FACULTY NAME	**:**	**DR.K.GEETHA**
DESIGNATION	**:**	**GUEST LECTURER**
DEPARTMENT	**:**	**SCHOOL OF COMPUTER SCIENCE,ENGINEERING AND APPLICATIONS**
CLASS	**:**	**M.S.C (CS)**
SEMESTER	**:**	**I**
SUBJECT	**:**	**DESIGN AND ANALYSIS ALGORITHM**
SUBJECT CODE	**:**	**MCS24012**

Design and Analysis of Algorithms

Design and Analysis of Algorithms

- ***Analysis:*** predict the cost of an algorithm in terms of resources and performance
- ***Design:*** design algorithms which minimize the cost

INTRODUCTION TO ALGORITHM

History of Algorithm

- The word algorithm comes from the name of a Persian author, Abu Ja’far Mohammed ibn Musa al Khowarizmi (c. 825 A.D.), who wrote a textbook on mathematics.
- He is credited with providing the step-by-step rules for adding, subtracting, multiplying, and dividing ordinary decimal numbers.
- When written in Latin, the name became Algorismus, from which algorithm is but a small step
- This word has taken on a special significance in computer science, where “algorithm” has come to refer to a method that can be used by a computer for the solution of a problem
- Between 400 and 300 B.C., the great Greek mathematician Euclid invented an algorithm
- Finding the greatest common divisor (gcd) of two positive integers. The gcd of X and Y is the largest integer that exactly divides both X and Y . Eg.,the gcd of 80 and 32 is 16.

What is an Algorithm?

- Algorithm is a set of steps to complete a task.

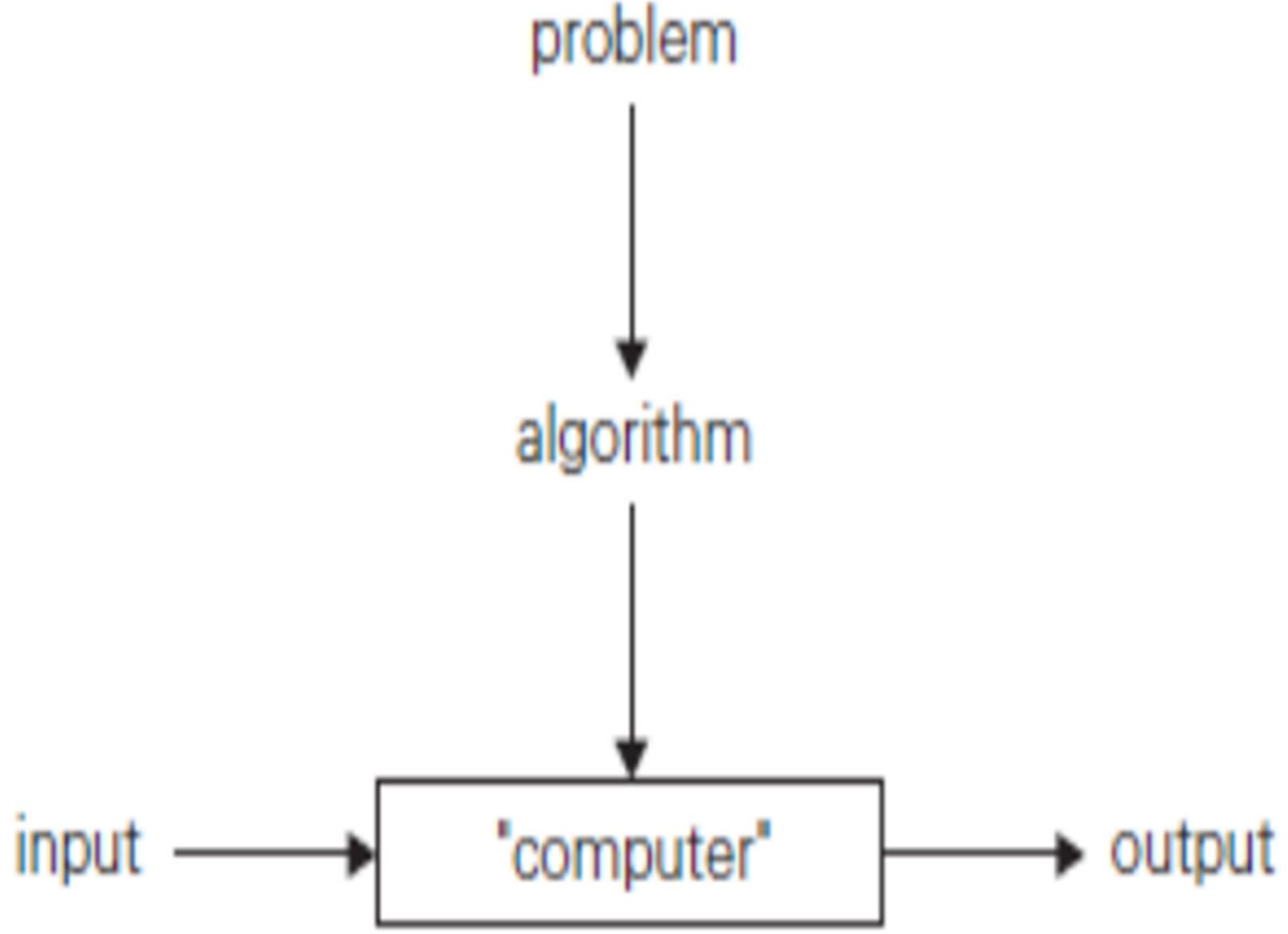
  For example, Task: to make a cup of tea. Algorithm: · add water and milk to the kettle, · boil it, add tea leaves, · Add sugar, and then serve it in cup.

- ‘’a set of steps to accomplish or complete a task that is described precisely enough that a computer can run it’’.

- Described precisely: very difficult for a machine to know how much water, milk to be added etc. in the above tea making algorithm.

Algorithm Definition:

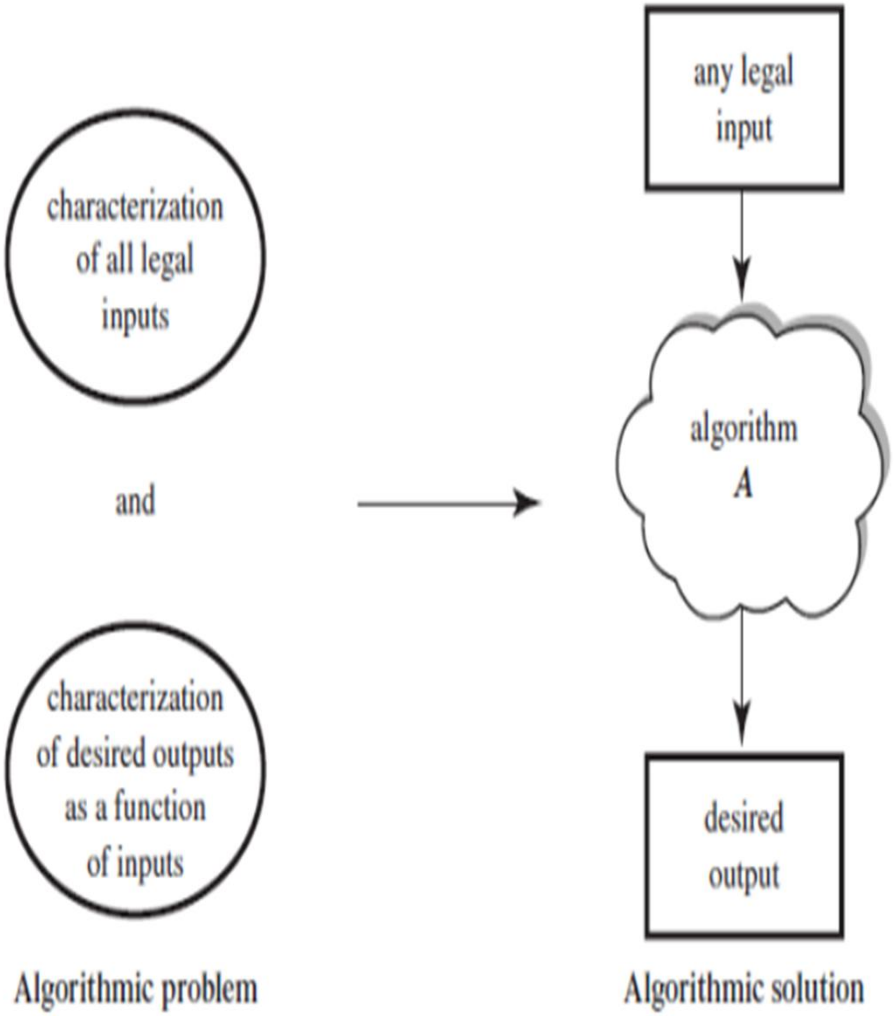
- An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:
    1. Input. Zero or more quantities are externally supplied.
    2. Output. At least one quantity is produced.
    3. Definiteness. Each instruction is clear and unambiguous.
    4. Finiteness. The algorithm terminates after a finite number of steps.
    5. Effectiveness. Every instruction must be very basic enough and must be feasible.


problem
algorithm
input
"computer"
output

Algorithms for Problem Solving

- The main steps for Problem Solving are:

1. Problem definition
2. Algorithm design / Algorithm specification
3. Algorithm analysis
4. Implementation
5. Testing
6. Maintenance



- **Step1.** Problem Definition What is the task to be accomplished?

  Ex: Calculate the average of the grades for a given student
- **Step2.**Algorithm Design / Specifications: Describe: in natural language / pseudo-code / diagrams / etc
- **Step3.** Algorithm analysis Space complexity - How much space is required

  Time complexity - How much time does it take to run the algorithm Computer Algorithm An algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some tasks which, given an initial state terminate in a defined end-state The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structures.
- **Steps 4,5,6:** Implementation, Testing, Maintainance
- Implementation: Decide on the programming language to use C, C++, Lisp, Java, Perl, Prolog, assembly, etc. , etc. Write clean, well documented code
- Test, test, test Integrate feedback from users, fix bugs, ensure compatibility across different versions
- Maintenance. Release Updates,fix bugs

PSEUDOCODE

- Algorithm can be represented in Text mode and Graphic mode
- Graphical representation is called Flowchart
- Text mode most often represented in close to any High level language such as C, Pascal Pseudocode.
- Pseudocode:
  - High-level description of an algorithm.
  - More structured than plain English.
  - Less detailed than a program.
  - Preferred notation for describing algorithms.
  - Hides program design issues.

Example of Pseudocode: To find the max element of an array

Algorithm *arrayMax*(A, n)
Input array A of n integers
Output maximum element of A
currentMax $\leftarrow A[0]$
for $i \leftarrow 1$ to $n - 1$ do
if $A[i] >$ *currentMax* then
    currentMax $\leftarrow A[i]$

return currentMax

- Control flow
- if … then … [else …]
- while … do …
- repeat … until …
- for … do …
- Indentation replaces braces
- Method declaration
- Algorithm *method* (*arg* [, *arg*…])
-     Input …
-     Output …
- Method call
- *var.method* (*arg* [, *arg*…])
- Return value
- return *expression*
- Expressions
- Assignment (equivalent to =)
- Equality testing (equivalent to ==)
- n^2 Superscripts and other mathematical formatting allowed

PERFORMANCE ANALYSIS:

- What are the Criteria for judging algorithms that have a more direct relationship to performance?
- computing time and storage requirements.
- Performance evaluation can be loosely divided into two major phases:
  - a priori estimates(performance analysis)
  - a posteriori testing(performance measurement).
- refer as performance analysis and performance measurement respectively
- The space complexity of an algorithm is the amount of memory it needs to run to completion.
- The time complexity of an algorithm is the amount of computer time it needs to run to completion.

Space Complexity:

- Algorithm sum(a,n)
- {
- s=0.0;
- for I=1 to n do
- s= s+a[I];
- return s;
- }

1. The problem instances for this algorithm are characterized by n,the number of elements to be summed. The space needed d by ‘n’ is one word, since it is of type integer.
2. The space needed by ‘a’a is the space needed by variables of tyepe array of floating point numbers.
3. This is atleast ‘n’ words, since ‘a’ must be large enough to hold the ‘n’ elements to be summed.
4. So,we obtain Ssum(n)>=(n+s)
5. [n for a[],one each for n,I a& s]

Time Complexity

```
Algorithm:
Algorithm sum(a,n)
{
s= 0.0;
count = count+1;
for I=1 to n do
{
count =count+1;
s=s+a[I];
count=count+1;
}
count=count+1;
count=count+1;
return s;
}
```

- This is done so that each time a statement in the original program is executes count is incremented by the step count of that statement.
- If the count is zero to start with, then it will be 2n+3 on termination. So each invocation of sum execute a total of 2n+3 steps.

Statement	*Steps per execution*	*Frequency*	*Total*
1. Algorithm Sum(a,n)	0	-	0
2.{	0	-	0
3. S=0.0;	1	1	1
4. for I=1 to n do	1	n+1	n+1
5. s=s+a[I];	1	n	n
6. return s;	1	1	1
7. }	0	-	0
Total			2n+3

Complexity of Algorithms

- The complexity of an algorithm M is the function f(n) which gives the running time and/or storage space requirement of the algorithm in terms of the size ‘n’ of the input data. Mostly, the storage space required by an algorithm is simply a multiple of the data size ‘n’.
- Complexity shall refer to the running time of the algorithm.
- The function f(n), gives the running time of an algorithm, depends not only on the size ‘n’ of the input data but also on the particular data. The complexity function f(n) for certain cases are:
  1. **Best Case** : The minimum possible value of f(n) is called the best case.
  2. **Average Case :** The expected value of f(n).
  3. **Worst Case :** The maximum value of f(n) for any key possible input.

How to analyse an Algorithm?

Let us form an algorithm for Insertion sort (which sort a sequence of numbers). The pseudo code for the algorithm is give below.

Pseudo code for insertion Algorithm:

Identify each line of the pseudo code with symbols such as C1, C2 ..

PSeudocode for Insertion Algorithm	Line Identification
for j=2 to A length	C1
key=A[j]	C2
//Insert A[j] into sorted Array A[1.....j-1]	C3
i=j-1	C4
while i>0 & A[j]>key	C5
A[i+1]=A[i]	C6
i=i-1	C7
A[i+1]=key	C8

Let Ci be the cost of ith line. Since comment lines will not incur any cost C3=0.

Cost	**No. Of times Executed**
C1	N
C2	n-1
C3=0	n-1
C4	n-1
C5	$\sum_{j=2}^{n-1} t_j$
C6	$\sum_{j=2}^{n} t_j - 1$
C7	$\sum_{j=2}^{n} t_j - 1$
C8	n-1

Running time of the algorithm is:

T(n)=C1n+C2(n-1)+0(n-1)+C4(n-1)+C5($\sum_{j=2}^{n-1} t_j$)+C6($\sum_{j=2}^{n} t_j - 1$)+C7($\sum_{j=2}^{n} t_j - 1$)+
C8(n-1)

Best case:

It occurs when Array is sorted. All tj values are 1.

T(n)=C1n+C2(n-1)+0 (n-1)+C4(n-1)+C5($\sum_{j=2}^{n-1} 1$) +C6($\sum_{j=2}^{n} 1-1$)+C7($\sum_{j=2}^{n} 1-1$)+ C8(n-1)

=C1n+C2 (n-1) +0 (n-1) +C4 (n-1) +C5 + C8 (n-1)

= (C1+C2+C4+C5+ C8) n-(C2+C4+C5+ C8)

· Which is of the form an+b.

→· Linear function of n.

→So, linear growth.

Worst case:

It occurs when Array is reverse sorted, and tj =j.

$$T(n)=C1n + C2(n-1)+0\ (n-1)+C4(n-1)+C5(\sum_{j=2}^{n-1} j) +C6(\sum_{j=2}^{n} j - 1\ )+C7(\sum_{j=2}^{n} j - 1\ ) + C8(n-1)$$

$$=C1n+C2(n-1)+C4(n-1)+C5(\frac{n(n-1)}{2} - \ 1) +C6(\sum_{j=2}^{n} \frac{n(n-1)}{2})+C7(\sum_{j=2}^{n} \frac{n(n-1)}{2})+ C8(n-1)$$

which is of the form an^2+bn+c

Quadratic function. So in worst case insertion set grows in n2.

Order of growth:

It is described by the highest degree term of the formula for running time. (Drop lower-order terms. Ignore the constant coefficient in the leading term.)

Example: We found out that for insertion sort the worst-case running time is of the form an2+ bn + c.

ASYMPTOTIC NOTATION

- Formal way notation to speak about functions and classify them
- The following notations are commonly use notations in performance analysis and used to characterize the complexity of an algorithm:

    1. Big–OH (O) ,
    2. Big–OMEGA (Ω),
    3. Big–THETA (Θ) and
    4. Little–OH (o)

Asymptotic Analysis of Algorithms:

- Our approach is based on the asymptotic complexity measure. This means that we don't try to count the exact number of steps of a program, but how that number grows with the size of the input to the program.
- That gives us a measure that will work for different operating systems, compilers and CPUs. The asymptotic complexity is written using big-O notation.

1. It is a way to describe the characteristics of a function in the limit.
2. It describes the rate of growth of functions.
3. Focus on what's important by abstracting away low order terms and constant factors.
4. It is a way to compare "sizes" of functions:

$O \approx \leq$, $\Omega \approx \geq$, $\Theta \approx =$, $o \approx <$, $\omega \approx >$

Time complexity	Name	Example
O(1)	Constant	Adding an element to the front of a linked list
O(logn)	Logarithmic	Finding an element in a sorted array
O (n)	Linear	Finding an element in an unsorted array
O(nlog n)	Linear	Logarithmic Sorting n items by 'divide-and-conquer'-Mergesort
$O(n^2)$	Quadratic	Shortest path between two nodes in a graph
$O(n^3)$	Cubic	Matrix Multiplication
$O(2^n)$	Exponential	The Towers of Hanoi problem

- **Big ‘oh’:** the function f(n)=O(g(n)) iff there exist positive constants c and no such that

  f(n)<=c*g(n) for all n, n>= no.

- **Omega:** the function f(n)=(g(n)) iff there exist positive constants c and no such that

  f(n) >= c*g(n) for all n, n >= no.

- **Theta:** the function f(n)=(g(n)) iff there exist positive constants c1,c2 and no such that c1

  g(n) <= f(n) <= c2 g(n) for all n, n >= no

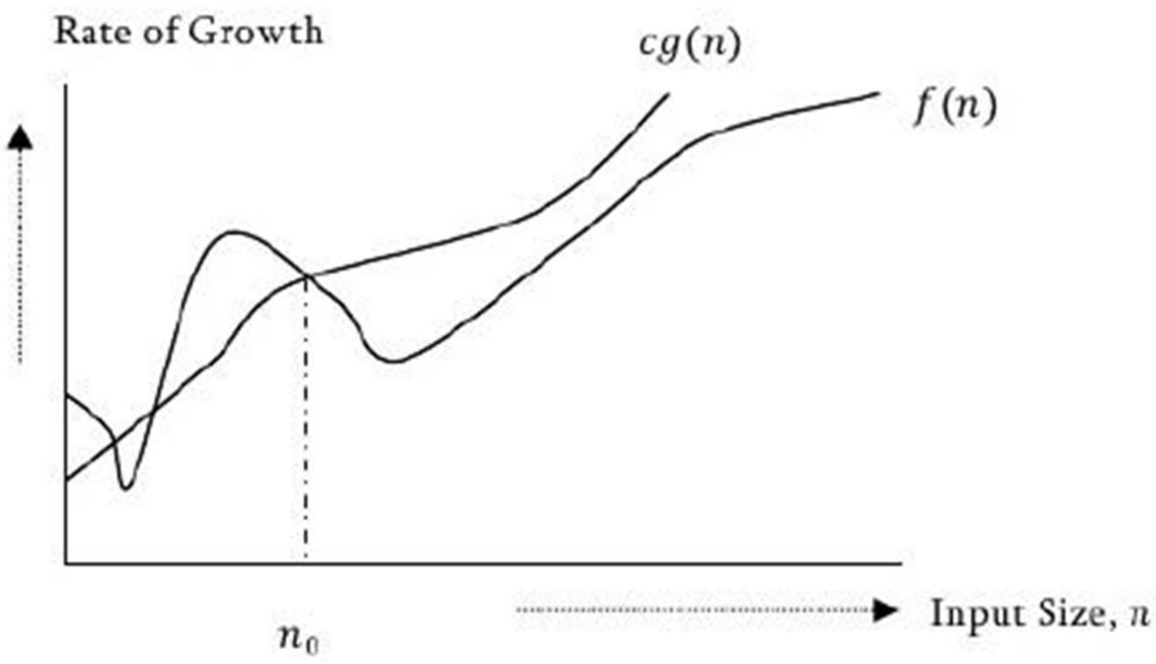
Big-O Notation

- This notation gives the tight upper bound of the given function. Generally we represent it as f(n) = O(g (11)). That means, at larger values of n, the upper bound off(n) is g(n).

For example,

if f(n) = n4 + 100n2 + 10n + 50 is the given algorithm, then n4 is g(n). That means g(n) gives the maximum rate of growth for f(n) at larger values of n.

<u>O —notation</u> defined as O(g(n)) = {f(n): there exist positive constants c and no such that 0 <= f(n) <= cg(n) for all n >= no}. g(n) is an asymptotic tight upper bound for f(n). Our objective is to give some rate of growth g(n) which is greater than given algorithms rate of growth f(n).



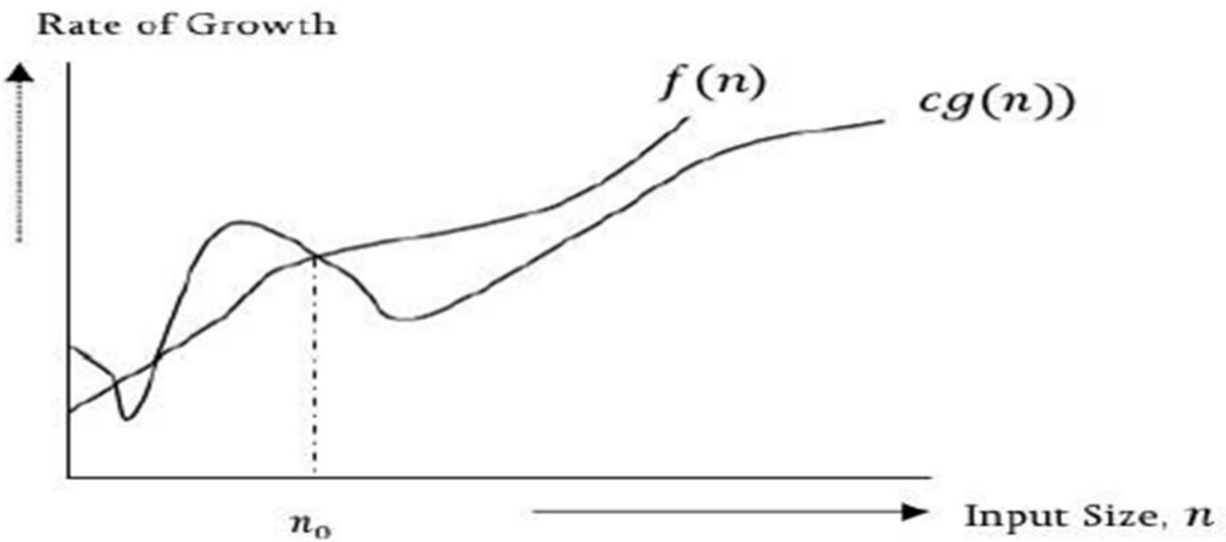
Note Analyze the algorithms at larger values of n only What this means is, below no we do not care for rates of growth.

Omega— Ω notation

- Similar to above discussion, this notation gives the tighter lower bound of the given algorithm and we represent it as f(n) = Ω (g(n)). That means, at larger values of n, the tighter lower bound of f(n) is g.

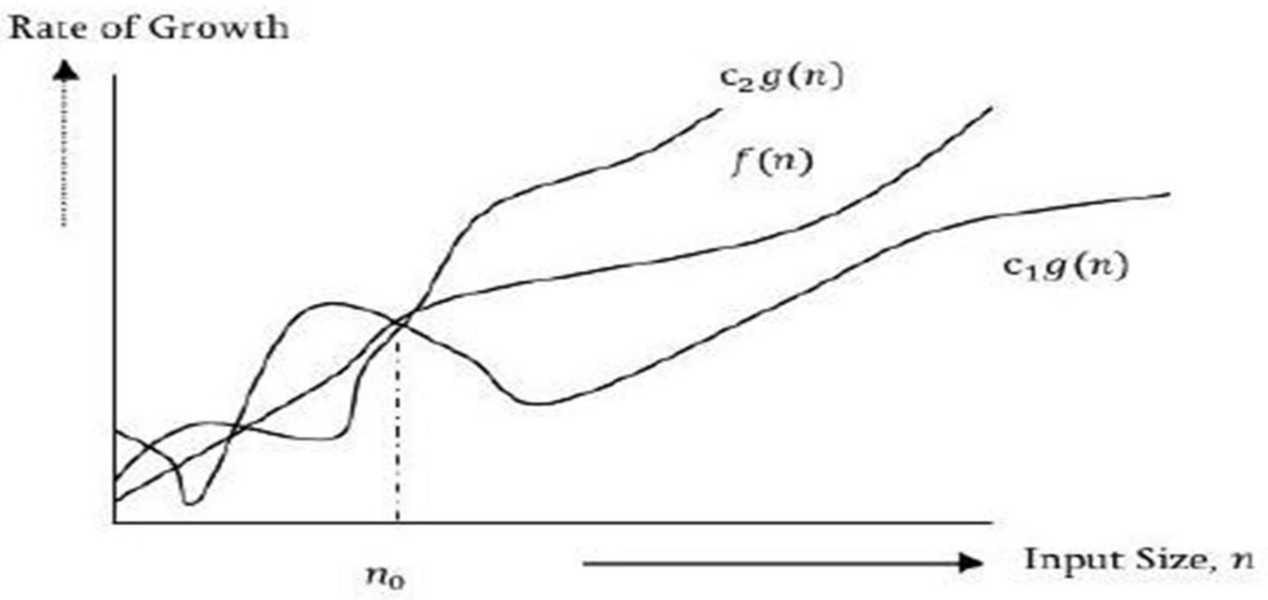
For example, if f(n) = 100n2+ 10n + 50, g(n) is Ω(n2).

- The . Ω. notation as be defined as Ω (g (n)) = {f(n): there exist positive constants c andno such that 0 <= cg (n) <= f(n) for all n >= no}. g(n) is an asymptotic lower bound for f(n). Ω (g (n)) is the set of functions with smaller or same order of growth as f(n).



Theta- Θ notation

- This notation decides whether the upper and lower bounds of a given function are same or not. The average running time of algorithm is always between lower bound and upper bound.
- If the upper bound (O) and lower bound (Ω) gives the same result then Θ notation will also have the same rate of growth. As an example, let us assume that f(n) = 10n + n is the expression. Then, its tight upper bound g(n) is O(n). The rate of growth in best case is g (n) = 0(n).



Little Oh Notation

- The little Oh is denoted as o. It is defined as : Let, f(n} and g(n} be the non negative functions then

$$\lim_{n \to \propto} \frac{f(n)}{g(n)} = 0$$

such that f(n}= o(g{n)} i.e f of n is little Oh of g of n.

f(n) = o(g(n)) if and only if f'(n) = o(g(n)) and f(n) != Θ {g(n))