$|f(x)| \leq c|g(x)| \leq c'|g(x)|$.

Date 14-10-24.

Chapter no. 3.

SECTION - 3.2 :-

(i) Studying and Analyzing the growth of a function.
(ii) To determine that if an algorithm is practical enough to be used as sol. to our problem.
(iii) Compare 2 algorithms to determine that which one is more efficient.

BIG-O NOTATION :- (special Notation used to describe the growth of a function.)
↳ Upper bound ↳ worst case.

"let f and g be functions from the set of integers or the set of real numbers to the set of real no.s. We we say that $f(x)$ is $O(g(x))$ if there are constant C & k such that;

$|f(x)| \leq c|g(x)|$ — (a) whenever $x > k$.

↳ implies :- $f(x)$ is big-oh of $g(x)$.

→ C & k are called as witnesses.
→ whe need only one pair of witnesses to prove (a) as have/having only one pair of witnesses means to infinitely many pairs of witnesses.

let 'C' and 'k' be one pair of witnesses
C' and k' be any pair wher $C < C'$ & $k < k'$
then C' & k' are also witnesses.
because of equ (b) at top.

Goal:- (i) first select the value of k.

$f(x) = x^2+2x+1 \leq C.x^2$ for which size of $|f(x)|$ can be readily estimated to $x>k$.

for large x (=find k) (ii) Then find value of 'C' to prove — (a).

Date

1

Example:-

Show $f(x) = x^2+2x+1$ is $O(x^2)$

solution :-

→ $|x^2+2x+1| = x^2+2x+1$ when $x \geq 0$.

Think.

where x is really big, let 1 million.

$\therefore x^2+2x+1 \approx x^2+2x$

where x is really really big ; let $(1mil)^2$

$\therefore x^2+2x+1 \approx x^2$

hence $2x+1$ become insignificant as $x \to \infty$ however they still contribute to functions growth so we need to find an upper bound for the whole function. — (i)a

Now upper bounding $2x+1$ term with x^2 each/respectively.

$x^2+2x+1 \leq x^2+2x^2+x^2$ when $x \geq 1$.

$\leq 4x^2$ when $x \geq 1$

Now this in std. definition of Big-oh.

$|x^2+2x+1| \leq 4|x^2|$ when $x \geq 1$.

$\therefore C = 4$ & $k = 1$.

NOTE:-

"When $f(x)$ is $O(g(x))$ and $h(x)$ is a function that has larger values than $g(x)$ does for sufficiently large (↑absolute) values of x, it follows that $f(x)$ is $O(h(x))$

⇒ $O(g(x))$ can be replaced by a function with larger absolute values.

Date

$|f(x)| \leq C|g(x)|$ if $x > k$

and if $|h(x)| > |g(x)|$ $\forall$ $x > k$ then,

$|f(x)| \leq C|h(x)|$ if $x > k$.

Example 2:

Show that $7x^2$ is $O(x^3)$.

→ let $x > 7$

→ multiply bothsides by x^2 $x^3 > 7x^2$.

→ here $\left.\begin{matrix} C = 1 \\ k = 7 \end{matrix}\right\}$ witnesses.

Example 3:-

Show that n^2 is not $O(n)$.

(Goal, we must show that no pairs exist to satisfy this) at $n^2 \leq Cn$.

Prove by contradiction; (prove that no C & k exist to satisfy).

$n^2 \leq Cn$. when $n > k$ ($C, k \Rightarrow$ consts)

→ divide bothsides by n $n \leq C$.

✓ now for any value of C & k we can not prove this inequality

Date

EXAMPLE NO: 4.

X^3 is $O(7x^2)$?

Prove by contradiction (Proof that no C & k exist).

If C and k are witnesses the inequality.

$x^3 \leq C(7x^2)$ holds for all $x > k$.

dividing x^2 on b.s

$x \leq C(7)$ —— (a)

Now for any value of C, it is not the case that $x \leq 7C$ for all $x > k$ no matter what k is. and disobey the condition (a).

∴ x can be made arbitrarily large; it follows that no witness C and k exist for this Big-O relationship.

3.2.3 BIG-O NOTATION FOR SOME IMP FUNCTS

Polynomials can be used to analyze the growth of funct. Instead of analyzing the growth of polynomials each time they occur, we would like a result that can be used instead.

Do we have to learn theorems? Is their proof included?

Date

Theorem 1:-

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

where $a_0, a_1 \dots a_{n-1}, a_n$ are $\in \mathbb{R}$

Then; $f(x) \leq O(x^n)$ / $f(x)$ is $O(x^n)$.

EXAMPLE 5:-

How can Big-O Notation be used to estimate the sum of the first n positive integers?

Because the sum of first n-positive integers doesnot exceed n.

$$1+2+\dots+n \leq n+n+\dots n = n \times n$$
$$\Rightarrow n^2.$$

from this inequality.

$1+2+3\dots+n$ is $O(n^2)$. → let $C=1$, $k=1$

In this function example
Domains of function in Big-O are set to positive integer.

EXAMPLE 6:-

Give Big-O Notations /estimates for factorial functions & log of the factorial function, where factorial function $f(n) = n!$ defined by ....

→ P.T.O

Date

$n! = 1.2.3 \ldots n.$

whereever n is positive integer.

$0! = 1$

$1! = 1.$

$2! = 2 \times 1.$

$3! = 1 \times 2 \times 3.$

Solution

A big-O estimate for $n!$ can be obtained by noting each term in the product does not exceed n.

$$n! = 1 \cdot 2 \cdot 3 \ldots n.$$
$$\leq n \cdot n \cdot n \ldots n$$
$$= n^n.$$

$n!$ is $O(n^n)$ (By $C = 1$ & $k = 1$).

taking log on b.s.

$\log n!$ is $O(n \log n)$ (again $C = 1$ & $k = 1$)

EXAMPLE 7:-

Show that inequality $n < 2^n$ implies. that n is $O(2^n)$ and use this inequality to show $\log n$ is $O(n)$.

Date

Using the inequality $n < 2^n$ we conclude that n is $O(2^n)$ by taking $K = 1 = C$.

$n < 2^n$ —— (a)

taking log (Base -2) on b.s.

$\log n < n$. —— (b)

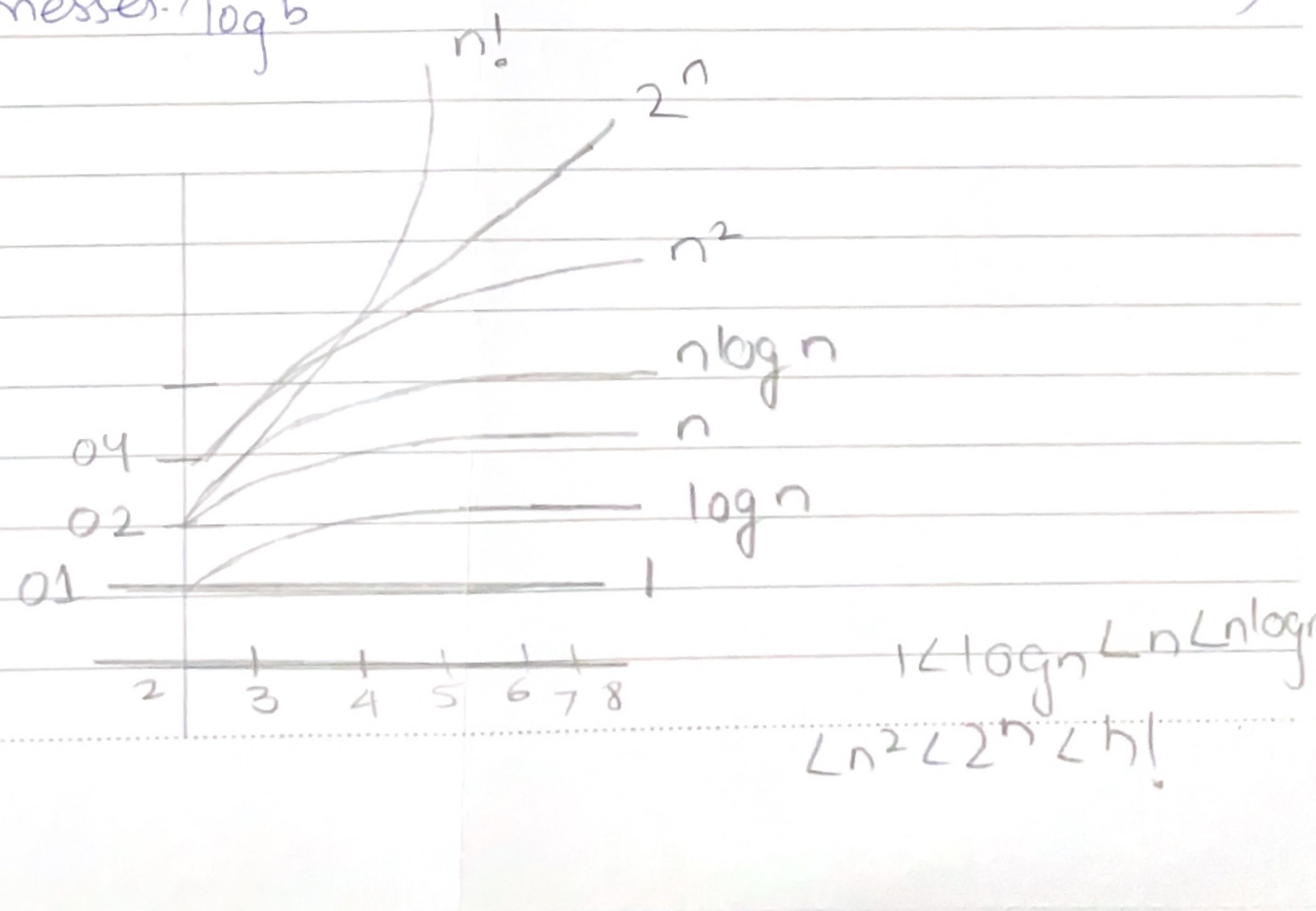
If follows that $\log(n) < O(n)$. $\left(\begin{matrix} C = 1 \\ k = 1 \end{matrix}\right)$

Now, if we have log to a base b where b is different from 2, we still have $\log_b n$ is $O(n)$ because

$$\log_b n = \frac{\log n}{\log b}(x\,1) < \frac{n}{\log(b)} \quad \text{—— (c)}$$

n is a positive integer take

(whenever λ $C = 1/\log b$ & $k = 1$ as witnesses. ; we take $C = 1/\log b$.)



Date — do we have to prove the statement a. or b)

NOTE:-

let $f(n)$ be a polynomial of degree d or less then $f(n)$ $O(n^d)$

Applying Theorem 1.

(b) ← if $d > c > 1$ n^c is $O(n^d)$ — (a)

not proved in book — (a) but n^d is not $O(n^c)$ — (b)

Also by example 7.

$\log_b n$ is $O(n)$ when $b > 1$.

then

$(\log_b n)^c$ is $O(n^d)$ — (c)

but n^d is not $O(\log_b n)^c$ — (d) → this tells us that positive power of the logrithm of n to the base b where $b > 1$ is Big-O of every positive power of n but the reverse never holds.

By example 7 and so on ....

n^d is $O(b^n)$
b^n is $O(c^n)$
c^n is $O(n!)$
} vice versa not possible

Date

THEOREM: 2

let.

$f_1(x)$ is $O(g_1(x))$

$f_2(x)$ is $O(g_2(x)$

Then $(f_1 + f_2)$ is $O(gx)$ where $g(x) = (\max(|g_1(x)|, |g_2(x)|)$ for all x.

THEOREM : 3 :-

$f_1(x)$ is $O(g_1(x))$

$f_2(x)$ is $O(g_2(x))$

is

Then $(f_1 f_2)$ is $O(g_1(x)\, g_2(x))$.

THEOREM : 4 :-

Let $f(x) = a_n x^n + (a_{n-1})x^{n-1} + \dots a_1 x + a_0$ where $a_0, a_1 \dots a_n$ are real no.

BIG OMEGA NOTATION :- (lower bound, best case)

let f and g be functions from the set of integers or the set of Real no.s to the set of real no.s.

We can say $f(x)$ is $\Omega(g(x))$ if there are constants C and k with C positive such that.

$$|f(x)| \geq C\,|g(x)| \quad \text{whenever} \quad x > k.$$

Big theta is thus the average case formed by combing Ω & O

$$C_1|g(x)| \leq |f(x)| \leq C\,|g(x)| \Rightarrow \text{Big } \Theta$$

Date

SECTION 3.1:-

Comparing two algorithms:-

$f(n)$ → input size / Data size.

No. of comparisons.

(Max function):-

```
procedure max (a1, a2 ... an : integers).
max := ai
for i = 2 i<n to n.                    (2)a   ← loop runs. (n-1) times.
    if max < ai then max := ai
return max { max is the biggest/largest element }
```

(1)a ← a: two comparisons in loop.

$\therefore f(n) = 2(n-1) + 1$ ← Terminating condition.

LINEAR SEARCH

```
procedure linearSearch (x : integer, ai - an)
                        integers
```

Date

$2\log n + 2$.

i := 1

while (i ≤ n ξ x ≠ ai)

i = i + 1.

if i ≤ n then location := i ;

else location := 0

return { location is subscript of the term that equals to x }.

$f(n) = 2n + 2$.

2 log n + 1 + 1 → terminating condition

2 log n → loop

2 log n + 2 → if

BINARY SEARCH

i = 1. ✓

j = n. ✓

$\frac{(n-1)(n)}{2}$.

{ while i < j ✓

m (i+j) / 2.

if (x > (m) then i = m + 1. ✓

else j := m }

if x := ai then location := i.

else location := 0 ✓

return location.

Date Bubble Sort

procedure

Similarly for insertion sort;

```
for i:= 1 to n-1.
    for j= 1 to n-i
        if aj > aj+1 then interchange aj + aj+1
{a1 a2 ... an in increasing order}
```

i = 1 ; j = 1 to n-1 => n-1.
i = 2 ; j = 1 to n-2 => n-2.
i = 3 ; j = 1 to n-3 = n-3.

(n-1) + (n-2) ... + 1.

$\frac{n(n+1)}{2}$ ∈ n → O(n²).

— x —

Matrix Chain Multiplication:-

Consider the simplest case.

$$[a_1\ a_2\ a_3\ a_4]_{1\times 4} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}_{4\times 1} \quad \text{— e.g ①}$$

=>

$$[a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4]_{1\times 1}$$

↓

4 products & 3 sums/additions.

Date

now.

$$\underset{A}{[n^2]_{n\times n}} \quad \underset{B}{[n^2]_{n\times n}} \ldots \quad \underset{C}{[\quad]_{n\times n}}$$

∴ we have predicted in matrix multiplication that

for one value;

# of multiplication = n. } frame ①.
# of addition. = n−1; }

for all values.

# of multiplications/products = $n^2 \times n$.
# of additions = $n^2(n-1)$.

Now.

Example

A_1 30×20 $\quad$ A_2 20×40 $\quad$ A_3 40×10.

let associative law.

$$(A_1 \times A_2) \times A_3 \text{ —— ④}$$

$$B \times A_3 \Rightarrow C$$

30×40 $\quad$ 30×10.

# of products for C ⇒ $30\times40\times10 = 12000$ ↑ $B\times A_3$
$+$
$30\times20\times40 = 24000$ ↑ $A_1\times A_2$
$+$
36000

Date

Now

$A_1(A_2 \times A_3)$

↓

$\Rightarrow A_{1 \times B}$
20×10.

$\Rightarrow C$
30×10.

# of products for C;

① $20 \times 40 \times 10 = 12000$

② $30 \times 20 \times 10 = + \underline{6000}$

₹4000

∴ $A_1(A_2 A_3)$ is more efficient

— X —

(1) Big-O estimate for the no. of operations ....

```
t:=0                    — 1
for i:=1 to 3           — 2
  for j:= 1 to 4        — 3
    t:=t+ij             — 4
```

→ Only line no. 4 has operation of sum or product.
→ It contains one product b/w i & j which is then added to t
→ ∴ 2 operations per value of i & j
→ i takes values from 1–3 ; j from 1–4.

Total no. of operation = product of the no. of values for i and j and the no. of operations in line 4.

∴ $4 \times 3 \times 2 = 12$

$12 \times 2 = 24$ ∴ O(1)