Agenda for today's session

Asymptotic Notation :

1. Big O Notation

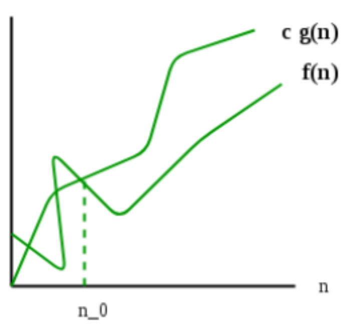
2. Omega Notation

3. Theta Notation

Big O Notation(very very important used everywhere) : upper bound of an algorithm

f(n) = O(g(n)), if f(n) <= cg(n) for all n , n >= n0 such that there exists two positive constants where c > 0 and n0 >= 1

So, if we say a = O(b) meaning is (b greater than a after taking c help)



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

Problem 1 :

f(n) = 5n

g(n) = n

f(n) = O(g(n)), it means f(n) <= c. (g(n))

5n <= c.n for all n, n >= 1

c = 5   5n <= 5n

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

now what should be the value of a constant "c"?

5

Problem 2 :

f(n) = n^2

g(n) = n

f(n) = O(g(n)), it means f(n) <= cg(n) (Mathematical definition)

n^2 <= c.n for all n, n >= 1 -------> equation 1

now what should be the value of constant "c"?

c = n   // Is this a constant -> not at all because here c depends on the value of n.....bigger the value of n, bigger the value of camd smaller the value of n,smaller the value of c.

c = 3 4 6

n^2 <= n^2

f(n) is not equal to O(g(n))

Increasing order of complexities :

1. **Constant Complexity :** O(1)

2. **Logarithmic Complexity :** O(logn)

3. **Linear Complexity :** O(n)

4. **Quadratic Complexity :** O(n^2)

5. **Cubic Complexity :** O(n^3)

6. Polynomial Complexity : **O(n^c) ; where c is constant**

7. Exponential Complexity : **O(c^n) ; where c is constant**

8. n! < n^n

2^n < n^n

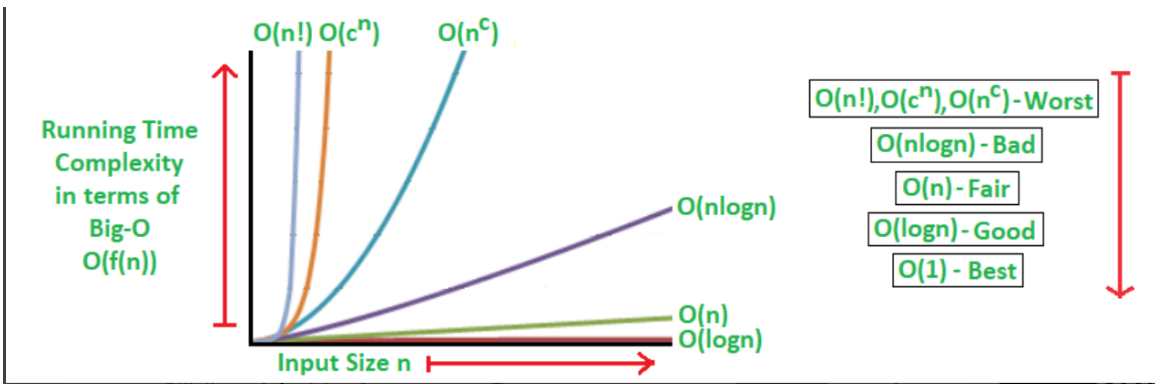
n! > 2^n

2^n < n! < n^n

OR

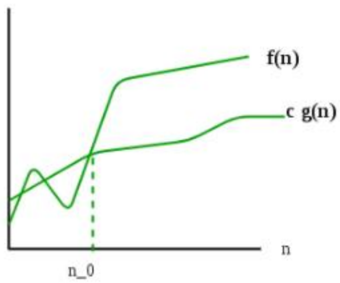
2^n = O(n!) ; n! = O(n^n)

Complexity Graph(Focus very carefully)



Omega Notation :

Let f(n) = omega(g(n)) if and only if f(n) >= c(g(n)) for all n, n>=n0 such that there exists two positive constants c > 0 and n0 >= 1.



f(n) = Omega(g(n))

Problem 1 :

f(n) = n   g(n) = 5n

f(n) = omega(g(n))

What should be the value of c ?

f(n) >= c(g(n))

n >= c. 5n

c = 1/5 - constant

f(n) = omega(g(n))

Problem 2 :

f(n) = 5n   g(n) = n

f(n) = omega(g(n))

What should be the value of c ?

f(n) >= c. omega(g(n))

5n >= c.n

c = 1 => 5n >= 1.n

f(n) = omega(g(n))

Problem 3 :

f(n) = n^2   g(n) = n^2 + n + 10

f(n) = omega(g(n))

What should be the value of c ?

f(n) >= c.g(n)

n^2 >= c.(n^2+n+10)

c = 1/2

n^2 >= 1/2(n^2 + n + 10)

f(n) = omega(g(n))

Problem 4 :

f(n) = n   g(n) = n^2

f(n) = omega(g(n))

What should be the value of c ?

f(n) >= g(n)

n >= c.n^2

c = 1/n -> means it is not a constant

That's why f(n) is not equal to omega(g(n)).

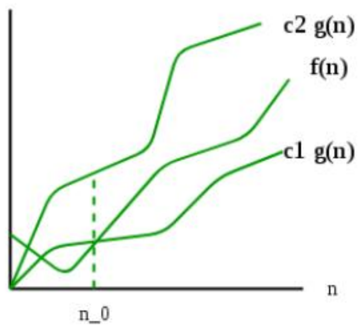
Theta Notation : which satisfies both bigO and omega

Let f(n) = theta(g(n)) if and only if f(n) >= c1g(n)(omega) and f(n) <= c2(g(n))(Big O) for all n; n >= n0 such that there exists three constants.

c1 > 0

c2 > 0

n0 >= 1



f(n) = theta(g(n))

Problem 1 :

f(n) = n g(n) = 5n

f(n) = theta(g(n))

1. Omega -> f(n) >= c.g(n)

n >= c. 5n

c = 1/5 - constant

f(n) = omega(g(n))

2. Big O -> f(n) <= c.g(n)

n <= c.5n

c = 1 - constant

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

Thus because f(n) holds true for both omega as well as O, thus f(n) = theta(g(n))

Problem 2 :

f(n) = n - 10 g(n) = n + 10

f(n) = theta(g(n))

1. Omega -> f(n) >= c.g(n)

n-10 >= c.(n+10)

c = 1/2 - constant

f(n) = omega(g(n))

2. Big O -> f(n) <= c.g(n)

n-10 <= c.(n+10)

c = 1

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

We can say that f(n) = theta(g(n)) as it holds true for both omega as well as big O.

Problem 3 :

f(n) = n   g(n) = n

f(n) = theta(g(n))

1. Omega -> f(n) >= c.g(n)

n >= c.n

c = 1 - constant

Thus, f(n) = omega(g(n))

2. Big O -> f(n) <= c.g(n)

n <= c.n

c = 1 - constant

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

And thus f(n) = theta(g(n)) as it holds true for both omega and Big O

Problem 4 :

f(n) = n   g(n) = n^2

f(n) = theta(g(n))

1. Omega -> f(n) >= c.g(n)

n >= c.n^2

c = 1/n - Not a constant

f(n) is not theta(g(n))

--