CE355 Design & Analysis of Algorithms

Credits and Hours:

Teaching Scheme	Theory	Practical	Tutorial	Total	Credit
Hours/week	4	2	-	6	**5**
Marks	100	50	-	150	

Dhaval Bhoi,
Assistant Professor,
U & P U. Patel Department of Computer Engineering,
CSPIT, CHARUSAT
E-mail: dhavalbhoi.ce@charusat.ac.in

Text Books and Reference Books

Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest and Clifford Stein, MIT Press	**Fundamental of Algorithms by Gills Brassard, Paul Bratley, Pentice Hall of India.**
	Fundamental of Computer Algorithms by Ellis Horowitz, Sartazsahni and sanguthevar Rajasekarm, Computer Sci.P.
	Design & Analysis of Algorithms by P H Dave & H B Dave, Pearson Education.

Suggested Courses

Design and analysis of algorithms [NPTEL]

Introduction to algorithms and analysis [NPTEL]

Importance of the Subject

1. One of the most Important/Core subject offered by ALL Universities
   - GATE (10%) [Competitive Examination]
   - UGC-NET Examination (10 Marks)
     - **[It is conducted for determining the eligibility of Indian nationals for the Eligibility for Assistant Professor only or Junior Research Fellowship & Assistant Professor Both, in Indian Universities and Colleges]**
   - Placement & Job Interview[Companies like Google, Facebook, Microsoft] and Entrance Exam]
2. Google Search, Google Map, Top Trend on YouTube
3. Prerequisite for many subject
4. Help in Solving Real Life Problems

What is an Algorithm? How is it different from Program?

An Algorithm is a step-by-step procedure for solving a computational problems in a finite amount of time.

Algorithm	**Program**
Written @ Design Time	Written @ Implementation Time
Domain Knowledge Required	Mainly works as a Programmer
Any Language [English like]	Programming languages
Not Dependent on H/W or OS	Dependent on H/W or OS
Analyze	Testing

Example of an Algorithm

Sum of Two Numbers

S1: Read First Number

S2: Read Second Number

S3: Sum=A+B

S4: Print(Sum)

Characteristics of An Algorithm

Input: There are zero or more quantities are externally supplied

Output: At-least one quantity is produced

Definite: Each instruction must be clear and unambiguous

Finiteness: Algorithm will terminate after finite number of steps

Effectiveness: Every instruction must be definite, feasible and effective.

Analysis Algorithm & Types

- It is a process of comparing two or more algorithms with respect to Time and Space

- **Priori** [Before Execution -Independent of Hardware] and **Posterior**[After Execution - Dependent on Hardware]

Example

Sum of Two Numbers

S1: Read First Number

S2: Read Second Number

S3: Sum=A+B

S4: Print(Sum)

Priori	Posterior [Dependency is on H/W Used]
4 Instruction	0.4 seconds ->0.3 seconds->0.1 seconds

Priori Analysis Vs. Posteriori Analysis

Priori Analysis (theoretical approach)	Posteriori Analysis (empirical approach)
Algorithm	Program
Independent of Language	Language Dependent
Hardware Independent	Hardware Dependent
Time and Space Function [Not exact time]	Watch time and Bytes

DESIGN AND ANALYSIS OF ALGORITHMS

- What is more important?
  - Time Complexity
  - Space Complexity

Time Efficiency Evaluation

Instance Size	Time
n (Check for n=10)	$10^{-4}*2^{n}$ Seconds(1/10th of a second)
n=20	2 minutes
n=30	more than a day
n=38	A year

Time Efficiency Evaluation with 100X faster Device

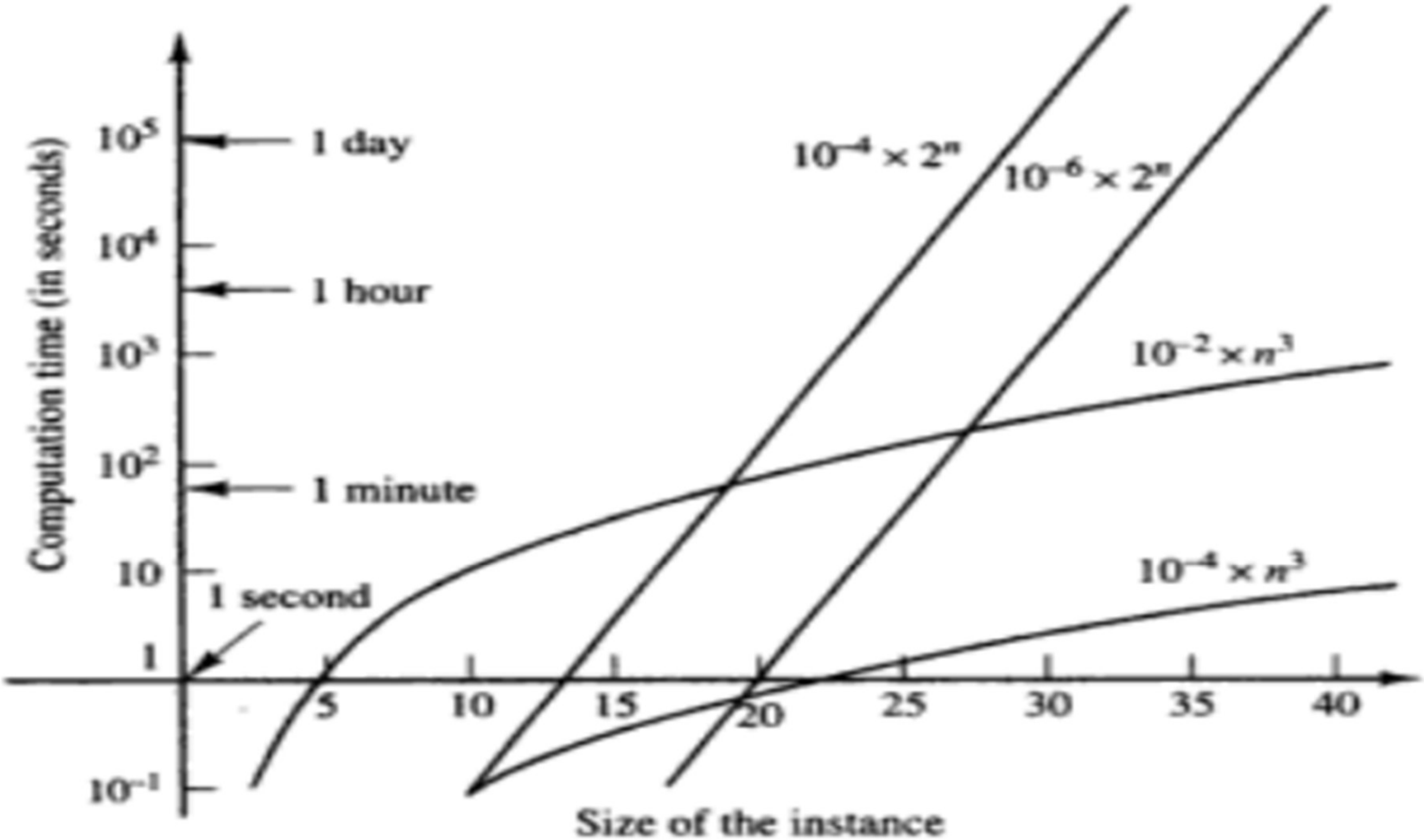
Instance Size	Time
n (Check for n=10)	$10^{-6}*2^{n}$ Seconds(?)
n=20	?
n=30	?
n=45	A year

Lets change -> Better Algorithm

Instance Size	Time
n (Check for n=10)	$10^{-2}*n^3$ Seconds(10 seconds)
n=20	1 to 2 minutes
n=30	4.5 minutes
n=1500	A year

n^2 and nlogn

when n is too small	marginal difference in execution time
n=50	nlogn is twice faster then n^2
n=100	3X faster
n=1000	insertion sort takes >3 sec quick sort takes ⅕ seconds
5000	insertion sort takes 1.5 minutes quick sort takes >1 second
100000	insertion sort takes 9.30 Hrs quick sort takes 30seconds



Computation time (in seconds)
10^5
1 day
10^4
1 hour
10^3
10^2
1 minute
10
1 second
1
10^{-1}
5
10
15
20
25
30
35
40
Size of the instance
$10^{-4} \times 2^n$
$10^{-6} \times 2^n$
$10^{-2} \times n^3$
$10^{-4} \times n^3$

Analysing Control Statement

Example: 1

```
        C1:      b = a * c
```

Here cost of C1= O(1) as it executes once

Example: 2

```
 for i=1 to n   C1: n+1 [executed n time + 1 time to check wrong
                condition]
     b = a * c  C2: n [executed n time]
```

T(n) =C1+C2

=n+1+n = 2n+1

=O(n)

Analysing Control Statement

Example: 3

```
for i=1 to n                             C1: n+1
   for j=1 to n   C2: n+1  }
       b = a * c  C3: n    }  n
   end
end
```

$T(n) = C1+C2+C3$

$= n+1+n(n+1)+n(n) = 2n^2+2n+1$

$= O(n^2)$

Types of Time Function
Classes of Function

$O(1)$ - Constant, $f(n)=2$ or $f(n)=2000$ or $f(n)=5$

$O(\log n)$- Logarithmic, base can be any

$O(n)$-Linear, e.g. $f(n)=n+3$, $f(n)=(n/3000)+6$

$O(n^2)$-Quadratic

$O(n^3)$-Cubic

$O(2^n)$-Exponential OR 3^n or n^n

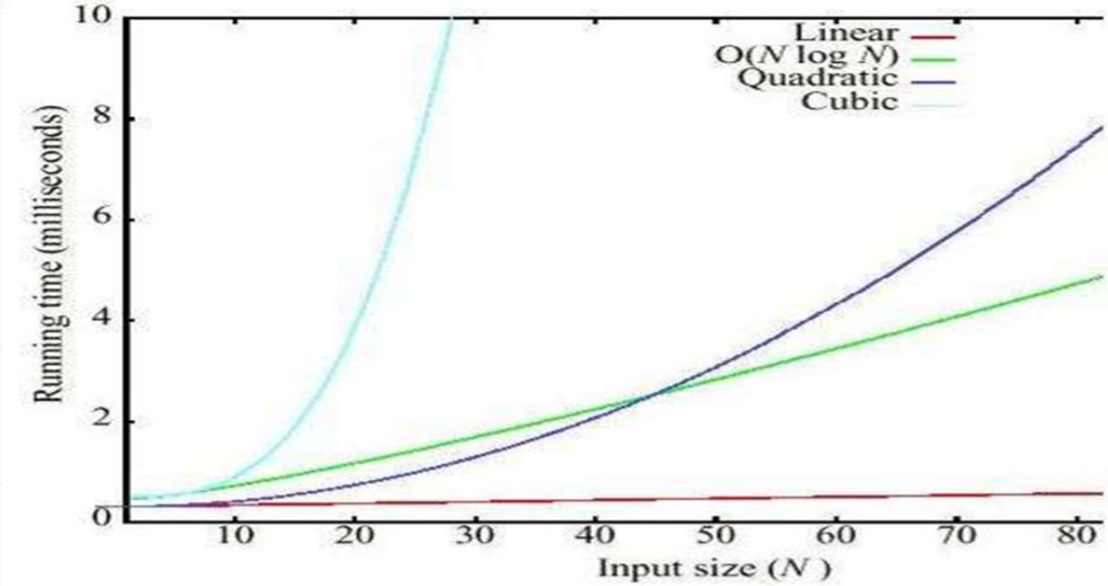
Compare Class of Function

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$

logn	n	n^2	2^n
0	1	1	2
1	2	4	4
2	4	16	16
3	8	64	256
?	9	81	512

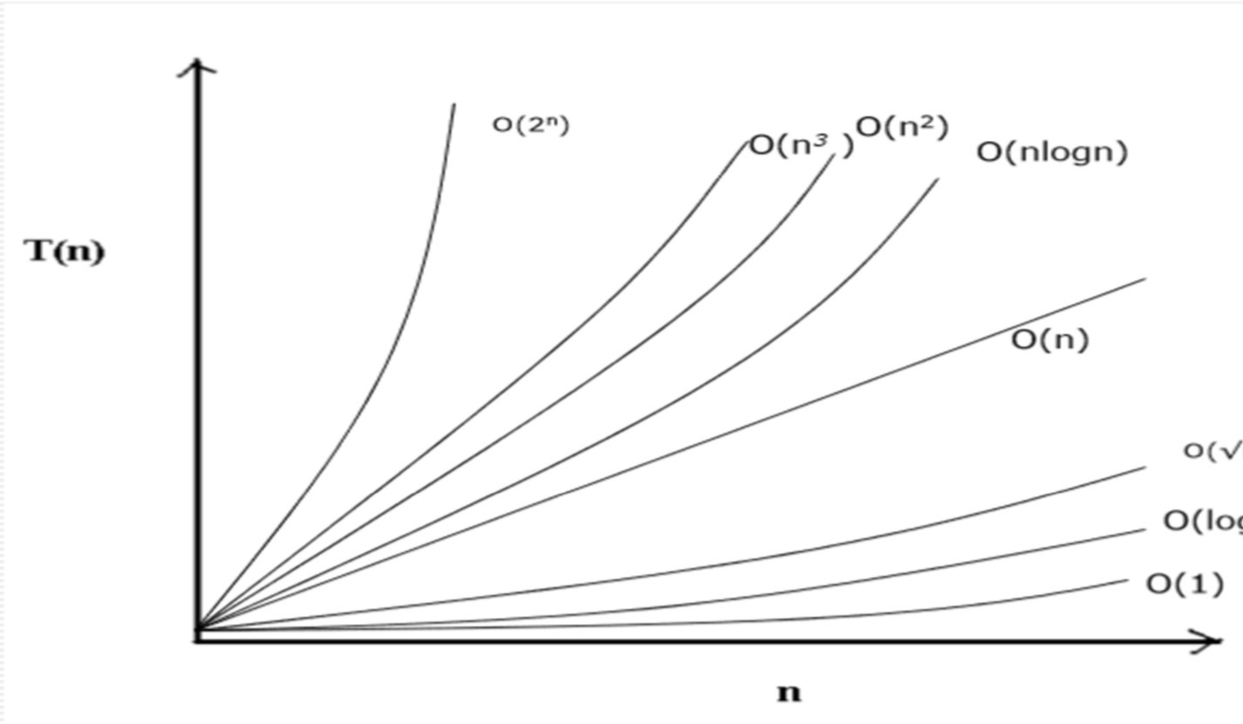
n^k<2^n

Running time for small inputs



Running times for small inputs

Common plots of O( )



Comparing Function to Analyze Time Complexity

$n^2 < n^3$

$2^n > n^2$

$3^n > 2^n$

Individual Exercises

What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

Required,

Smallest positive n Such that $100n^2 < 2^n$

Solving using the trial and error method

n	$100n^2$	2^n
5	2500	32
10	10000	1024
11	12100	2048
13	16900	8192
14	**19600**	**16284**
15	22500	32768
16	25600	65536

At n=15, the algorithm runs faster at $100n^2$ than the one with a run time of 2^n

Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of *size* n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64^n \lg n$ steps. For which values of n does insertion sort beat merge sort?

Insertion sort = $8n^2$ steps
Merge sort= 64nlogn steps
Required to find: n such that insertion sorts > merge sorts

Solution:
$(8n^2)/8$= 64nlogn/8

n^2/n = 8nlogn/n
n = 8logn

solving using trial and error method

n	8logn
2	8
4	16
8	24
16	32
32	40
40	42.56
42	43.139
43	43.410
44	**43.675**
46	44.188

At n=44, insertion sort runs in $8(44)^2$ = 15488 steps
Merge sort runs 15373.8 steps
Therefore, at n=44 insertion sorts beat merge sorts

How to write and Analyze Algorithm

```
Algorithm Swap(a,b)
{
        temp=a;
        a=b;
        b=temp;
}
```

1. Time
2. Space
3. Network
4. Power
5. CPU Registers

Types of Time Function
Classes of Function

O(1) - Constant, $f(n)=2$ or $f(n)=2000$ or $f(n)=5$

O(log n)- Logarithmic, base can be any

O(n)-Linear, e.g. $f(n)=n+3$, $f(n)=(n/3000)+6$

$O(n^2)$-Quadratic

$O(n^3)$-Cubic

$O(2^n)$-Exponential OR 3^n or n^n

How to write and Analyze Algorithm

```
Algorithm Swap(a,b)
{
        temp=a; // 1 unit of time
        a=b;     //1 unit of time
        b=temp;// 1 unit of time
}
```

Time Complexity f(n)=3, O(1)

Space Complexity

Variable	Space
a	1
b	1
temp	1
S(n)	=3 OR O(1)

FCM[Frequency Count Method] to Analyze Algorithm

```
Algorithm Sum(A,n)
{
        s=0;// 1 unit of time
        for(i=0; i<n;i++)// n+1 unit of time
        {
                s=s+A[i];// n unit of time
        }
        return s;// 1 unit of time
}
```

Time Complexity $f(n)=2n+3$, $f(n)=O(n)$

Space Complexity

Variable	Space
A	n
n	1
s	1
i	1
S(n)	=n+3 OR O(n)

FCM [Frequency Count Method] to Analyze Algorithm

```
Algorithm ADD(A,B,n)
{
        for(i=0; i<n;i++)// n+1 unit of time
                for(j=0; i<n;j++)// n*(n+1) unit of time
        {
                C[i,j]=a[i,j]+b[i,j];// n*n unit of time
        }
}
```

Time Complexity $f(n)=2n^2+2n+1$,

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

Space Complexity

Variable	Space
A	n^2
B	n^2
C	n^2
n	1
i	1
j	1
S(n)	$=3n^2+3$ OR $O(n^2)$

FCM[Frequency Count Method] to Analyze Algorithm

```
Algorithm Multiply(A,B,n)
{
        for(i=0; i<n;i++)
  {
        for(j=0; i<n;j++)
        {
                C[i,j]=0;
          for(k=0; k<n;k++)
          {
            C[i,j]=C[i,j]+a[i,k]*b[k,j];
        }
        }
        }
}
```

Time Complexity f(n)=?, f(n)=O(?)

Space Complexity

Variable	Space
A	?
B	?
C	?
n	?
i	?
j	?
k	?
S(n)	=? OR O(?)

FCM [Frequency Count Method] to Analyze Algorithm

```
Algorithm Multiply(A,B,n)
{
        for(i=0; i<n;i++)// n+1 unit of time
        {
        for(j=0; i<n;j++)// n*(n+1) unit of time
        {
                C[i,j]=0;// n*n unit of time
          for(k=0; k<n;k++)// (n+1)*n*n unit of time
          {
           C[i,j]=C[i,j]+a[i,k]*b[k,j];// n*n*n unit of time
        }
        }
        }
}
```

Time Complexity $f(n)=2n^3+3n^2+2n+1$,

$f(n)=O(n^3)$

Space Complexity

Variable	Space
A	n^2
B	n^2
C	n^2
n	1
i	1
j	1
k	1
S(n)	$=3n^2+4$ OR $O(n^2)$

Examples

for(i=0; i<n;i++)
{
statement;
}
	O(n)

for(i=n; i>0;i--)
{
statement;
}
	O(n)

for(i=1; i<n;i=i+2)
{
statement;
}
	(?)

for(i=1; i<n;i=i+20)
{
statement;
}
	f(n)=n/20, f(n)=O(n)

for(i=0; i<n;i++)
{
for(j=0;j<i;j++)
{statement;}
}
	0+1+2+3+...+n=[n(n+1)]/2
	so, f(n)=O(n^2)

Some More Examples

p=0
for(i=1; p<=n;p++)
{
p=p+i;
}
	O(square_root(n))

for(i=1; i<n;i=i*2)
{
statement;
}
	O(logn)

for(i=n; i>=1;i=i/2)
{
statement;
}
	O(logn)

for(i=0; i*i<n;i++)
{
statement;
}
	O(square_root(n))

for(i=0; i<n;i++)
{
statement;
}
for(j=0;j<n;j++)
{
statement;
 	
}
	O(n) as f(n)=n+n=2n

Some more example Proof

```
p=0;
for(i=1; p<=n; i++)
{
    p=p+i;
}
```

i	p
1	$0+1=1$
2	$1+2=3$
3	$1+2+3$
4	$1+2+3+4$
⋮	
k	$1+2+3+4+\dots+k$

Assume $p > n$

$$\because p = \frac{k(k+1)}{2}$$

$$\frac{k(k+1)}{2} > n$$

$$k^2 > n$$

$$k > \sqrt{n}$$

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

Some more example proof

```
for(i=1; i<n; i=i*2)
{
    stmt;
}
```

1

$1 \times 2 = 2$

$2 \times 2 = 2^2$

$2^2 \times 2 = 2^3$

$\vdots$

2^k

Assume $i >= n$

$\because\ i = 2^k$

$\therefore\ 2^k >= n$

$2^k = n$

$k = \log_2 n$

```
for(i=1; i<n; i=i*2)
{
    stmt;
}
```

$i = 1 \times 2 \times 2 \times 2 \cdots = n$

$2^k = n$

$k = \log_2 n$

```
for(i=1; i<=n; i++)
{
    stmt;
}
```

$i = \underbrace{1 + 1 + 1 + 1 \cdots + 1}_{} = n$

$k = n$

```
for(i=n; i>=1; i=i/2)
{
    stmt;
}
```

i
n
$\frac{n}{2}$
$\frac{n}{2^2}$
$\frac{n}{2^3}$
$\vdots$
$\frac{n}{2^k}$

Assume $i < 1$

$$\therefore \frac{n}{2^k} < 1$$

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

$$O(\log_2 n)$$

```
p=0
for(i=1; i<n; i=i*2)
{
    p++;                    p=log n
}
for(j=1; j<p; j=j*2)
{
    stmt;                   log p
}
```

$$O(\log \log n)$$

```
for(i=0; i<n; i++) ———————— n
{
    for(j=1; j<n; j=j*2) —— n×logn
    {
        stmt; ————————————— n×logn
    }
}
```

$$2n\log n + n$$

$$O(n\log n)$$

In General

`for(i=0; i<n; i++)` — $O(n)$

`for(i=0; i<n; i=i+2)` — $\frac{n}{2}$ $O(n)$

`for(i=n; i>1; i--)` — $O(n)$

`for(i=1; i<n; i=i*2)` — $O(\log_2 n)$

`for(i=1; i<n; i=i*3)` — $O(\log_3 n)$

`for(i=n; i>1; i=i/2)` — $O(\log_2 n)$

$\frac{n}{2}$ — $O(n)$

$\frac{n}{200}$ — $O(n)$

Analysis of if and while

```
i=0; ———— 1
      10
while(i<n) — n+1
{
    stmt; ———— n
    i++; ———— n
}
```

$f(n) = 3n+2$

$O(n)$

```
for(i=0; i<n; i++) — n+1
     *    n+1   **
{
    stmt; ———— n
}
```

$f(n) = 3n+2$

$f(n) = 2n+1$

$O(n)$

```
a=1;
while(a<b)
{
    stmt;
    a=a*2;
}
```

a
1
$1\times 2=2$
$2\times 2=2^2$
$2^2\times 2=2^3$
$\vdots$
2^k

Terminate

$a \geq b$

$\because a=2^k$

$2^k \geq b$

$2^k=b$

$k=\log_2 b$

```
for(i=1; i<n; i=i*2)
{
    stmt;
}
```

$O(\log n)$

```
i=1;
k=1;
while(k<n)
{
    stmt;
    k=k+i;
    i++;
}
```

$O(\sqrt{n})$

i	k
1	1
2	$1+1=2$
3	$2+2$
4	$2+2+3$
5	$2+2+3+4$
⋮	⋮
m	$2+2+3+4+\dots+m$

$$\frac{m(m+1)}{2}$$

$$k \geq n$$

$$\frac{m(m+1)}{2} \geq n$$

$$m^2 \geq n$$

$$m = \sqrt{n}$$

```
for(k=1, i=1; k<n; i++)
{
    stmt;
    k=k+i;
}
```

```
while(m!=n)
{
    if(m>n)
        m=m-n;
    else
        n=n-m;
}
```

m=16	n=2	
14	2	
12	2	
10	2	
8	2	$\frac{16}{2}$
6	2	
4	2	$\frac{n}{2}$
2	2	

min $O(1)$ $O(n)$

```
Algorithm Test(n)
{
    if(n>5)
    {


        for(i=0;i<n;i++)
        {
            printf("%d",i); — n
        }
    }
}
```

SINGLE TASKING

ZERO DISTRACTION

INTENSE FOCUS

EXTENDED PERIODS OF TIME

Asymptotic Notation

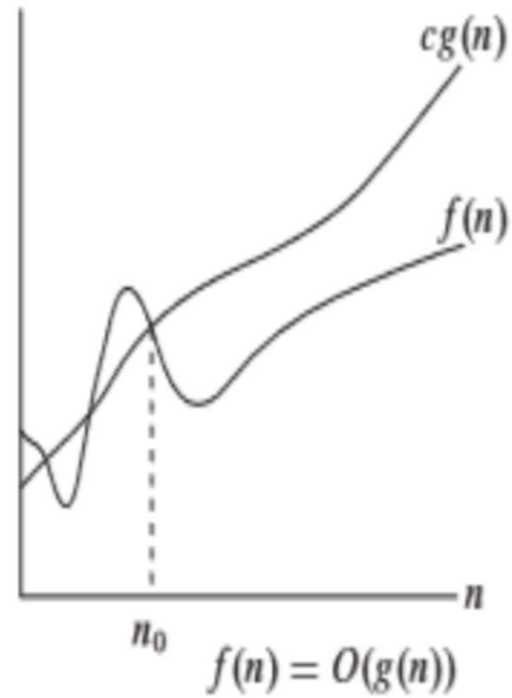
- Simple method for representing time complexity
    - Big-Oh : Upper Bound
    - Big-Omega : Lower Bound
    - Big-theta : Average Bound [To represent Exactness]

Big-O Notation

Big-O

Big-O, commonly written as **O**, is an Asymptotic Notation for the worst case, or ceiling of growth for a given function. It provides us with an ***asymptotic upper bound*** for the growth rate of runtime of an algorithm. Say `f(n)` is your algorithm runtime, and `g(n)` is an arbitrary time complexity you are trying to relate to your algorithm. `f(n)` is O(g(n)), if for some real constants c (c > 0) and n_0, `f(n)` <= c `g(n)` for every input size n (n > n_0).

$f(n) <= c*g(n)$, $c>0$ and $n_0>=1$, $n>=n_0$



Example

$f(n)=2n+3$

$2n+3<15n$, $n_0>=1$, where $c=15$

$2n+3<5n$, $n_0>=1$, where $c=5$

$f(n)=3n+2$, $g(n)=n$, Is $f(n)=O(g(n))$

$f(n)<=c*g(n)$, $c>0$ and $n_0>=1$, $n>=n_0$

$3n+2<=c*n$

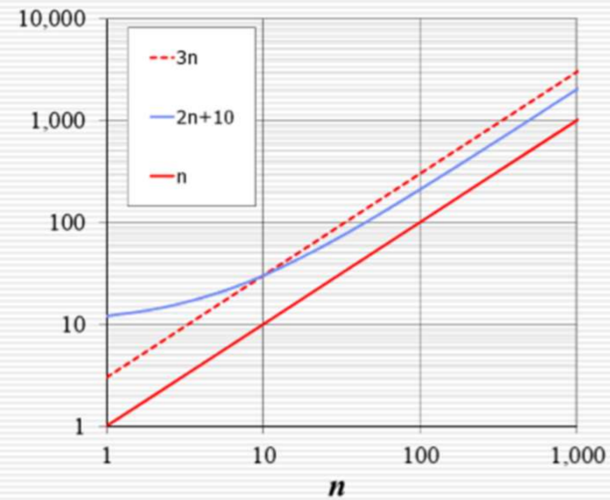
$3n+2<=4n$, $n_0>2$, $c=4$,

We can also say $f(n)$ is in order of (n^2), (n^3) (2^n)

Big-Oh Notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for $n \geq n_0$
- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$



More Big-Oh Examples

- 7n-2

  7n-2 is O(n)
  need $c > 0$ and $n_0 \geq 1$ such that $7n-2 \leq c \cdot n$ for $n \geq n_0$
  this is true for $c = 7$ and $n_0 = 1$

- $3n^3 + 20n^2 + 5$

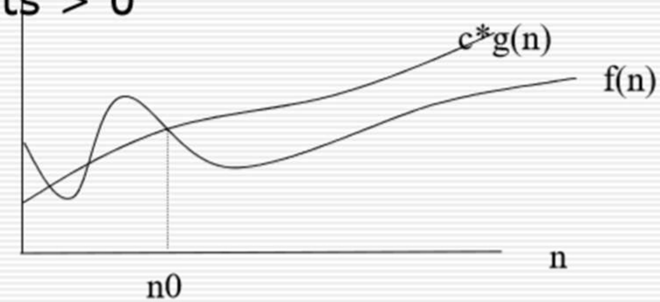
  $3n^3 + 20n^2 + 5$ is $O(n^3)$
  need $c > 0$ and $n_0 \geq 1$ such that $3n^3 + 20n^2 + 5 \leq c \cdot n^3$ for $n \geq n_0$
  this is true for $c = 4$ and $n_0 = 21$

- 3 log n + log log n

  3 log n + log log n is O(log n)
  need $c > 0$ and $n_0 \geq 1$ such that $3 \log n + \log \log n \leq c \cdot \log n$ for $n \geq n_0$
  this is true for $c = 4$ and $n_0 = 2$

Big-Oh Notation: Asymptotic Upper Bound

- T(n) = f(n) = O(g(n))
  - if f(n) <= c*g(n) for all n > n0, where c & n0 are constants > 0



- Example: T(n) = 2n + 5 is *O*(n). Why?
  - 2n+5 <= 3n, for all n >= 5
- T(n) = 5*n^2 + 3*n + 15 is *O*(n^2). Why?
  - 5*n^2 + 3*n + 15 <= 6*n^2, for all n >= 6

Big-Omega

Big-Omega, commonly written as **Ω**, is an Asymptotic Notation for the best case, or a floor growth rate for a given function. It provides us with an ***asymptotic lower bound*** for the growth rate of runtime of an algorithm.

`f(n)` is $\Omega(g(n))$, if for some real constants c ($c > 0$) and n_0 ($n_0 > 0$), `f(n)` is >= `c g(n)` for every input size n ($n > n_0$).

Big Omega

f(n)=3n+2, g(n)=n, Is f(n)=O(g(n))

f(n)>=c*g(n), c>0 and n_0>=1

3n+2>=c*n

3n+2>=4n, n_0>1, c=1,

Example 1

```
f(n) = 3log n + 100
g(n) = log n
```

Is `f(n)` O(g(n))? Is `3 log n + 100` O(log n)? Let's look to the definition of Big-O.

```
3log n + 100 <= c * log n
```

Is there some pair of constants c, n_0 that satisfies this for all n > n_0?

```
3log n + 100 <= 150 * log n, n > 2 (undefined at n = 1)
```

Yes! The definition of Big-O has been met therefore `f(n)` is O(g(n)).

Example 2

```
f(n) = 3*n^2
g(n) = n
```

Is `f(n)` O(g(n))? Is `3 * n^2` O(n)? Let's look at the definition of Big-O.

```
3 * n^2 <= c * n
```

Is there some pair of constants c, n_0 that satisfies this for all n > $_0$? No, there isn't. `f(n)` is NOT O(g(n)).

Theta

Theta, commonly written as **Θ**, is an Asymptotic Notation to denote the ***asymptotically tight bound*** on the growth rate of runtime of an algorithm.

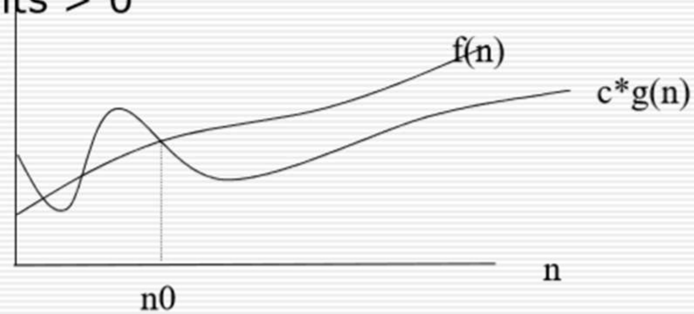
`f(n)` is Θ(g(n)), if for some real constants c1, c2 and n_0 (c1 > 0, c2 > 0, n_0 > 0), `c1 g(n)` is < `f(n)` is < `c2 g(n)` for every input size n (n > n_0).

∴ `f(n)` is Θ(g(n)) implies `f(n)` is O(g(n)) as well as `f(n)` is Ω(g(n)).

Feel free to head over to additional resources for examples on this. Big-O is the primary notation use for general algorithm time complexity.

Ω Notation: Asymptotic Lower Bound

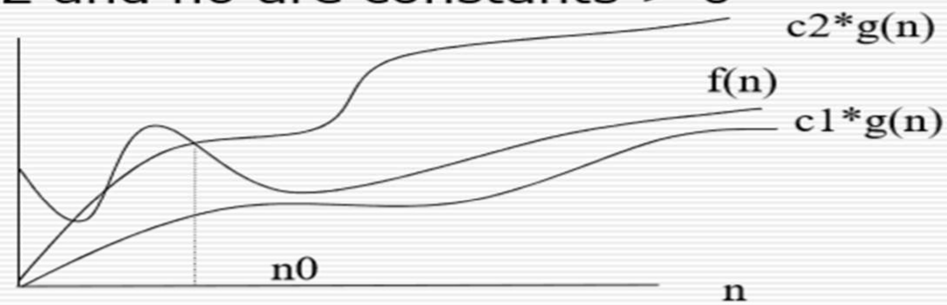
- T(n) = f(n) = Ω(g(n))
  - if f(n) >= c*g(n) for all n > n0, where c and n0 are constants > 0



- Example: T(n) = 2n + 5 is Ω(n). Why?
  - 2n+5 >= 2n, for all n > 0
- T(n) = $5*n^2 - 3*n$ is $\Omega(n^2)$. Why?
  - $5*n^2 - 3*n >= 4*n^2$, for all n >= 4

Θ Notation: Asymptotic Tight Bound

- $T(n) = f(n) = \Theta(g(n))$
  - if c1*g(n) <= f(n) <= c2*g(n) for all n > n0, where c1, c2 and n0 are constants > 0



- Example: T(n) = 2n + 5 is Θ(n). Why?

  2n <= 2n+5 <= 3n, for all n >= 5

- T(n) = $5*n^2 - 3*n$ is $\Theta(n^2)$. Why?
  - $4*n^2 <= 5*n^2 - 3*n <= 5*n^2$, for all n >= 4

Big-Oh and Growth Rate

- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

	$f(n)$ is $O(g(n))$	$g(n)$ is $O(f(n))$
$g(n)$ grows more	Yes	No
$f(n)$ grows more	No	Yes
Same growth	Yes	Yes

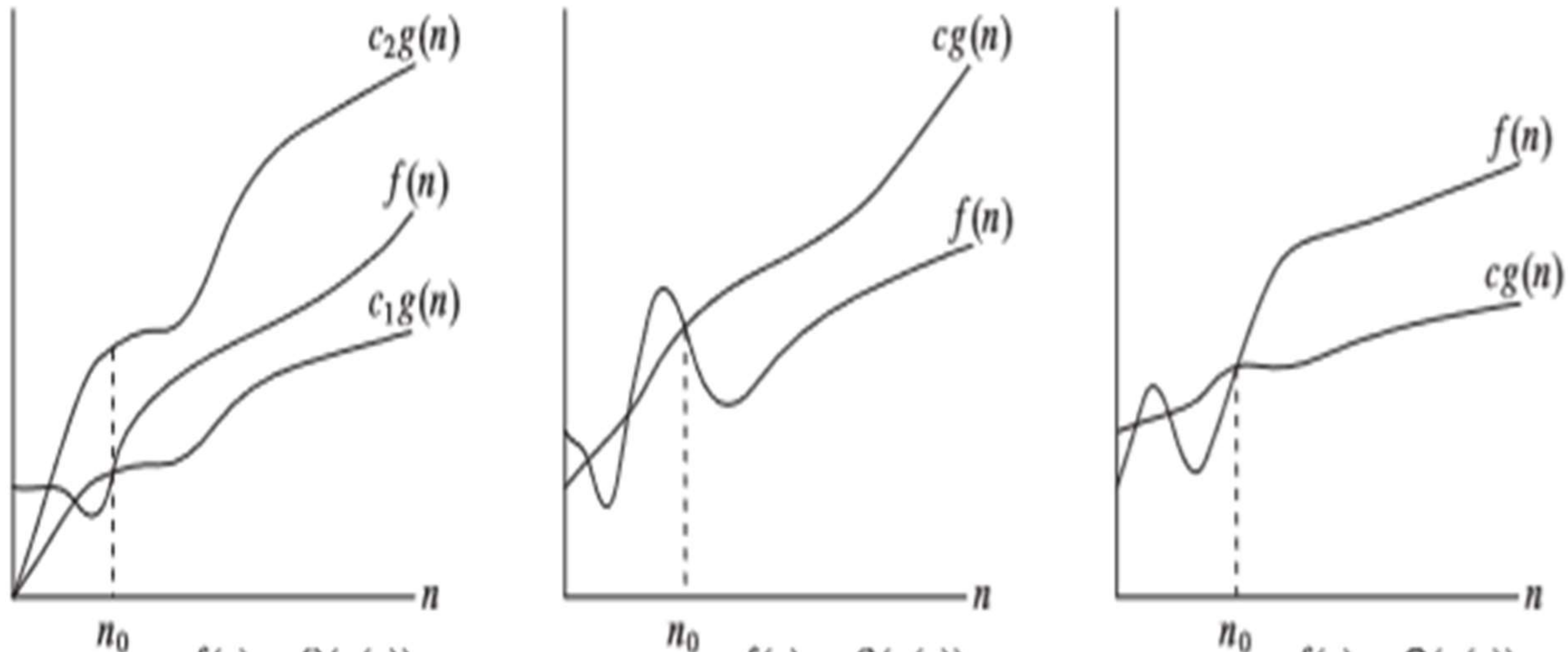
Properties of Asymptotic Notations

1. f(n)=O(g(n)), then [a*f(n)] is O(g(n))
2. f(n)=omega(g(n)), then [a*f(n)] is omega(g(n))
3. f(n)=theta(g(n)), then [a*f(n)] is theta(g(n))
4. Reflexive: f(n)=O(f(n))
5. Transitive: f(n)=O(g(n)) and g(n)=O(h(n)) then f(n)=O(h(n)), f(n)=n, g(n)=n^2, h(n)=n^3
6. Transpose Symmetric is true for O and Omega
7. Symmetric property is true only for theta notation, f(n)=n, g(n)=n^2
8. f(n)=O(g(n)), f(n)=omega(g(n)), then f(n)=theta(g(n))
9. f(n)=O(g(n)), d(n)=O(e(n)), then f(n)+d(n)=O(max(g(n),e(n)))
10. f(n)=O(g(n)), d(n)=O(e(n)), then f(n)*d(n)=O((g(n)*e(n)))

		Reflexive	Symmetric	Transitive
Big O	f(n)<=c*g(n)	YES	NO	YES
Big Omega	f(n)>=c*g(n)	YES	NO	YES
Big Theta	c1*g(n)<=f(n)<=c2*g(n)	YES	YES	YES
Small o	f(n)<c*g(n)	NO	NO	YES
Small omega(w)	f(n)>c*g(n)	NO	NO	YES

Complexity of An Algorithm

Worst Case Complexity: Maximum of the running times over all instances of a given size
Average Complexity: Average of the running times.
Best Case Complexity: Minimum of the running times over all instances of a given size.



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

(a) (b) (c)

Examples

Find upper bound, lower bound and tight bound range for the function:

$f(n)=10n^2+4n+2$

Lower Bound=$10n^2$ Tight bound=$10n^2$ Upper Bound=$11n^2$

Big- O

$10n^2+4n+2<=11n^2$

for n=1, 16<11, false

n=2, 50<=44, false

n=3 104<=99, false

n=4 178<=176 false

n=5 272<=275, is true for n>=5 and c=11

Big- Omega

$10n^2+4n+2>=10n^2$

for n=1, 16>=11, true

hence, $10n^2+4n+2=Omega(n^2)$

Big- theta

$10n^2 <= 10n^2+4n+2 <= 11n^2$

for n>=5, c1=10, c2=11, n>=5

Question 8: What does it mean when we say that an algorithm X is asymptotically more efficient than Y?

A. X will be a better choice for all inputs

B. X will be a better choice for all inputs except possibly small inputs

C. X will be a better choice for all inputs except possibly large inputs

D. Y will be a better choice for small inputs

Limit Rule

given arbitrary functions f and $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$,

1. if $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}^+$ then $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$,
2. if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) \in O(g(n))$ but $g(n) \notin O(f(n))$, and
3. if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$ then $f(n) \notin O(g(n))$ but $g(n) \in O(f(n))$.

We illustrate the use of this rule before proving it. Consider the two functions $f(n) = \log n$ and $g(n) = \sqrt{n}$. We wish to determine the relative order of these functions. Since both $f(n)$ and $g(n)$ tend to infinity as n tends to infinity, we use de l'Hôpital's rule to compute

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{1/n}{1/(2\sqrt{n})}$$
$$= \lim_{n\to\infty} 2/\sqrt{n} = 0.$$

Now the limit rule immediately shows that $\log n \in O(\sqrt{n})$ whereas $\sqrt{n} \notin O(\log n)$.

Comparison of Time Complexities

$$O(c) < O(\log\log n) < O(\log n) < O(n^{1/2}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(n^k) < O(2^n) < O(n^n) < O(2^{2^n})$$

$$f_1(n) = n^2 \log n \qquad f_2(n) = n(\log n)^{10}$$

Comparison of Time Complexities

$$O(c) < O(\log \log n) < O(\log n) < O(n^{1/2}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(n^k) < O(2^n) < O(n^n) < O(2^{2^n})$$

$$f_1(n) = n^2 \log n \qquad f_2(n) = n(\log n)^{10}$$

f1>f2

$n \cdot n \cancel{\log n}$	$n \cancel{\log n} (\log n)^9$
n	$(\log n)^9$
$\log n$	$\log (\log n)^9$
$\log n$	$\cancel{9} \cdot \log \log(n)$

GATE BASED QUESTION

Let $f(n)=n^2\log n$ and $g(n)=n(\log n)^{10}$ be two positive functions of n which of the following statement is correct?

a) $f(n)=O(g(n))$ and $g(n)!=(f(n)$
b) $g(n)=O(f(n)$ and $f(n)!=O(g(n))$
c) $f(n)=O(g(n))$ and $g(n)!=(f(n)$
d) $f(n)=O(g(n))$ and $g(n)=O(f(n)$

SOLUTION

b)

Arrange in increasing order

$f_1(n) = 2^n, \quad f_2(n) = n^{3/2}, \quad f_3(n) = n \log_2 n, \quad f_4(n) = n^{\log_2 n}$

Solution

f3f2f4f1