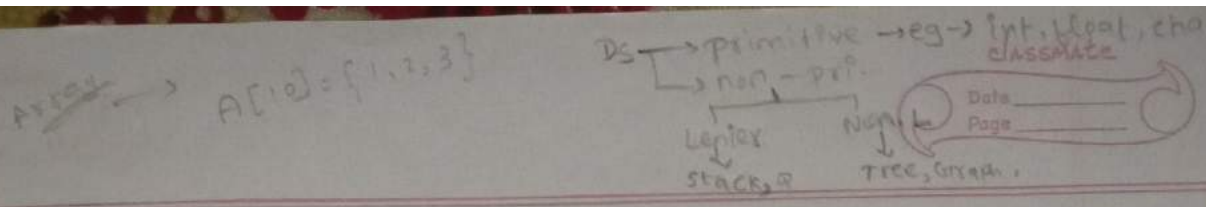


01 : ELEMENTARY DATA ORGANISATION.

* Data :- Datas are simply / set of values.
* Data item :- Single unit of data is data item.
* Grp item :- Data items are divided into sub grps.
* Entity :- It is something that has certain attributes which may be assigned values.

→ Entity set :-
An Entity with similar attribute → Entity set.

* Information :- It is meaningful / processed data.
* Field :- Single elementary unit of Information representing an attribute of an Entity.
* Records :- They are collection of field values of an given entity
* File :- They are collection of record of the ~~Entries~~ entities of a given set.

→ Definition of Data structure :-

* It is the specialised format of orgauising, retreiving and storing data.
* eg → Arrays, Stack, queue, Tree, Graph.

* Algorithms are used to manipulate the data containing in these D.S.

→ D.S operations :-

1) Traversal → Defined as the process of visiting every elemnt of a D.S atleast once.

* This operation is most commonly used for searching, printing, Displaying / ~~sorting~~ reading the elemnts stored in D.S.
* It is usually done by using a variable or pointer that indicates the crnt elemnt of D.S ~~of~~ being processed.

2) Insertion :-
* once a D.S is created it can be extented by adding new elemnt, this process → Ins.

3) Deletion :-
~~a) searching~~ ~~b) Merging~~ once a data str is created a user can remove any of the existing elemnts and free up the space occupied by it. This process → Dlt.
~~c) Datatype Vs DS~~ Deletion operation always ends up in reducing the size of D.S.

4) Search :-
The process of locating an elemnt in DS and returning its index / address → searching.

5) Sorting :-
The process of arranging the data elemnts in a DS in a specific order. [ascending / Descending] by specific key values → sorting.

6) Merging :- process of combaining elemnts of 2 D.S.

→ Data type Vs D.S :-

Datatype	Data Structure.
* It is the kind / form of a variable which is being used throughout the prgm.	It is the collection of different kinds of data.
* Implimentation through datatypes is a form of abstract implimentation	Implimentation through D.S is → Concrete implimentation.
* It can hold values Ep not data, so it is data less.	It can hold ~~diff~~ different type of data.
* values can directly be assigned ~~through~~ to datatype variables.	The ~~type~~ data is assigned to D.S using some set of algorithms.
* No Prbm of time complexity.	Time complexity comes into play when wrking with D.S

→ Types of D.S :-

~~primarily are~~
D.S are primarily divided into 2 classes → primitive and non-primitive D.S

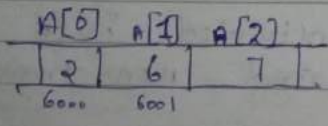
* primitive D·S include all the fundamental D·S that can be directly manipulated by machine level instructions.
eg → int, char, real, boolean

* Non-primitive D·S refer to all those primitive D·S that are derived from 1 / more D·S.
- Non-primitive D·S are further classified into 2 types —

(a) Linear	(b) Non-linear
eg → Stack, Queue, Array, Linked list	eg → Tree, graph

* In linear D·S all the data elements are arranged in a linear / sequencial manner.
* In non-linear D·S there is no definite order / sequence in which data elements are arranged.
Non-linear D·S could arrange data elements in a hierarchical manner.

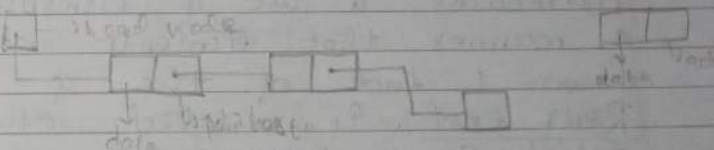
⇒ Array :-
* An array is a collection of similar type data elements stored at a consicutive location in the memory.
eg → list of integrs, grp of names, grp of array elements is refferred with a common name



→ logical representation of array.

→ Application of array :-
* Arrays are particularly used in pgms that require storing large collection of similar type data elemnts.

⇒ Linked list :-
* It is a data str used for D·S ~~for the form of~~ used for storing data in form of list it comprises of multiple nodes connected to each other through a pointer.
* Each node comprises 2 part, 1 part contains the data value while the other part contain a pointer to the nxt node in the list.
* linked list eliminates 1 of the main disadvtages associated with arrays (i.e) inefficient utilisation of the memory space



→ Application of L.list :-
* used in the situation where there is a need of dynamic memory allocations.

⇒ Stack :- It is a linear data str; [LIFO] that maintains a list of elemnts in such a manner that elements can be

Ins/Dlt → Top

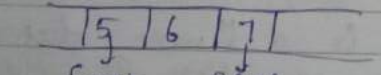
inserted / deleted only from end of the list.
* This end is referred ~~to~~ as Top of the stack.
* Stack is based on Last In First Out (LIFO) principle, which means the element (i.e) last added to the stack is the one that is 1st removed from the stack.

→ Application of Stack :-

* Typically used in the implimentation of system process, such as compilation & program control.

⇒ Queue :-

(FIFO) * It is linear str. that maintains a list of elements in such a manner that elements are inserted from 1 end of the Q called Rear end. & deleted other end called front end.
* Q is based FIFO [First In First Out] principle. → the element that is 1st added to the Q is also the 1 that 1st removed from the Q.

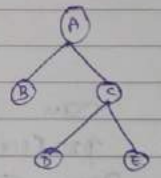


→ Application of Q :-

* Q's are typically used in the implication of system process such as CPU Scheduling, Resource sharing, etc.

⇒ Tree :-

It is a linked data str that arrange its nodes in the form of hierarchical tree str.
* Each node comprises of 0 / more child node.
* The node present at the top of the tree is referred as root node

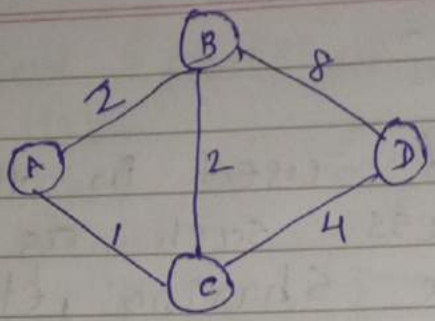


→ Application of Tree :-

* Tree data str is typically used in for storing hierarchical data implementing search tree & maintaining sorted data.

⇒ Graphs :-

It is a linked data str that comprises of a grp of vertices → nodes & grp of edges.

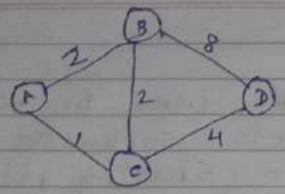


→ Application of Graph :-

* 1 of the typical application of graph is in the modelling of networking systems.
It helps to compute the cost of transmitting data from a particular network path.

⇒ Algorithms :-

An algorithm can be defined as a stp by stp procedure that provides soln to a gvn prbm.
It comprises of a well defined set of finite no. of stps/ rules that are executed sequentially to obtain the desired soln.



→ Application of Graph :-

* 1 of the typical application of graph is in the modelling of networking systems.
It helps to compute the cost of transmitting data from a particular network path.

⇒ Algorithms :-

An algorithm can be defined as a stp by stp procedure that provides soln to a gvn prbm.
It comprises of a well defined set of finite no. of stps/ rules that are executed sequencially to obtain the desired soln.

* Insertion Sort :-

1) Start
2) Set i=1.
3) Repeat stps 3-9 while i<=limit-1
4) Set val = A[i]
val = A[1] = 11
5) Set j = j-1
6) Repeat stps 6-7 while j>=0 and A[j] > val.
7) Set A[j+1] = A[j]
8) j = j-1
9) Set A[j+1] = val
10) Set i = i+1
11) Stop.

j = 1-0 = 0 A[j] = A[0] = 12

① i=1 j=0 j=-1

upto 7 →

0	1	2	3	4
12	11	13	5	6

1st iteration → | 12 | 12 | 13 | 5 | 6 |

val=12 ② J=-1

by ⑩ → j = j-1 ⇒ j = 0-1 = -1

2nd iteration ⇒ | 11 | 12 | 13 | 5 | 6 |

J=1 val=13 ③

⑥ false

A[j+1] = val

5	11	12	13	13
5	11	12	12	13
5	6	11	12	13

1st iteration i=1 val=11 J=0
A[0] = 12 12 > 11 true.
inner loop.
~~J = J + 1~~ will be replaced with 12.
J = J-1 become -1 break.

2nd iteration i=2 val=13 J=1 a[J] = a[1] = 12
12 > 13 false. break.

3rd iteration i=i+1 i=3 val=5 J=2
a[2] = 13
1st inner loop. A[2+1] = A[2]
S = 13
2nd inner loop J = J-1 become J=1
A[1+1] = A[1]
A[1] = A[1]
3rd inner loop J = J-1 J=0.
A[0+1] = A[0]

* Selection Sort :-

1) Set i=0
2) Repeat stps 3-9 while i<n-1
3) Set min = i
4) Set j = i+1
5) Repeat stps 6-7 while j<n
6) if A[j] < A[min] then min=j
7) Set j = j+1
8) Swap A[i] and A[min]
9) Set i=i+1
10) Stop

⇒ Efficiency of algorithms :-

* The term efficiency in the context of algorithms it points at 2 aspects
  * whether the algorithm runs faster
  * whether the (al) takes a amount of memory space.

* Time complexity is the measure of the running time of an (al) for a gvn set of inputs.

* Space complexity is the measure of the amount of memory space required by an (al) for its complete execution, for a gvn set of inputs.

→ Big O notation :-

* It is the method used to express the upper bound of running time of (al).
* Denoted by O.
* Using this notation one can compute max possible amount of time that an (al) will take for its computation.

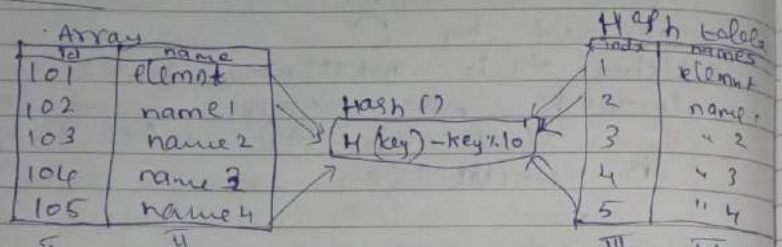
~~→ Types of arrays :-~~

* Bubble Sort :-

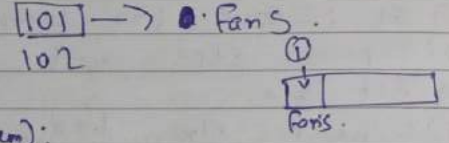
1) Start
2) Set i = size, j=0, temp=0
3) Repeat stps 4-9. while i>1
4) Set j=0
5) Repeat stps 6-8. while j<i-1
6) if A[j] > A[j+1] goto stp 7 else stp 8.
7) Swap the elemnts by performing follow.. stps
   1) temp = a[j+1].
   2) a[j+1] = a[j]
   3) a[j] = temp
8) Set j = j+1
9) Set i = i-1
10) Stop.

02 : ARRAYS.

* Hashing



I & III → H(key)




```
def hash(item):
    ite
    return item % 10

def hashtable():
    c = hash(101)        # 1
    hash(101)            # 1
```

→ Quick Sort :- lb = 0 up = 5.

1) Start
2) Set pivot = 0, nxt_pvt = 0, left = lb, right = ub.
3) Set pivot = arr[left].
4) Repeat steps 5-14 while lb < ub

0	1	2	3	4	5
34	99	5	2	57	40

5) Repeat step 6 while a[ub] >= pivot and lb < ub.
6) Set ub = ub - 1
7) If lb is not = ub goto 8 else to
8) Set a[lb] = a[ub]
9) Set lb = lb + 1.
10) Repeat steps 11 while a[lb] <= pivot & lb < ub.
11) Set lb = lb + 1.
12) If lb ≠ ub goto step 13 else goto step 15
13) Set a[ub] = a[lb]
14) Set ub = ub - 1
15) Set a[lb] = pivot
16)

→ Limitations of array :-

* Static memory allocations
* Ins & dlt difficult.

=> Types of arrays :-

1) 1D array → It is a grp of same data elements like integrs, floats / char
representation → a[].
eg → int a[] = {2, 3, 5} → a grp of int

2) Multi D array → It is a grp of data elements where each elemnts is itself an array.

Representation → a[][][].

eg → int a[][][] = {"AK programz", "AK Editz"}

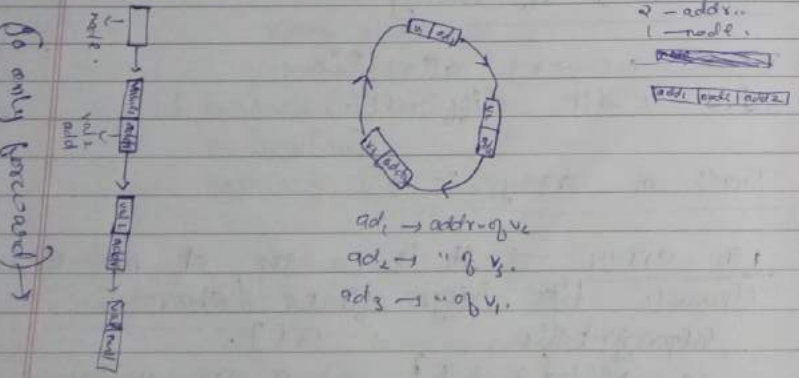
→ Array Traversal :-

traverse (a[], n).

1 – Start
2 – Set i=0
3 – Repeat stps 4-5 , while i<n
4 – Access a[i]
5 – Set i=i+1
6 – stop.

=> Linked list :-

Singly L.S | Circular L.S | Doubly L.S



→ L.S operations =

1) ins → biginning, middle, ending.
2) dlt
3) traversal.

Pass by value ① return ② pass by address ③.

int a=10
int main(a)
int main(&a)

ins

```
Struct NODE {
        int INFO;
        Struct node *NEXT;
};
```

1) Create NEW NODE
2) set data(NEW NODE) = ITEM
3) if START = null then
   Set START = NEWNODE
   set next(START) = NULL
   else
   Set next(NEW NODE) = START
   set START = NEWNODE.

=> Algorithm for array ins at a specific point

① insert (A[N], K, P).

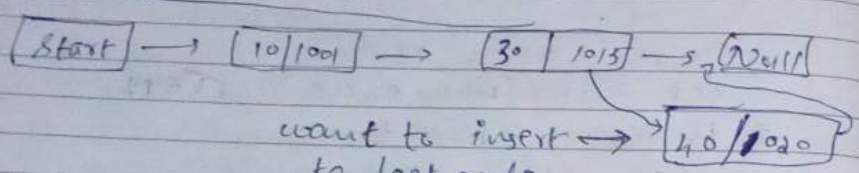
1) Start
2) Set i=n
3) Repeat stps 4-5 while i>=12
4) Set a[i+1] = a[i]
5) Set i = i-1
6) Set a[K] = p.
7) Set n = n+1
8) stop.

→ ins at beginning :- (array)

② insert (A[limit], N).

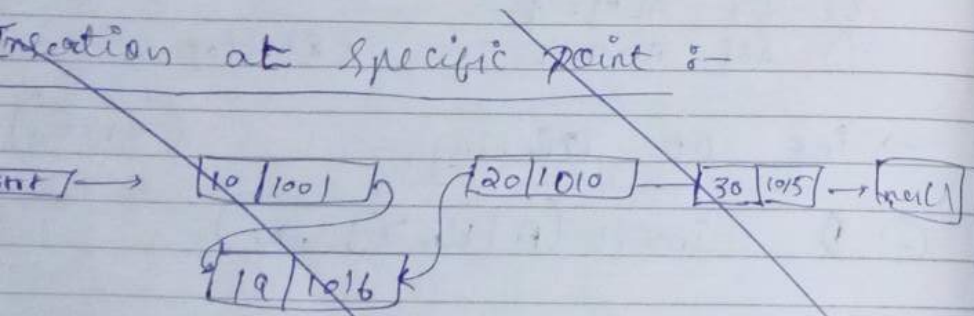
1) Start
2) if limit = max, display ins not possible.
3) j = limit - 1

③ Insertion at the end :- (1.5)



Insert_end(Start, item)
1) Start
2) Create new node
3) Set Data(newnode) = item
4) Set next(newnode) = null
5) if Start = null then → same statements: Start=null.
Start = new node.
goto Stp 9 else goto 6.
6) Repeat 7 while next(current) ≠ null.
7) Set current = next(current)
8) next(current) = new node.
9) Stop.

Insertion at specific point :-



beginning continue → array & ins & type & at end → alg. } exam

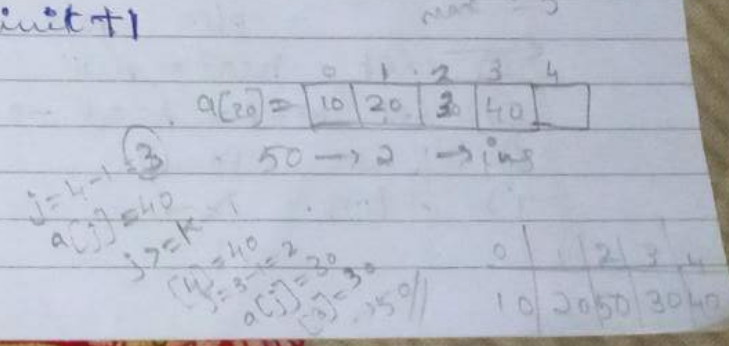
4) Read element in n
5) Repeat Stp 6 & 7 while j >= 0.
6) a[j+1] = a[j]
7) j = j - 1
8) a[0] = n
9) limit = limit + 1.
10) Stop.

* Ins at end :- [ ]

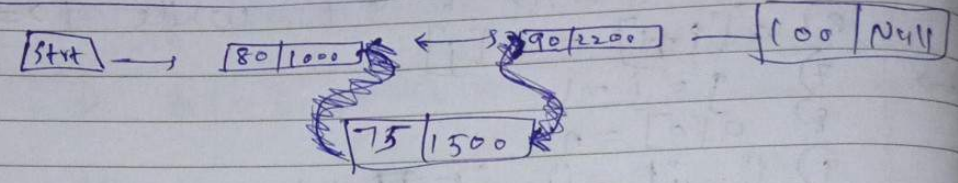
1) Start
2) if limit = max, ins not possible.
3) Read new n.
4) a[limit] = n
5) limit = limit + 1
6) Stop

* ins at specific at point :- [ ]

1) Start.
2) if limit = max, ins is not possible.
3) j = limit - 1
4) Repeat Stps 5 & 6. while j >= k.
5) a[j+1] = a[j]
6) j = j - 1
7) a[k] = n.
8) limit = limit + 1
9) Stop.



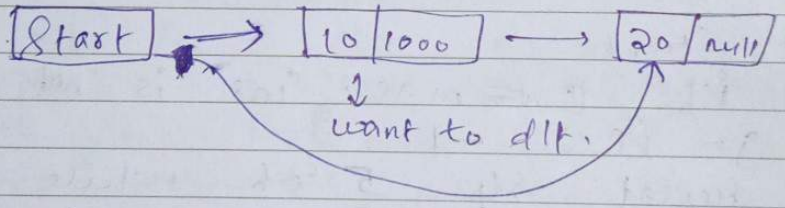
* ins at specific point → (L.5)



insert(pos, item)

1) Start
2) create new node.
3) Set temp = head
4) Repeat stp 5-6 while i < pos
5) Set temp = temp → next
6) Set i = i+1
7) Set new node → next = temp → next.
8) Set temp → next = new node.
9) Stop.

→ Deletion in L.S :- (beginning)



1) Start
2) Set temp = head → common in dlt.
3) if head → next = null, goto stp 4.
   else goto stp 5.
4) Set head = null & goto stp 6
5) Set head = head → next.
6) free temp
7) Stop.