TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

Topics Covered

1. Recursion and its analysis
2. Asymptotic notations

Complexity Analysis

- Complexity measure: time complexity
  - Recursion
    - A same set of instructions are executed repeatedly
    - If it repeats ‘*n*’ times and a single set of instruction requires ‘*m*’ unit of time, time complexity will be *(m*n)* unit
    - Recursion is an alternate implementation technique of loop structure
      - Coding or writing algorithm is simple (involves less number of instructions)
      - Difficult to understand the running mechanism
      - It uses system STACK to maintain intermediate data to be processed
      - Processing overhead is higher for STACK operations

Complexity Analysis

- Complexity measure: time complexity
  - Recursion (example)

Problem: Find out the factorial of a given number ‘*n*’

Algorithm:

```
factorial (n)
{
 if n <= 1
    return 1
 else
    return (n * factorial(n-1))
}
```

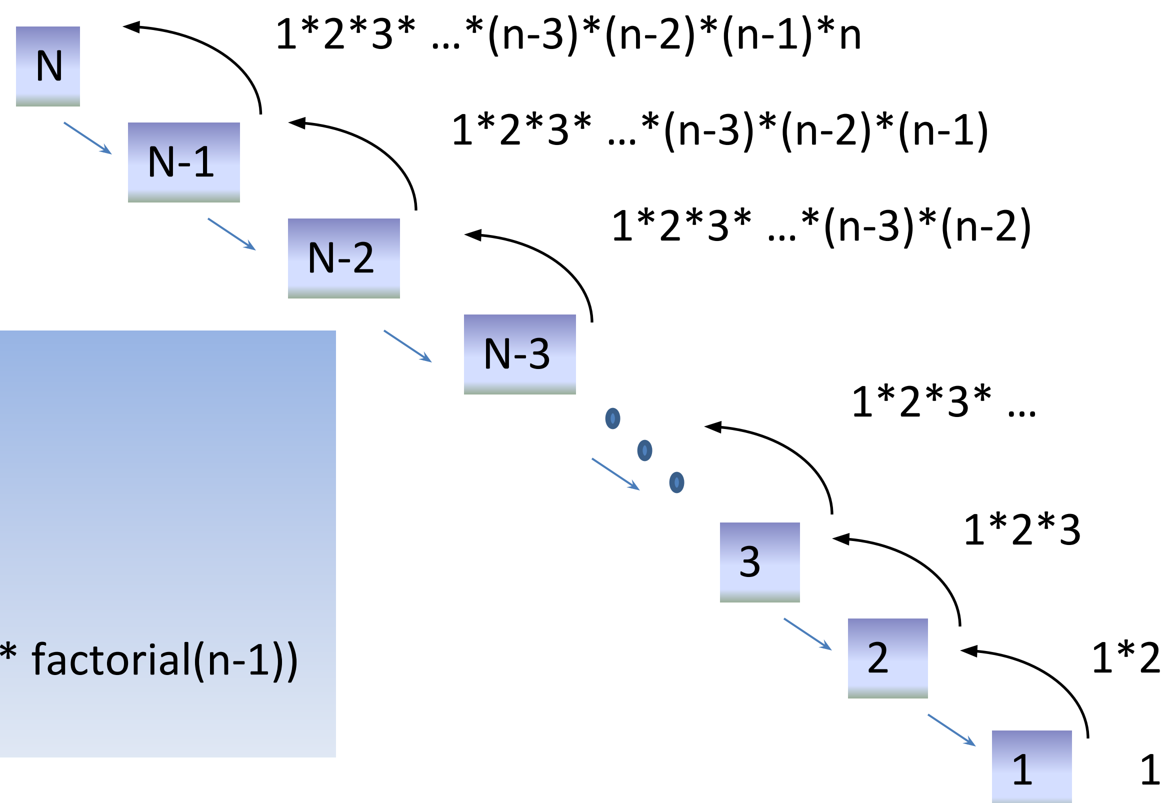
Complexity Analysis

- Complexity measure: time complexity
  - Recursion (visualization)

2 , Factorial(1)
3 , Factorial(2)
…………………………….
N-3 , Factorial(N-4)
N-2 , Factorial(N-3)
N-1 , Factorial(N-2)
N , Factorial(N-1)

STACK

```
factorial (n)
{
  if n <= 1
      return 1
  else
      return (n * factorial(n-1))
}
```



Complexity Analysis

- Complexity measure: time complexity
  - Recursion (analysis)

Problem: Find out the factorial of a given number '*n*'

```
factorial (n)
{
 if n <= 1
    return 1
 else
    return (n * factorial(n-1))
}
```

Time complexity::

$$T(n) = \begin{cases} c & \text{if } n<=1 \\ T(n-1) + d & \text{if } n>1 \end{cases}$$

(c and d are constants)

T(n) = T(n-1) + d

Now, T(n-1)= T(n-1-1)+d (substitute *n* by *n-1)*
T(n) = T(n-2) +2d

Again, T(n) = T(n-3) + 3d *(substituting n by n-2)*

In general term, T(n) = T(n-i-1) + (i+1)*d *(n by n-i)*

T(n) = T(1) + (n-1)*d (after *n-2* iteration when i=n-2)
= c + (n-1)d [as T(1)=c]

Complexity Analysis

- Asymptotic Notation
  - Rate of growth
    - How the complexity (running time) increases with input size
  - Asymptotic meaning
    - If two function graphs (curve) never meets for any input data, in mathematics, termed as asymptotic curves
    - Algorithm complexity is represented by employing such curves, called as asymptotic notations
  - Need of asymptotic notation
    - The complexity analysis of an algorithm may produce a large polynomial equations (contains several terms)
    - We can write such terms in a single-term representation

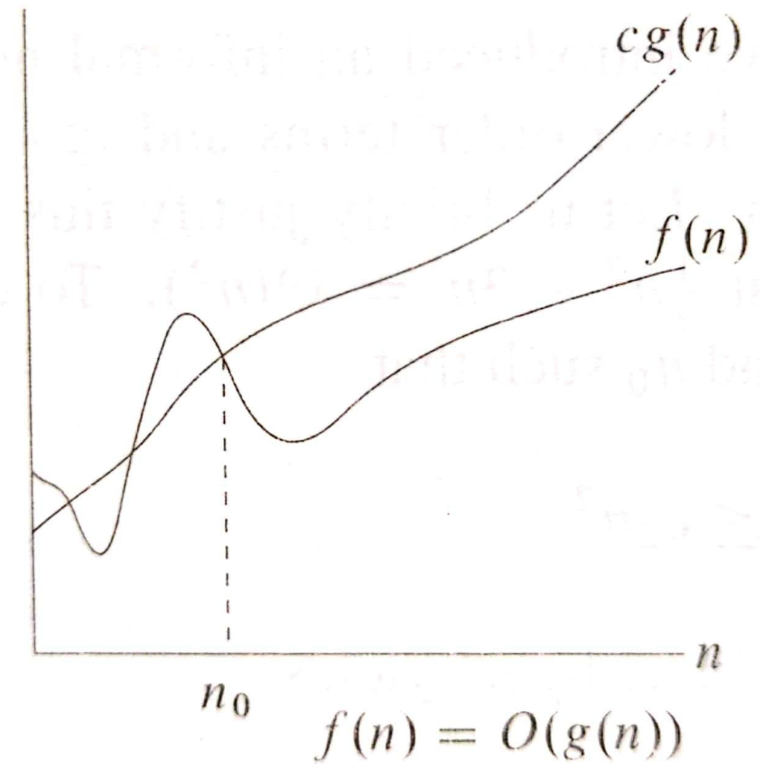
Complexity Analysis

- Asymptotic Notation
  - Big-Oh (O)
    - Defines the upper bound of an algorithm complexity
    - Talks about the **worst case** complexity of an algorithm
  - Omega (Ω)
    - Defines the lower bound of an algorithm complexity
    - Talks about the **best case** complexity of an algorithm
  - Theta (ϴ)
    - Defines both the upper and lower bound of algorithm
    - Talks about the **average case** complexity of an algorithm

Complexity Analysis

- Asymptotic Notation
  - Big-Oh (O): definition

$O(g(n))$ = { $f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le c*g(n)$ for all $n \ge n_0$ }



Scanned with CamScanner

Complexity Analysis

- Asymptotic Notation
  - Big-Oh (O): example

$T(n) = 3n^2 + 5n + 10$
$\leq 4n^2$
$= O(n^2)$

$f(n) = 3n^2 + 5n + 10$

$g(n) = n^2 \;,\; c = 4$

$T(n) = (n^2/3) + 5n + 10$
$\leq n^2$
$= O(n^2)$

$f(n) = (n^2/3) + 5n + 10$

$g(n) = n^2 \;,\; c = 1$

$T(n) = n\log(n) - n + 5$
$\leq n\log(n)$
$= O(n\log(n))$

$f(n) = n\log(n) - n + 5$

$g(n) = n\log(n) \;,\; c = 1$

Largest term of the polynomial is considered as the Big-oh time complexity

Complexity Analysis

- Asymptotic Notation
  - Big-Oh (O): example (finding n_0)

$T(n) = 3n^2 + 5n + 10$
$\leq 4n^2$
$= O(n^2)$

$f(n) = 3n^2 + 5n + 10$

$g(n) = n^2$, $c = 4$, $n_0 = 7$

$3n^2 + 5n + 10 \leq 4n^2$ or $\mathbf{5n+10 \leq n^2}$

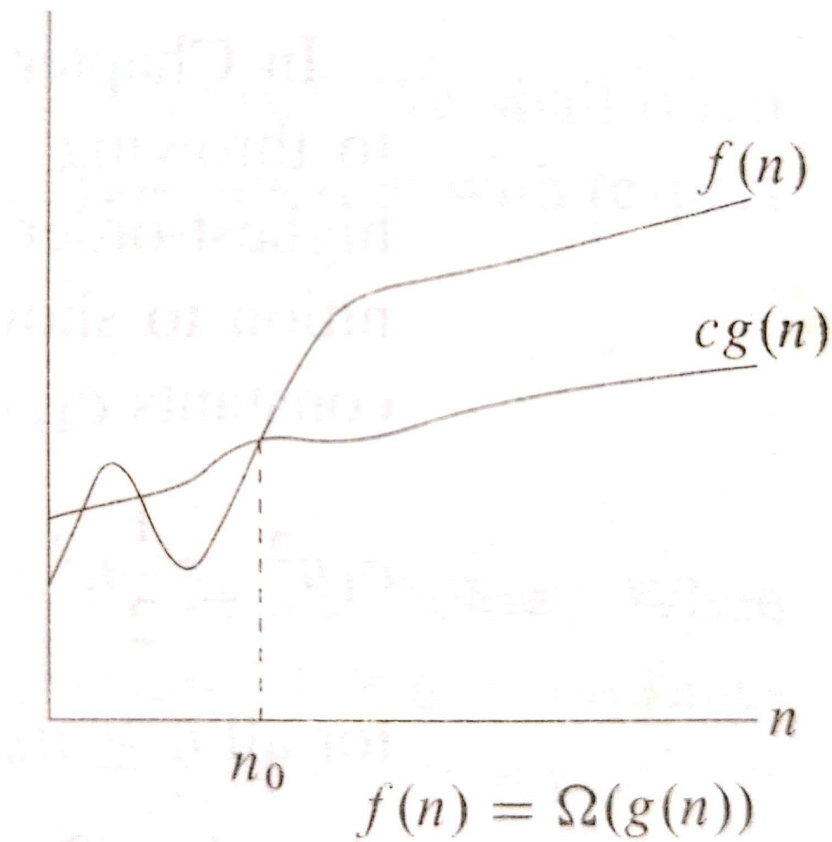
for n=1 , $5*1+10 > 1^2$
for n=2 , $5*2+10 > 2^2$
for n=3 , $5*3+10 > 3^2$
for n=4 , $5*4+10 > 4^2$
for n=5 , $5*5+10 > 5^2$

for n=6 , $5*6+10 > 6^2$
for n=7 , $5*7+10 < 7^2$
for n=8 , $5*8+10 < 8^2$
for n=9 , $5*9+10 < 9^2$
for n=10 , $5*10+10 < 10^2$

Complexity Analysis

- Asymptotic Notation
  - Omega (Ω): definition

$\Omega(g(n))$ = { $f(n)$: there exist positive constants c and n_0 such that $0 \le c{*}g(n) \le f(n)$ for all $n \ge n_0$ }



CS Scanned with CamScanner

Complexity Analysis

- Asymptotic Notation
  - Omega (Ω): example

$T(n) = 3n^2 + 5n + 10$
$\geq 3n^2$
$= \Omega(n^2)$

$f(n) = 3n^2 + 5n + 10$

$g(n) = n^2$, $c = 3$

$T(n) = (n^2 /3) + 5n + 10$
$\geq n^2 /4$
$= \Omega(n^2)$

$f(n) = (n^2 /3)+ 5n + 10$

$g(n) = n^2$, $c = 1/4$

$T(n) = n \log(n) - n$
$\geq \log(n)$
$= \Omega(\log(n))$

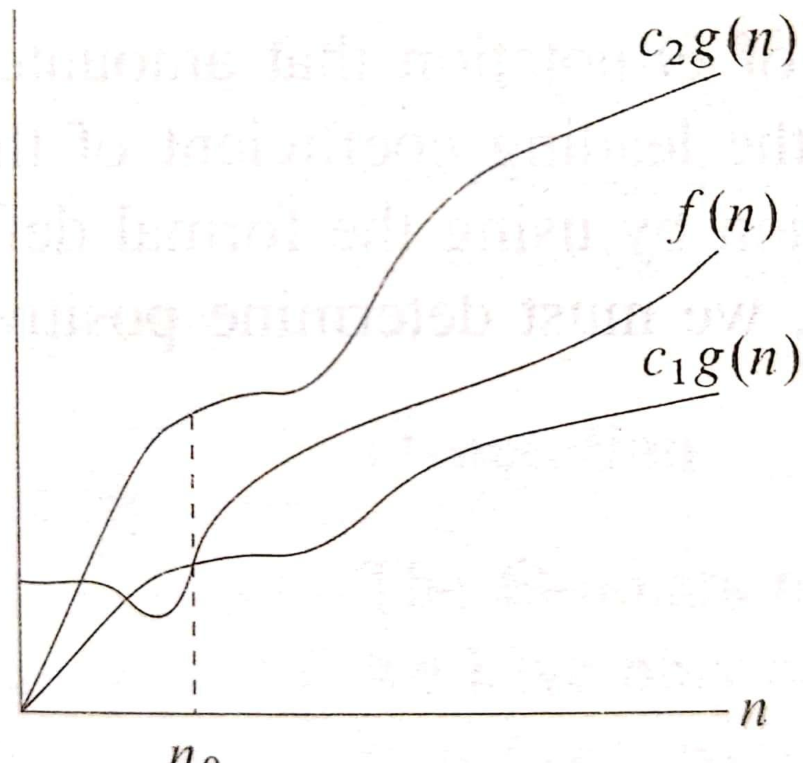
$f(n) = n \log(n) - n$

$g(n) = \log(n)$, $c = 1$

Complexity Analysis

- Asymptotic Notation
  - Theta (Θ): definition

$\Theta(g(n))$ = { $f(n)$: there exist positive constants c_1, c_2 and n_0 such that $0 \le c_1 * g(n) \le f(n) \le c_2 * g(n)$ for all $n \ge n_0$ }



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

Scanned with CamScanner

Complexity Analysis

- Asymptotic Notation
  - Theta (ϴ): example

$$T(n) = 3n^2 + 5n + 10 = \Theta(n^2) \quad \text{as} \quad \begin{cases} T(n) = O(n^2) \\ T(n) = \Omega(n^2) \end{cases}$$

$$T(n) = 100 = \Theta(1) \quad \text{as} \quad \begin{cases} T(n) = O(1) \\ T(n) = \Omega(1) \end{cases}$$

$$T(n) = n \log(n) - n = \Theta(n \log(n)) \quad \text{as} \quad \begin{cases} T(n) = O(n \log(n)) \\ T(n) = \Omega(\log(n)) \end{cases}$$

Theta can be estimated for random input size if both upper bound and lower is defined

Complexity Analysis

- Rate of Growths

$n! > 2^n > n^2 > n \log(n) > \log(n!) > n > 2^{\log(n)} > \log^2(n) > \sqrt{\log(n)} > \log(\log n) > 1$

Queries?