Introduction to algorithms

What is Algorithm?

Sequence of finite steps used to solve the particular problem.

Properties of Algorithms:

- It should terminate after finite time
- It should produce at-least one output
- It should be unambiguous [DETERMINISTIC]
    - For same input same output will always come

Types of Analysis:

Aposteriori Analysis	Apriori Analysis
Dependent on **Language of compiler** and type of hardware used	Independent of Language of compiler and hardware.
Exact Answers	Approximate answers

Few problems:

```
main(){
          i = 1
          while i < n{

                        i = 2 * i
}
}
```

$$TimeComplexity = log2(n)$$

```
main(){
          i = n
          while i > 1{
                        i = i / 2
                        i = i / 5
                        i = 2 * i
}
}
```

$$i/5^k = 1n/5^k = 1$$

k is number of times my loop is running.

$$Timecomplexity = log5(n)$$

```
main(){
        i = n
        while i > 2 {
                    i = sqrt(i) }

}
```

$$i = n = n^(1/2) = n^(1/2)(1/2)$$

n ^(1/2^k) = 2

log2(n ^(1/2^k)) = log2(2)

(1/2^k) log2(n) = 1

log2(n) = 2^k

log2(log2(n)) = k log2(2)

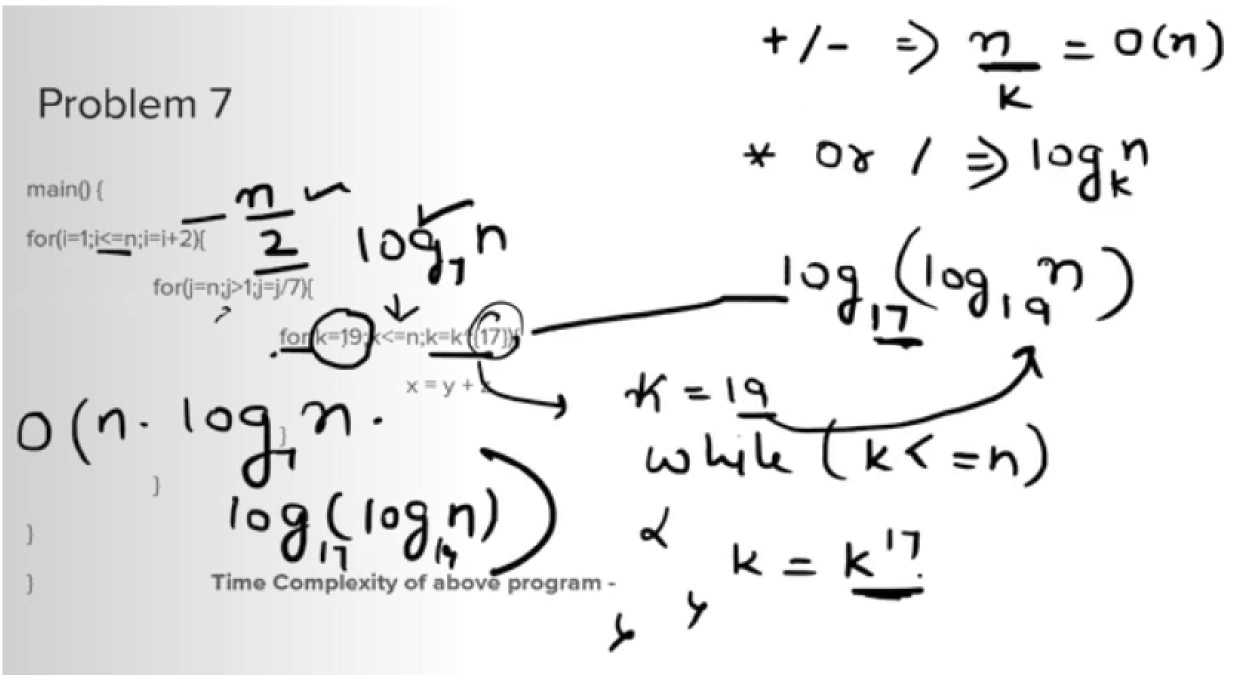
$$Timecomplexity = O(log2(log2(n)))$$

```
main(){
        i = n
        while i > 2 {
                    i = (i)**(1/25) }

}
```

$$TimeComplexity = O(log25(log2(n)))$$

+/-	O(n)
* or /	logk(n)



Asymptotic Notations:

Big O Notation

Let f(n) and g(n) be two positive functions. Then,

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

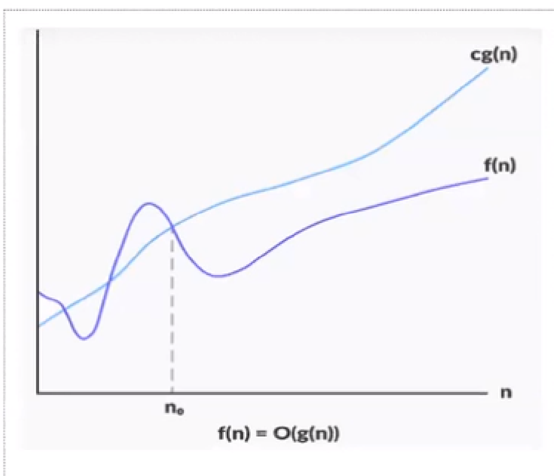
if f(n) <= c.g(n) ; for all n, n >= n_0 such that there exists two positive constants.

$$c > 0 , n_0 >= 1$$

For example : a = O(b)

Meaning : b is somehow greater than a after taking c help.

Graphical Representation of Big O



- F(n) = O(g(n))
- If there exists a positive constant c such that it c >= 0, for sufficiently large value of n.
- For any value of n, the running time of an algorithm does not cross the time provided by O(g(n))

Omega Notation

Let f(n) and g(n) be two positive functions. Then,

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

if f(n) >= c.g(n) ; for all n, n >= n_0 such that there exists two positive constants.

$$c > 0 , n\_0 >= 1$$

For example : a = Omega(b)

Meaning : b is somehow lesser than a after taking c help.

Problem 2

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

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

$\underline{n^2} >= \underline{c}. (n^2+n+10)$

f(n) = n^2

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

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

$n = 100$

10000

$c \leq \frac{1}{2}$

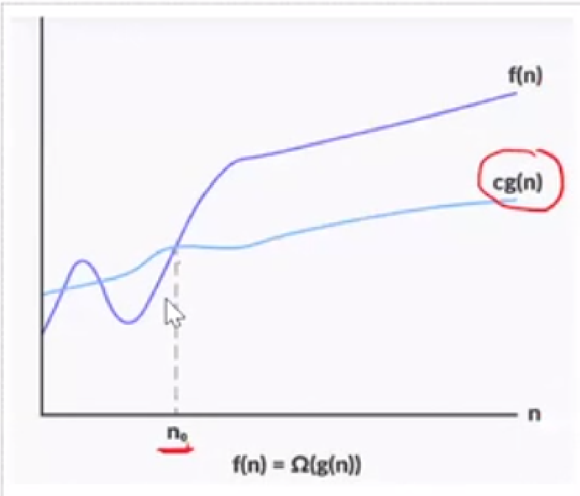
$\frac{(n^2+n+10)}{2}$

$\frac{(10000+100+10)}{2}$

For what value of c above statement holds true?

$\underline{f(n) = \Omega(g(n))}$

Graphical Representation of Omega Notation



- If there exists a constant c such that it lies above cg(n), for sufficiently large value of n.
- For any value of n, the minimum time required by the algorithm is given by Omega(g(n))

Theta Notation

Let f(n) and g(n) be two positive functions. Then,

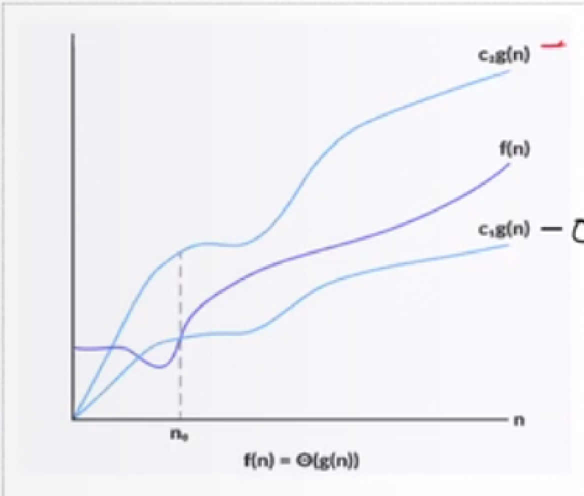
f(n) = Theta(g(n))

if f(n) >= c_1.g(n) and f(n) <= c_2.g(n) ;

for all n, n >= n_0 such that there exists two positive constants.

c_1 > 0 , c_2>0, n_0 >= 1

Graphical Representation of Theta Notation



Big O

Omega

- If there exists positive constants c1 and c2 such that it can be sandwiched between c1g(n) and c2g(n), for sufficiently large value of n.
- If a function lies anywhere in between c1g(n) and c2g(n) for all n >= n_0, then f(n) is said to be asymptotically tight bound.

Theta Notation

Problem 1

$f(n) >= c_1 g(n)$ (omega)

$n >= c_1 5 n$

f(n) = n

g(n) = 5n

$c_1 = 1/5$ constant

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

constant

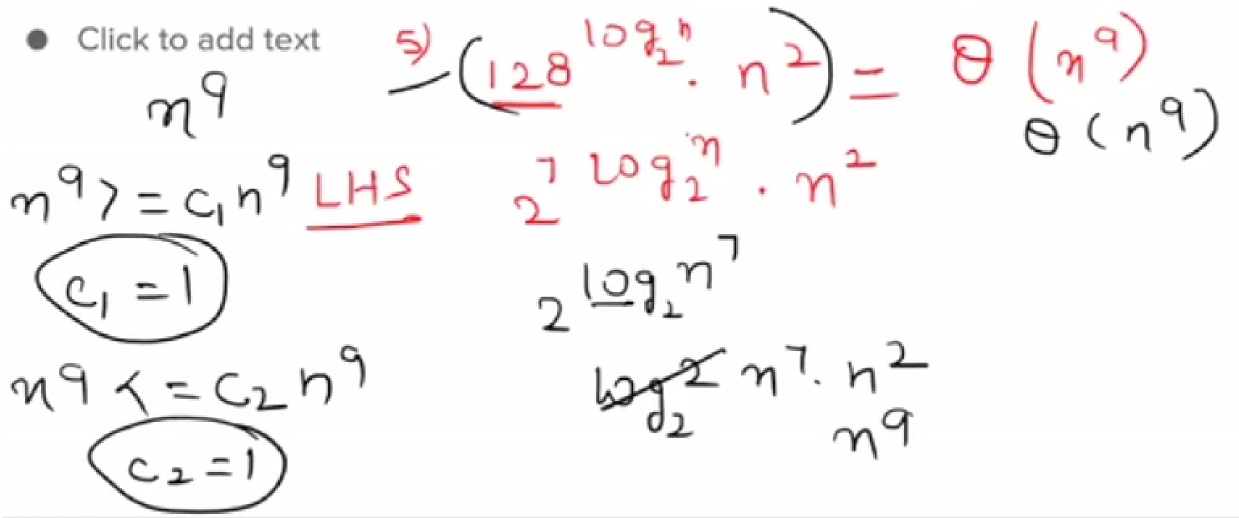
$f(n) <= c_2 g(n)$

For what value of c_1 and c_2 above statement holds true?

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

$n <= c_2 . (5n)$

$c_2 = 1/5$ constant



Complexity Classes

1. `O(1)` : Constant time complexity
2. `O(log(log(n)))`
3. `O(logn)` : Logarithmic time complexity
4. `O(root(n))`
5. `O(n)` : Linear time complexity
6. `O(n2)` : Quadratic Time complexity
7. `O(n3)` : Cubic Time Complexity
8. `O(c**n)` : Exponential Time complexity

Some points: **Important**:

1. n! < n^n
2. 2^n < n^n
3. n! > 2^n

$$2^n < n! < n^n$$

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

$$n! = O(n^n)$$

$$log(n) < n$$

$$(log(n))^2 < n$$

$$log(n)^1 1000 < n$$

$$(logn)^(logn) > n$$