Chapter 1 :- Introduction

called information

- Data in systamatic way becomes structured & meaningfull
- Data structure information provides that organisation or structure to your data
- Proper choice of D.S. provides efficieny to your data / code

1.1 :- Data Types :-

provided by language

- Defines domain of allowed values & operations allowed
- for example in float, '%' operation / bitwise operations are not allowed.
- custom data types can be created f.ex. no data types for "date" in C. we can create, but may need to provide range & allowed operations manually.
- also called "premitive data type or built in data type"

another name

imp.

independent view

1.2 :- Abstract Data types :-

- specifies set of data or collection of operations that can be performed on that data
- specifies operations to be performed but doesnt show how this operations will be implemented.

defination of abstraction

- provides only essential & hides all other details

List ADT :-

- elements of same type arranged sequencially
- allowed operations are
  - initialise() – initialise list to be empty
  - get() – return element at any given position
  - insert() – insert – " –
  - remove() – remove 1st occurance of any element
  - removeAt() – remove from perticular position
  - replace – replace at any position by another element
  - size() – returns no of elements from list
  - isEmpty() – returns if list is empty else return false
  - isFull() – " true if full, false if not full

Stack ADT :-

- elements of same type arranged in sequencial manner
- allowed operations are:

like print value only!

- initialise()
- Push(): insert an element at one end called top
- Pop(): remove from top of stack
- Peek(): return without removing from top
- size(): returns no of elements
- isEmpty(): -"-
- isFull(): -"-

Queue ADT

- elements of some type arranged sequencial order
- allowed operations are
  - Initialise():
  - Enqueue(): insert at end
  - Dequeue(): remove first element
  - Peek(): return 1st element without removing it
  - size(): return no of elements of queue
  - isEmpty(): -"-
  - isFull(): -"-

Dont know how we imp. it in lower level (Abstract)

* Those ADT's could be implemented using anything like array's, linked list or doubly link list.

1.3: Data structure :-

- to implement ADT we need Data structures
- operations on ADT ≅ functions in coding implementation
- Data structure consist of

two essential part of D.S.

(1) Bunch of variables for storing data in ADT
(2) Algorithm for implementing more operations

what to do?

- ADT → abstract representation of data + operations

How to do?

- D.S. → actual -"-
- Data structure can be nested

Advantages of Data structure:

- Efficiency - Reusability - abstraction

1.3.1:- Liniar + Non liniar Data structures.

liniar D.S. is when data is sequencially ordered. Every one has successor + predecessor. e.g. array, linked list, stack, queue.

Non linear DS is when data is not ordered. E.g. tree & graph

1.3.2: Static & Dynamic Data structures:

Static: memory allocated at compile time & fixed

Dynamic: memory allocated at run time & can change

1.4. :- Algorithms :-

Data stored in D.S. is manipulated using diff. algorithms

common approaches of algorithm design:

1.4.1: Greedy Algorithm :-

- take decision that appears best at that moment. once decision is taken, it never thinks about the same decision
- local optimum is choosen at every step in the hope to get global optimum at the end.
- not always guanteed to have optimum solution.

1.4.2 :- Divide And Conquer Algorithm

- Divide problem into smaller problem
- solve smaller problem and merge solutions to get final answer
- e.g. quick sort, merge sort, binary search.

1.4.3: Backtracking (trial & error process)

- several options, where one might lead to solution.
- take one option, try it fails try another called trial & error

1.4.4: Randomised Algorithms

- Random numbers are used to make decisions
- e.g. merge sort where random no. is choosen as pivot.

1.5 :- Big O Notation :-

Most common notation to measure performance of algorithm by defining its order of growth

* IMP

$f(n) \leq c\,g$ for all $n \geq n_0$ for constant c and n_0

This implies f(n) doesnt grow faster than g(n) or g(n) is upper bounded on function f(n)

time complexity = linear time — O(n) (big oh of n)

constant time — O(1) (big oh of 1)

quadratic time — $O(n^2)$ (big oh of n^2)

Rules of Big O Notation :-

(1) Transitivity: if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then $f(n)$ is $O(h(n))$

(2) If $f_1(n)$ is $O(h(n))$ and $f_2(n)$ is $O(h(n))$ then $f_1(n) + f_2(n) = O(h(n))$

(3) If $f_1(n)$ is $O(h(n))$ and $f_2(n)$ is $O(g(n))$ then $f_1(n) + f_2(n)$ is $\max(O(h(n)), O(g(n)))$

(4) If $f_1(n)$ is $O(h(n))$ and $f_2(n)$ is $O(g(n))$ then $f_1(n) \times f_2(n)$ is $O(h(n).g(n))$

(5) If $f(n) = c \Rightarrow O(1)$

(6) If $f(n) = c \cdot g(n)$ where c is constant then

(7) $f(n) = O(h(n))$

Any polynomial $P(n)$ of degree m is $O(n^m)$

$O(1) \Rightarrow$ constant	$O(n^2) \Rightarrow$ quadratic
$O(n) \Rightarrow$ liniar	$O(n^3) \Rightarrow$ cubic
$O(\log n) \Rightarrow$ logarithmic	$O(2^n) \Rightarrow$ exponential
$O(n \log n) \Rightarrow$ liniar logarithmic	$O(n!) \Rightarrow$ multiple permutation of set of data