\Rightarrow T_K = a r^{K-1}$

$n = 1 \cdot 2^{K-1}$

$n = 2^{K-1} \Rightarrow n = 2K/2$

$2^n = 2K$

$\log_2 (2n) = K(\log_2 2)$

$K = \log_2 2n$

$K = \log_2 2 + \log_2 n \Rightarrow K = 1 + \log_2 n$

$O(\log_2 n)$

$O(n)$.

Q3 $T(n) = \{3\cancel{6}T(n-1)$ if $n>0$, otherwise $1\}$

$$T(n) = 3T(n-1)$$
$$= 3(3T(n-2))$$
$$= 3^2\cancel{6}\,T(n-2))$$
$$= 3^3\,T(n-3)$$
$$= 3^n\,T(n-n)$$
$$= 3^n\,T(0)$$
$$= 3^n$$
$$\Rightarrow O(3^n)$$

Q4 $T(n) = \{2T(n-1) - 1$ if $n>0$, otherwise $1\}$

$$T(n) = 2T(n-1) - 1$$
$$= 2(2T(n-2) - 1) - 1$$
$$= 2^2(T(n-2)) - 2 - 1$$
$$= 2^2(2T(n-3) - 1) - 2 - 1$$
$$= 2^3\,T(n-3) - 2^2 - 2^1 - 2^0$$

$$\cancel{2^n\,T(n-3) - 2^2 - 2^1 - 2^0}$$
$$2^n\,T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3}$$
$$\ldots\ldots 2^2 - 2^1 - 2^0$$
$$2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots 2^2 - 2^1 - 2^0$$
$$= 2^n - (2^n - 1)$$
$$T(n) = 1$$
$$O(1)$$

Q5 What should be the time complexity

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

$i = 1, 2, 3, 4, 5, 6 \ldots\ldots K$

$s = 1 + 2 + 3 + 4 + 5 \ldots\ldots K$

when $S \geq n$, then loop will stop at K^{th} iteration.

$$S \geq n \Rightarrow S = n$$

$$2 + 2 + 3 + 4 + \ldots + K = n$$

$$1 + (K \cdot (K+1))/2 = n$$

$$K^2 = n \Rightarrow K = \sqrt{n}$$

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

Q6 Time complexity of ~~void function(int n)~~

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

as $i^2 <= n$

$i < \sqrt{n}$

$i = 1, 2, 3, 4, \ldots \sqrt{n}$

$$\sum_{i=1}^{n} 1 + 2 + 3 + \ldots\ldots + \sqrt{n}$$

$$T(n) = \frac{\sqrt{n} \times (\sqrt{n}+1)}{2}$$

$$T(n) = \frac{n\sqrt{n}}{2}$$

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

Q7. 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 ++.
```

for $k = k * 2$

$k = 1, 2, 4, 8, \ldots n;$

G P $\Rightarrow a = 1, r = 2$

$$\frac{a(r^n - 1)}{r - 1} = \frac{(2^k - 1)}{1}$$

$$n = 2^k$$

$$\log n = k$$

i	j	k
1	$\log n$	$\log n * \log n$
2	$\log n$	$\log n * \log n.$
⋮		
n	$\log n$	$\log n * \log n.$

$\Rightarrow O(n * \log n * \log n)$

$\Rightarrow O(n \log^2 n).$

classmate

Date

Page

Q8 function(int n)

{

if(n ==1)

return }

for(i=1 to n)

{

for(j =1 to n)

}

function(n-3);

}

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

a.t.,

$a = 1, b = 3, f(n) = n^2$

$c = \log_3 1 = 0$

$n^0 = 1 > (f(n) = n^2)$

$T(n) = \Theta(n^2)$

Q9 void function(int n)

{

for(i =1 to n)

{

for(j =1; j <=n; j+=1)

printf("*");

}

}

for $i = 1 \Rightarrow j = 1, 2, 3, 4, \ldots n$.

for $i = 2 \Rightarrow j = 1, 3, 5, 7 \ldots n$.

for $i = 3 \Rightarrow j = 1, 4, 7, \ldots n$.

for $i = 1 \Rightarrow j = 1, 2, 3, 4, \ldots n$

for $i = 2 \Rightarrow j = 1, 3, 5, 7, \ldots n$

for $i = 3 \Rightarrow 1, 4, 7 \ldots n$

for $i = n \Rightarrow j = 1 \ldots \ldots 1$

$$\sum_{i=n}^{1} n + 1/2 + n/3 + n/4 + \ldots\ldots 1/n]$$

$$\sum_{j=n} n(\log n)$$

$$T(n) = (n \log n)$$

$$T(n) = O[n \log n].$$

Q10 as given n^k & c^n
relation b/w n^k & c^n is

$$n^k = O(c^n)$$

as $n^k \leq c^n$

$\forall\ n \geq n_0$ & some constant $a > 0$

for $n_0 = 1$

$c = 2$

$\Rightarrow 1^k \leq a\, 2^1$

$n_0 = 1$ & $c = 2$.