TY CSE
2022-23
Design and Analysis of Algorithm

Lect. 1 13/08/2022

1. Course Information
2. Teaching Scheme
3. Examination Scheme
4. Course Objectives
5. Course Outcomes (CO) with Bloom's Taxonomy Level
6. Module Contents
7. Text Books
8. References
9. Useful Links
10. **Module 1: Introduction**
    a. Introduction to Algorithm Analysis Time and Space Complexity, Elementary operations and Computation of Time Complexity-Best, worst and Average Case Complexities- Complexity Calculation of simple algorithms. Recurrence Equations: Solution of Recurrence Equations –Iteration Method and Recursion Tree Methods. Master's theorem for complexity computation.
11. **Definition**:

Finite set of Steps to solve a problem.

Steps means Fundamental Instructions (i.e. Instructions which contains the operatn +, −, *, ÷, %, =, etc. are called fundamental Instructions.

12.Characteristics of fundamentals operators:

1. Definiteness : Every fundamental operator should be defined without any ambiguity.

e.g. $i = i + 1$ → Invalid

$i = i + 1$ → Valid

2. Finiteness : Every fundamental instructions should be terminated within finite amount of time.

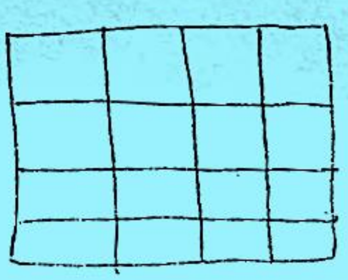
```
i = 1
while (1)
{
    i = i+1 ;      // This type of instn are not allowed .
}
```

3. Every fundamental expressn accepts at least 0 i/p. & provides at most one o/p.

13. Steps to solve problem:

1. Identifying problem Statement :

ex: Arrange 4 queens Q_1, Q_2, Q_3, Q_4 into 4x4 Chess board



② Constraints : No two pins on same row and No two pins in same column & diagonal.

~~Des~~

③ Design Logic : Depending upon the characteristics of problem we can choose any one of the following design strategy for designing logic.

a) Divide & conquer.
b) Greedy method
c) Dynamic programming
d) Backtracking
e) Branch & Bound
f) Bruteforce

etc.

④ Validation : Most of the algorithms validated by using mathematical induction.

⑤ Analysis : The process of comparing two algorithms with respect to time, space, rate of growth, network Bandwidth, number of registers, etc. is called Analysis.

Priory Analysis	Posterior Analysis
① Analysis done before executing. ex. $x = x+1$ Principle: freq. count of fundamental instructions since $x = x+1$ being carried out only one time so its time complexity is $O(1)$.	① Analysis done executing. ex: $x = x+1$ System1 0.1 ns, Sys. 2 0.2 ns, Syst. 3 0.4 ns
② It is independent of O.S., syst. Architecture, Processor speed.	② Dependent
③ It provides estimated values.	③ It provides exact values.
④ All these are uniform values	④ These are non-uniform values.

⑥ Implementation.

⑦ Testing and Debugging

14.Asymptotic notation:

<u>Asymptotic Notation</u>

To compare two algorithm's growth rate, we need notations called Asymptotic Notations.

1) Big-Oh (O) :-

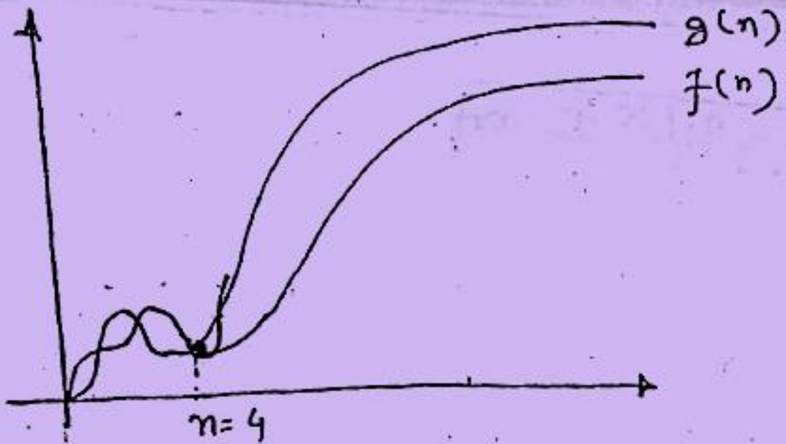
Def: $f(n)$ and $g(n)$ are two functions

$f(n)$ is $O(g(n))$ iff $\exists$ some 'c' and 'k'

$\ni f(n) \leq c \cdot g(n)$; $\forall\ n \geq k$.

ex:

n	$f(n) = n^2$	$g(n) = 2^n$	
1	1	2	$\rightarrow f(n) < g(n)$
2	4	4	$\rightarrow f(n) = g(n)$
3	9	8	$\rightarrow f(n) > g(n)$
4	16	16	$f(n) \leq g(n)$.
5	25	32	
6	36	64	
7	49	128	
8	64	256	
9	81		



(3)

$f(n) = O(g(n)) \Leftrightarrow f(n) \leq g(n).\ ;\ \forall\ n \geq 4$

$n^2 = O(2^n) \quad \forall\ n \geq 4.$

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

<u>Note</u>:— $f(n) = O(g(n))$ means $g(n)$ is an upper bound to $f(n)$ for <u>large values of n</u>.

* Lower growth rate function = O(higher growth rate function).

pb1 If $f(n) = n^2 + n + 1$ then $f(n) = O(\,)$.

Sol^n:

To prove $n^2 \leq$

$$f(n) = O(g(n)) \iff f(n) \leq c \cdot g(n); \ \forall n \geq k$$

$$n^2 \leq n^2$$

$$n^2 + n \leq n^2 + n^2$$

$$n^2 + n + 1 \leq n^2 + n^2 + n^2$$

$$n^2 + n + 1 \leq 3n^2 . \quad \forall n \geq 1$$

c. k

$$\therefore n^2 + n + 1 = O(n^2) \qquad n \geq 1$$

Shortcut: If $f(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3 + \dots + a_m n^m$ $(a_m \neq 0)$.

then $f(n) = O(n^m)$.

$n^2 \leq n^3$.

$n^2 + n + 1 \leq n^3$.

$n^2 + n + 1 = O(n^3)$.

14 lack

10.5 lac

$= fn(n) = 10$ lack

<u>Note</u>: Even though n^2, n^3, n^4 are upper bounds to $f(n) = n^2 + n + 1$ take ~~up~~ least upper bounds only.

$\therefore f(n) = O(n^2)$.

(Problem 2): If $f(n) = n!$ then $f(n) = O(\;)$

<u>Sol.</u>: $f(n) = n!$

$= n * (n-1) * (n-2) * n - 2)$

$= n^n \{ 1 \cdot (1 - \frac{1}{n}) (1 - \frac{2}{n}) \cdots \frac{1}{n} \}$

$= O(n^n)$ ($\because$ polynomial of degree n)

③ If $f(n) = \log(n!)$ then $f(n) = O(\ )$.

sol^n: $f(n) = \log(n!)$

$= \log(O(n^n))$ $\quad (\because n! = O(n^n))$

$= O(\log n^n)$ $\quad$ ($\because$ log is constant fun. it satisfies Associative prop).

$= O(n \log n)$.

④ If $f(n) = n^2$ for $n \leq 100$

$= n$ for $n > 100$

and $g(n) = n$ for $n < 1000$

$g(n) = n^3$ for $n \geq 1000$

which of the following is true?

If $f(n) = n^2 \log n$. $g(n) = n \log^{10} n$.

then which of the following is true?

15.Dominance ranking:

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

$f_4(n) = n^{\log_2 n}$

Arrange f_1, f_2, f_3, f_4 in the increasing order.

(a) $f_3\ f_2\ f_4\ f_1$ (b) $f_2\ f_3\ f_4\ f_1$

(c) $f_3\ f_2\ f_1\ f_4$ (d) $f_2\ f_3\ f_1\ f_4$.

16.

Big-Omega (Ω).

Defn: $f(n)$ is $\Omega(g(n))$ iff $\exists$ some c and 'k' $\ni f(n) \geq c \cdot g(n) : \forall n \geq k$

Problems

Ex 1. If $f(n) = n^2+n+1$ then $f(n) = \Omega(\ \underline{\quad}\ )$.

Soln :-

I. $n^2 \geq n^2$

$n^2 + n \geq n^2$

$\longrightarrow n^2+n+1 \geq n^2 \quad \forall n \geq 0$

$\therefore n^2+n+1 = \Omega(n^2)$.

II. $n^2 \geq n$.

$n^2 + n \geq n$.

$n^2+n+1 \underset{\uparrow}{\geq} \underset{\substack{\uparrow \\ c}}{1} \cdot n \quad \forall n \geq \underset{\substack{\uparrow \\ k}}{0}$

$\therefore n^2+n+1 = \Omega(n)$.

Note: Even though n^2, n are lower boundary to $f(n)$ we have to greatest lower boundary

$\therefore n^2+n+1 = \Omega(n^2)$.

Shortcut: If $f(n) = a_0 + a_1 n + a_2 n^2 + \cdots a_m n^m$ $(a_m \neq 0)$
then $f(n) = \Omega(n^m)$.

$n^2+n+1 = O(n^2)$ & $n^2+n+1 = \Omega(n^2)$.

$\therefore$ $\underline{n^2+n+1 = \Theta(n^2)}$.

Theta (Θ):
$f(n)$ is $\Theta(g(n))$ iff $f(n)$ is $O(g(n))$
and $f(n)$ is $\Omega\, g(n)$.

Ex:- If $f(n) = n^2+n+1$ then
for $c_1 = 3$, $n^2+n+1 \leq 3n^2$ $\forall n \geq 1$] $f(n) = O(n^2)$
for $c_2 = 1$, $n^2+n+1 \leq 1n^2$ $\forall n \geq 0$] $f(n) = \Omega(n^2)$

i.e. $c_2 \cdot n^2 \leq n^2+n+1 \leq c_1 \cdot n^2$ $\forall n \geq 1$.

$\Rightarrow$ $n^2+n+1 = \Theta(n^2)$.

Shortcut: If $f(n) = a_0 + a_1 n + a_2 n^2 + \cdots a_m n^m$ $(a_m \neq 0)$
then $f(n) = \Theta(n^m)$.

Little-Oh : $f(n)$ is $o(g(n))$ iff

$f(n) < c \cdot g(n) \quad \forall\, n \geq K$
$\forall\, c$.

O	o
1. $\leq$	1. $<$
2. $\exists$ some c	2. $\forall\, c$. for all c

ex Let $f(n) = n^2$ $\Rightarrow f(n) < g(n) \Rightarrow f(n) = o(g(n))$
$g(n) = n^3$, $n^2 = o(n^3)$.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n)).$$

6

Ex $$\lim_{n \to \infty} \frac{n^2}{n^3} = \lim_{n \to \infty} \frac{1}{n} = 0$$

$\therefore n^2 = o(n^3)$.

Little omega (ω):

$f(n)$ is a $\omega(g(n))$ iff $f(n) > c \cdot g(n)$. $\forall n \geq K$
$\forall c$.

Ω	ω
1. $\geq$	1. $>$
2. $\exists$ some c	2. $\forall c$:

Shortcut: If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$ then $f(n) = \omega\, g(n)$.

Recap:

1. lower = O (higher)
2. If $f(n) = a_0 + a_1 n + a_2 n^2 + \dots a_m n^m$ $(a_m \neq 0)$
   then $f(n) = O(n^m)$, $\Theta(n^m)$, $\Omega(n^m)$
3. Dominance Ranking
4. $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \Rightarrow f(n) = o(g(n)) \\ \infty & \Rightarrow f(n) = \omega(g(n)). \end{cases}$
5. Properties of Asymptotic

Lect. 2 14/08/2022

i) Reflexivity:

$a = a$

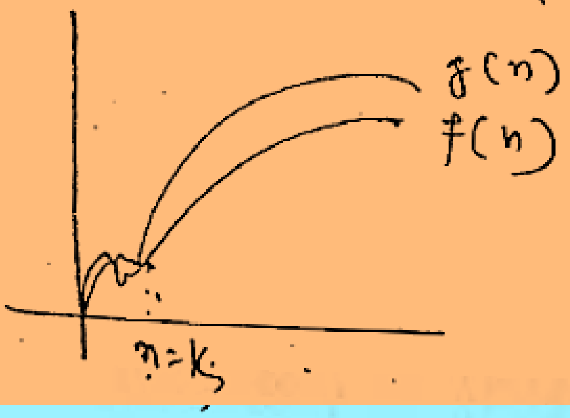
$f(n) \leq 1 \cdot f(n)$. $\forall n \geq k$.

$\Rightarrow f(n) = O(f(n))$

ii) Symmetric:

If $a = b \Rightarrow b = a$.

$\rightarrow f(n) = O(g(n)) \not\Rightarrow g(n) = O f(n)$.



iii) Transitive

If $a = b$, $b = c$, then $a = c$.

If $f(n) = O(g(n))$, $g(n) = O(h(n))$

$\Rightarrow f(n) = O(h(n))$.

iv) Transpose:

$$f(n) = \Theta(g(n))$$
$$\therefore \quad g(n) = \Omega\, f(n).$$

Notation \ prop	Reflexivity	Symmetric	Transitive	Transpose
$O(\leq)$	✓	✗	✓	$O \Rightarrow \Omega$
$\Omega(\geq)$	✓	✗	✓	$\Omega \Rightarrow O$
$\Theta(=)$	✓	✓	✓	$\Theta \Rightarrow \Theta$
$o(<)$	✗	✗	✓	$o \Rightarrow \omega$
$\omega(>)$	✗	✗	✓	$\omega \Rightarrow o$

prob

Consider:

1. $(n+K)^m = \Theta(n^m)$ (where K, m are constants)
2. $2^{n+1} = O(2^n)$
3. $2^{2n+1} = O(2^n)$.

Simple for loop

```
sum = 0
for (i = 1 ; i <= n ; i = i+2)
{
    sum = sum + i ;
}
```

$\rightarrow O\left(\frac{n+1}{2}\right) = O(n+1)$

$= O(n).$

8

```
sum = 0
for ( i = 1 ; i <= n ; i = i * 2)
{
    sum = sum + i;
}
```

$\longrightarrow O(1 + \log_2 n)$

$O(\log_2 n).$

```
j = 1 .
while (j <= n)
{
    j = j * 2 ;
}
```

How many ~~compilations~~ comparisons performed.

```
sum = 0
for (i = n; i > 0; i = i/2)
  {  sum = sum + i;
  }
```

Nested For loop Questions

Q.
```
sum = 0
for (i = 1; i <= n; i = i+1)
  {
     for (j = 1; j <= n; j = j+1)
        {
           sum = sum + j;
        }
  }
```

$\underline{\underline{Sol^n}}$

i	1	2	3	....	n
j	n	n	n	...	n
sum	n	n	n	....	n

$= O(n * n)$

$= O(n^2)$.

It is not the value
number of times executed

<u>means</u>:

for i = 1. n times will be executed

for i = 2 n times ——"——

⋮

i = n ——"——

Q. for (i=1; i ≤ n; i = i+1)

{

for (j = 1; j ≤ n; j = j*2)

{ sum = sum+j;

}

}.

10

i	1	2	3	--	n
j	$\log_2 n$	$\log_2 n$	$\log_2 n$	---	$\log_2 n$ times executes.
sum	$\log_2 n$ +	$\log_2 n$ +	$\log_2 n$ +	.. -	$\log_2 n$.

$= O(n \log_2 n)$

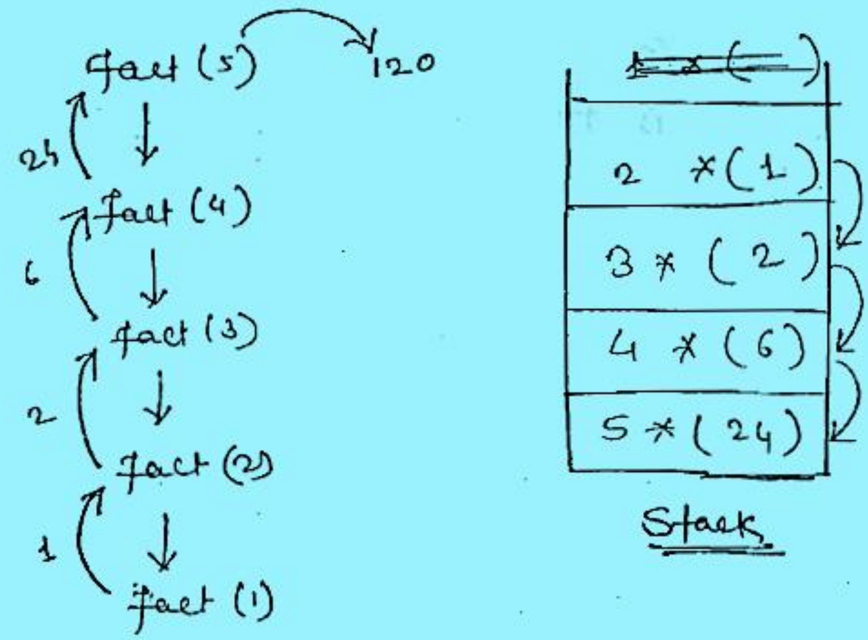
Q.

```
for (i = 1; i ≤ n; i = i+1)
{
for (j = 1; j ≤ i; j = j*2)
{ sum = sum + j;
}
```

Time Complexity of Recursive Algorithm.

```
int fact (int n)
{
    if (n==0 || n==1)  // Base condition
        return 1.
    else
        return n x fact(n-1) ; // induct^n cond^n
}
```

⇒ Execution



Note: 1) Time complexity of recursive algorithm
= Number of function calls

$\therefore$ Time complexity of fact = $O(n)$.

2) Space Complexity of Recursive Algorithm
= Depth of Recursion tree

$\therefore$ Space Complexity of fact (n) = $O(n-1)$
= $O(n)$.

17.Back-Substitution Method