Page No.:
Date:

Notes on Data Structures. & Algorithms.

① Introduction

ⓐ → DS is a way of collecting and organizing data so as to use it effectively later.

ⓑ → Basic Terminology.

→ Data → set of values. (Eg:- Roll no:s of students, marks)

→ Singular of data → data item.

→ ~~Entity → data with similar attributes~~
~~Eg:- (All employees of a single org.)~~.

→ ~~Field → Single unit of information representing an attribute of an entity.~~

Collec of field → Record
Collection of Record → File.

② Types →

ⓐ Int, float, bool, char, etc also store data, ∴ they are also DS called Primitive data structures.

ⓑ complex data structures.

Linear DS. ← → Arrays, Linked lists
→ Stacks, Queues

Non Linear DS ← → Trees and graphs also file.

③ Basic operations Inside a Data Structure:

a) → Traversation → Going through the list, accessing each record.

b) → Searching → Finding a particular value from all record.

c) → Inserting → Adding a record (At beg, end or at a pos).

d) → Deleting → Deleting a record (").

e) → Sorting → Arranging data in some logical order.

f) → Merging → Combining 2 records into one single one.

④ Abstract Data Types

→ Set of data values & operations specified accurately by coder.

→ We know what a ADT does, but not how it does it.

* Note → For a stack → Position on top | Status of Stack.
-1 | Empty
0 | Only one elem.
N-1 | Stack is full
N | Overflow.

Orders of Stacks (&) Queues

Stacks	Queue.
① Push → O(1)	Enqueue → O(1)
② Pop → O(1)	Dequeue → O(1)
③ Top op → O(1)	Size → O(1)
④ Search → O(n)	

Page No.
Date:

Design And Analysis of Algorithms.

Class – 1.

(1.) Parts of Algorithms → 1) Their correctness
2) Efficency (measured using Asym compl)
3) Modelling (Graphs, decomsing the problem, other data structures).
4) Techniques (D&C, Greedy & DP).

(2) First we will look at some examples.

Class 2

(1.) Air Travel → Same as Graph mentioned in Maths – 1 in Week 10 – 12

(2) Xerox Shop → Have to xerox a set of papers in a pre-determind time then how to optimize and finish

Method 1 → Brute force i.e try every possible order & choose the best possible/optimal sol.

Method 2 → Decompose [using techniques & recursion].

Method 3 → choice of next job, to be optimal [Greedy].
Eg:- do the least page once next.

* variations exist → If photocopies is fast (or) slow and time for reloading papers.

Page No.:
Date:

Class 3 → Introduction

① Design & Analysis of Algorithms.

Design ↓ We have an problem. & we have to design an algo to solve it

Analysis ↓ Trying to see how efficent an algo is using time & space required.

② If we design an algo and run it on 2 diff system both will give diff time of exec.

This is bcz of hardware, compiler, etc differences. So to measure an algo we don't use a partic system rather a method.

③ Input size affects running time. $f(n)$ is worst case. Hardware has it's limits, improving time in algorithms is way better.

④ When we look at func of n, we ignore the constants
∴ if n^3 vs $5000n^2$

$5000n^2$ might look big, but n^3 will overtake it in time
5000 is ignored.

and what happens ~~in~~ the limit, as n increases is called asymptotic notations.

Typical fns → $\log n$ $\quad n\log n$ $\quad n$ $\quad n^2$ $\quad n^3$ $\quad 2^n$ $\quad n!$ [Feasibility discussed]

fast ⟶ slow

Class 4 → Input size, worst & Avg Case

(1.) How we measure input large or not.
→ We take a typical parameter (eg: for sorting : array size is input size and for graphs : No: of vertices + No: of edges)

(2.) In an example set (53, 503, 5003, 50003), time is not growing prop. i.e to power of ten i.e we don't take magnitute, we take no: of digits as input size.

(3.) Worst case → For each n, worst case input forces algo to take the max amount of time.
(You can guess worst case by looking at algo).

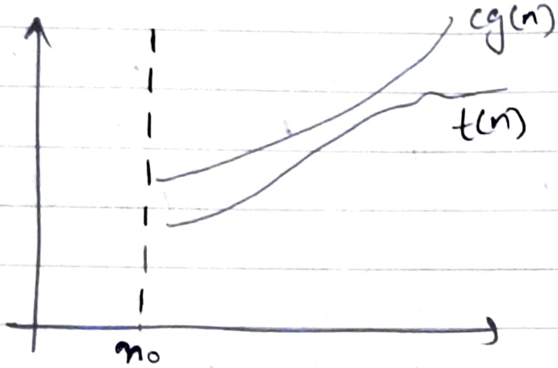
(4) Avg Case → Compute time taken over all input. Though this looks more sensible to take than worst case, it is very hard to find ~~worst~~ avg case for every algo.

∴ we generally use worst case as a upper bound.

Class 5 → Quantifying efficiency.

① Now that we have seen basic elements, let's see how we can compare fns wrt to order of magnitude.

② Upper bound, (big 'O')



$t(n) \leq cg(n)$
for $n \geq n_0$
then $cg(n)$ is upper bound for $t(n)$.

Eg:- Upper bound for $100n + 5$.

$100n + 5 \leq 100n + 5n$ for $n \geq 1$*
$\Rightarrow 105n$

WKT, $105n \leq 105n^2$ [In this step we ignore consts]

$\Rightarrow \therefore$ for $n_0 = 1$*, $c = 105$

[remember n_0 & c values can change]

Eg 2 :-

$100n^2 + 20n + 5$

$\Rightarrow 100n^2 + 20n^2 + 5n^2$

$\Rightarrow 125n^2 \geq 100n^2 + 20n + 5$

[Here $n_0 = 1$, $c = 125$]

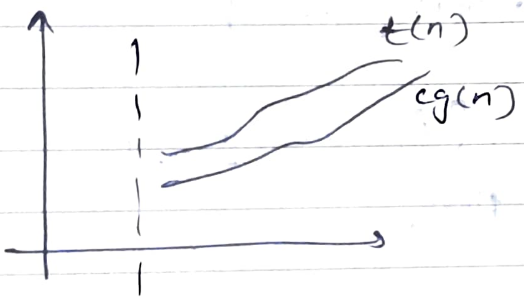
∴ we fig out a way to find upper bound.
i.e we ignore smaller terms & func.

③ Suppose you have an algo with has diff phases then,

Phase (A) takes $O(g_A(n))$ } then upper bound is max b/n $O(g_A(n))$ & $O(g_B(n))$.
Phase (B) takes $O(g_B(n))$

∴ Even if there are n phases, max upper bound is Upper bound is ~~max~~ upper bound for the whole algo.

④ <u>Lower bound, Ω (omega)</u>



$t_n \geq cg(n)$

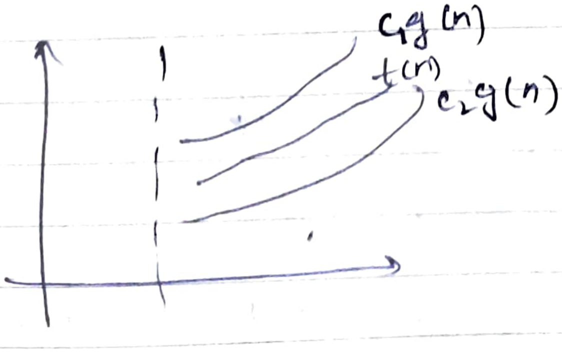
for $n \geq n_0$.

cg(n) is lower bound.

⇒ We use Ω for saying 'this problem takes atleast this much time'.

Eg:- [sorting has Ω as '$\Omega(n \log n)$']

⑤ <u>Tight bound, $\Theta(n^2)$</u>



if t_n becomes both upper & lower bound.

$c_2g(n) \leq t(n) \leq c_1g(n)$

Class 6 → Examples

① Calculating complexity → Iterative program
→ Recursive program

② func max-ele(A):

```
maxval = A[0]          → O(1) [ignored]
for i=1 to n-1:        → runs n-1 times.
    if A[i] > maxval   → comparision } c = constant
        maxval = A[i]  → assigning.  }
return (maxval)        → O(1) [ignored]
```

∴ $c \cdot (n-1)$ [ignoring consts $-1, c$]

$\rightarrow O(n)$ Ans.

③ 2 loops → $O(n^2)$. [even if 2 loop is optimized it comes as $O(n^2)$].

④ Matrix Mul.

```
function MatrixMultiply (A, B):
    for i=0 to n-1:        ] n2
        for j=0 to n-1:    ]
            C[i][j] = 0
            for k=0 to n-1:  ] n
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
    return (c)
```

$O(n^3)$.

Page No
Date:

(5) function noOfBits(n):

```
count = 1
while n>1:
    count = count + 1
    n = n div 2
return(count)
```

$\frac{n}{1} \to \frac{n}{2} \to \frac{n}{4} \dots\dots 1$

$n \times \left(\frac{1}{2}\right)^x = 1$

$\Rightarrow n = 2^x \times 1$

$\Rightarrow O(\log_2 n)$ is comp.

(6) Recursive Algo

$\Rightarrow$ Towers of Hanoi problem.

$\Rightarrow$ takes $2^n - 1$ time $\Rightarrow O(2^n)$.

x