Data Structure Operation:

The data which is stored in our data structure are processed by some set of operations.

(i) Insertion:- Add a new data in data structure.

(ii) Deletion:- Remove a data from the data structure

(iii) Sorting:- Arrange data in increasing or decreasing order.

(iv) Searching:- Find the location of data in data structure.

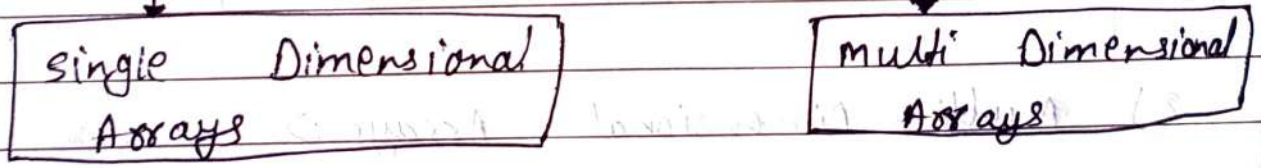
(v) Merging:- Combining the data of two different sorted files into a single sorted file.

(vi) Traversing:- Accessing each data exactly one in the data structure so that each data item is traversed or visited.

Arrays

- An Array can be defined as an infinite collection of homogenous (similar type) element.
- Array are always stored in consecutive (specific memory Location).
- Array can be stored multiple values which can be refrenced by a single name.

Types of Arrays:



(1) Single Dimensional Arrays:

- It is also known as One Dimensional 1D Array.
- It's use only one subscript to define the elements of Arrays.

[row] [coloumn]

Declaration :-

Data_type var_name [Expression];
↓
size

Ex: int num [10];
char C [5];

Initializing one - Dimensional Array :-

Data_type var_name [Expression] = {values};

Ex ⇒

int num [10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
char a [5] = {'A', 'B', 'C', 'D', 'E'}

2) Multi - Dimensional Arrays ⇒

Multi Dimensional Arrays use more than one subscript to describe the Arrays elements.

Two Dimensional Arrays :-

- It's use two subscript, one subscript to represent row value and second subscript to represent coloumn value.
- It mainly use for matrix value Representation.

Declaration

Data_type var_name [rows], [coloumns]

Ex. int num [3],[2];

Initialization of 2D arrays:

Data_type var_name [rows] [coloumn] = {values};

Ex ⇒

int num [3] [2] = {1, 2, 3, 4, 5, 6};

or

int num [ ] [ ] = {1, 2, 3, 4, 5, 6};

Write a program to read and write one Dimensional Arrays:

```
# include <stdio.h>
# include <conio.h>

  void main ( )
{
    int a [10], i;
    clrscr ( );

Printf ("\n Enter the Array Element");
```

P.T.O.

```
for ( i=0 , i <=9 , i++)
{
   scanf ("%d", &a[i] );
}

Printf ("The Entered Array is");

for (i=0; i<=9 , i++)
{
   Printf ("%d\n", a[i]);
}
   getch ();
}
```

Write a program to read & write 2D arrays

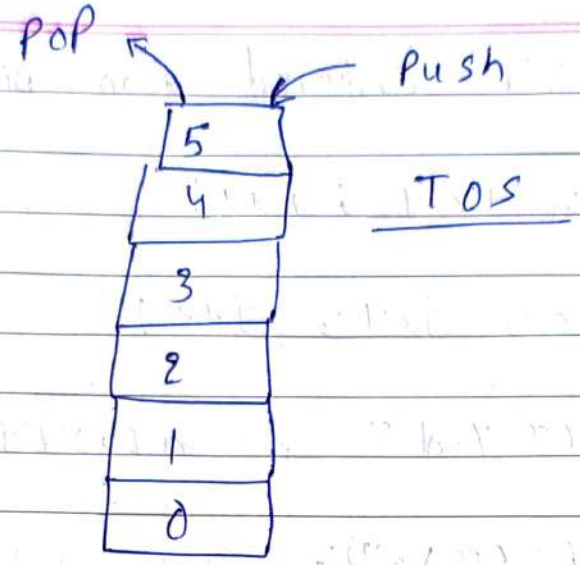
```
# include <stdio.h>
# include <conio.h>

  void main ()
{
   int a[2] [2], i, j;
   clrscr();

Printf ("Enter the elements of Array");
  for (i=0, i<=1 , i++)
  {
     for (j=0 , j<=1 , j++)
     {
        scanf ("%d ", &a[i] [j];
     } }
```

```
Printf ("The Entered two Di Array is");
for (i=0 ; i<=1 ; i++)
{
  for (j=0 ; j<=1 ; j++)
{
    Printf ("%d", a[i][j])
}
    Printf ("\n");
}
getch ();
}
```

Stacks

- Stack is a non - primitive linear data structure.
- It is an ordered list in which addition of new data item and deletion of already existing data item is done from only one end known as Top of Stack (TOS)

P.T.O.



- The Last added Element will be the first to be Removed from the stack This is the reason stack is called "Last - in - first - out" type of List.

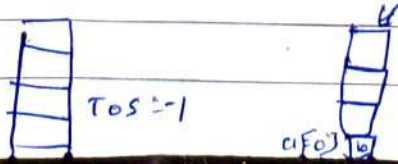
Operations on stack :-

There are two Operation on Stack.

(1) PUSH Operation :-

- The Process of adding a new element to the top of the stack is called PUSH Operation.

- Every new Element is adding to stack top is incremented by one.



- In case the array is fully and no new element can be added it is called stack full or stack overflow condition.

(2) POP Operation :

- The process of deleting an element from the top of stack is called POP Operation.
- After every POP operation the stack (TOP) is decremented by one.
- If there is no element on the stack and POP is performed then this will result into stack under flow condition. or Stack empty condition.

P.T.O.

Stack Operation & Algorithm:

- Stack has two operation.

Algorithm for inserting an item into the stack (PUSH Operation).

PUSH (stack [max size], item)

Step 1: initialize
Set top = -1

Step 2: Repeat steps 3 to 5 until TOP < maxsize - 1

Step 3: Read item

Step 4: set top = top + 1

Step 5: set stack [TOP] = item

Step 6: Print "stack overflow"

Ex: let insert 10, 20, 30, 40

max size = 3

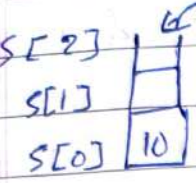
Top = -1

(2) TOP < maxsize - 1
-1 < 3 - 1
-1 < 2 True

(3) Read item
Read 10

(4) set top = top + 1
top = -1 + 1 = 0

(5) set stack [top] = item
stack [0] = 10



Top = 0

step (2) Top < maxsize - 1

0 < 3 - 1

0 < 2 True → 3, 4, 5

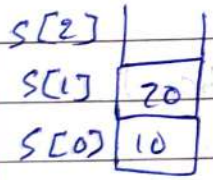
(3) Read ~~20~~ item

Read 20

(4) set Top = top + 1

Top = 0 + 1 = 1

(5) set stack [1] = 20



condition 4

step 2 Top < maxise - 1

2 < 3 - 1

2 < 2 false

Print Stack overflow.

condition

step (2) Top < maxsize - 1

1 < 3 - 1

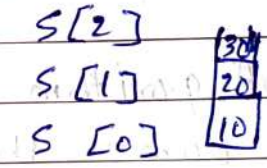
1 < 2 True

(3) Read 30

(4) set Top = Top + 1

Top = 1 + 1 = 2

(5) set stack [2] = 3



P.T.O.

POP Operation in Stack & Algorithm.

- The process of deleting an Element from the top of stack is called POP operation.
- After every POP operation the stack TOP is decremented by one

TOP = TOP - 1

Algorithm for deletion an item from the stack (POP)

POP (Stack [max size], item)

Step 1: Repeat steps 2 to 4 until top > 0

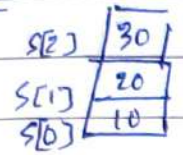
Step 2: set item = stack [ ]

Step 3: set top = top - 1

Step 4: Print, No delete

Step 5: Print stack underflows.

Ex



Case 1 Step ① TOP ≥ 0
2 ≥ 0
True

② Set item = Stack [TOP]
item = S [2]

③ Set top = TOP - 1
top = 2 - 1
top = 1

④ item deleted S[2]
30 deleted S[1] 20 TOP=1
S[0] 10

Case 2 Step - 1 TOP ≥ 0
1 ≥ 0
True

② Set item = Stack [TOP]
item = S [1]

③ Set top = TOP - 1
top = 1 - 1
top = 0

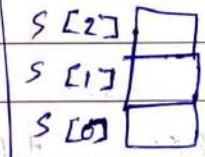
④ item deleted
20 deleted S[2]
S[1]
S[0] 10 TOP=0

Case 3
Step 1 TOP ≥ 0
0 ≥ 0
0 = 0 ~~False~~

② Item = Stack [TOP]
item = S [0]
item = 10

③ Set top = top - 1
top = 0 - 1
TOP = -1

④ delete item
deleted 10



Case 4:
Step TOP ≥ 0
-1 ≥ 0
False
go to step 5

Stack underflow

Stack (Prefix & Postfix)

Stack Notation:

There are three stack Notation.

(1) Infix Notation:- Where the operator is written between the operands.

Ex. (A + B) + operator , A, B operands.

(2) Prefix Notation: In this operator is written before the operands.

It is also known as polish Notation.

Ex. + AB

(3) Postfix Notation: In this operator is written after the operands.

It is also known as suffix Notation.

Ex. AB +

without stack implementation

Q convert the following infix to prefix and postfix for (A+B) * C/D + E ^ F/G

A/C to BODMAS

Prefix:

+AB * C/D + E ^ F/G

Let +AB = R1

R1 * C/D + E ^ F/G

let ^EFG R1 * C/D + ^EF/G

~~Let ^EF = R2~~ ⇒ R1 * C/D + R2/G

R1 * /CD + R2/G

let /CD = R3

R1 * R3 + R2/G

R1 * R3 + /R2G

let /R2G = R4

R1 * R3 + R4

* R1 R3 + R4

let * R1 R3 = R5

R5 + R4

+R5R4

⇒ putting the value of R1, R2, R3, R4, R5

+ * R1 R3 /R2G

+ * +AB /CD/ ^ E F G Prefix:

P.T.O.

Postfix : $(A+B) \star C/D + E \uparrow F/G$

$AB+ \star C/D + E \uparrow F/G$

Let $AB+ = R1$

$R1 \star C/D + E \uparrow F/G$

$R1 \star C/D + EF \uparrow /G$

Let $EF \uparrow = R2$

$R1 \star C/D + R2/G$

$R1 \star CD/ + R2/G$

Let $CD/ = R3$

$R1 \star R3 + R2/G$

$R1 \star R3 + R2G/$

Let $R2G/ = R4$

$R1 \star R3 + R4$

$R1\ R3 \star + R4$

Let $R1\ R3 \star = R5$

$R5 + R4$

~~R5~~ R5 R4 +

⇒ Now Enter the value of R1, R2, R3, R4 & R5

R5 R4 +

R1 R3 ★ R4

AB+ CD/ ★ R2G/ +

AB+ CD/ EF↑G/ +

Postfix

Prefix and Postfix using Tabular Form

Ex Convert (A+B ✱ C) into prefix and Postfix using tabular form.

To convert into prefix following operation perform.

(1) Reverse the input string
(2) Perform Tabular method and find Postfix Expression.
(3) Reverse this Postfix Expression string to find the Prefix.

Priority

^ ⇒ highest
✱, / → 2 highest
+ − → Lowest priority

Ex. A + B ✱ C

First to Add brackets

(A + B ✱ C)

Reverse string

(C ✱ B + A)

Tabular form

Symbol scanned	Stack	Postfix Expression
(	(	−
C	(	C
✱	(✱	C
B	(✱	CB
+	(+	CB✱
A	(+	CB✱A
	(−	CB✱A+
		P.T.O.

So the Postfix Expression CB*A+.
Now reverse this Expression to get the prefix is

+A*BC Answer

To Convert Postfix => Direct Perform Tabular form (A+B*C)

Symbol Scanned	Stack	Postfix Expression
(	(	Null
A	(	A
+	(+	A
B	(+	AB
*	(+*	AB
C	(+*	ABC
)		ABC*+

Postfix Expression ABC*+

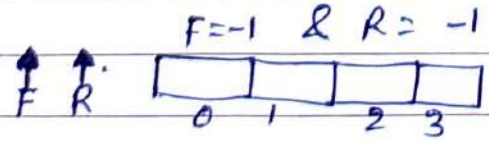
Queues

- Queue is an Non-primitive Linear data structure.
- It is an homogenous collection of elements in which new elements are added ~~the~~ at one end called "Rear end", the existing elements are deleted from other end called the "Front end".
- The First added element will be the first to be remove from the queue that is the reason queue is called (FIFO) first-in first out type List.
- In queue every insert operation Rear is incremented by one

$$R = R+1$$

and every deleted operation front end is incremented by one

Ex



Empty queue

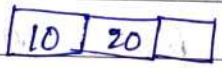
delete operation

(1) insert → 10

F = F + 1 = 0

R = R + 1 = 0

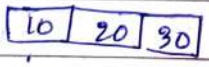
(2) insert → 20



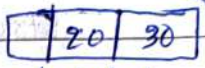
F = 0 R = R + 1

F = 0 = 0 + 1 = 1

(3) insert → 30



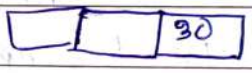
F = 0 , R = R + 1 = 1 + 1 = 2



10 deleted

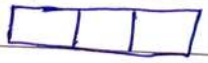
F = 0 + 1 = 1 R = 2

(2) delete 20



F = 1 + 1 = 2 R = 2

(3) delete 30



F = -1 R = -1

After performing all the operation Queue will be empty.

Operation On Queue

(1) To insert an element in a Queue =>

Algo => 0 Insert [QUEUE, [Maxsize], ITEM]

step 1: Initialization

set front = -1
set Rear = -1

step 2: Repeat steps 3 to 5 until
Rear ≤ maxsize - 1

step 3: Read item

step 4: If front = = -1 then
front = 0
Rear = 0

else
Rear = Rear + 1

step 5: set QUEUE [Rear] = item

step 6: Print, QUEUE is OVERFLOW

P.T.O.

Ex

maxsize = 4

Step-1

① F = -1 R = -1

② Rear < maxsize - 1
-1 < 4 - 1
-1 < 3 True

③ read item 10

④ if F = = -1 True
F = 0
R = 0

⑤ set q[0] = 10

F = 0 R = 0

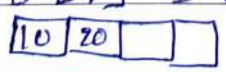
Second step

Step-2 Rear < maxsize - 1
0 < 4 - 1
0 < 3 True

③ Read item 20

④ if f = = -1 false
F = 0
Rear = Rear + 1
R = R + 1
= 0 + 1
= 1

⑤ set Q[1] = 20



3 condition

Step

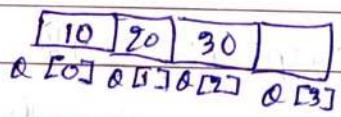
② Rear < maxsize - 1
1 < 3 True

③ Read item 30

④ If front = = -1
1 = = -1 False

Rear = Rear + 1
= 1 + 1
= 2

⑧ set q[2] = 30



4 condition.

Steps

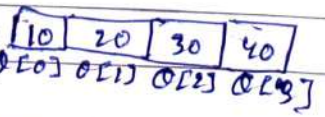
② Rear = maxsize - 1
2 < 3 True

③ Read item 40

④ if F = = -1 False
2 = = -1

Rear = Rear + 1
= 2 + 1
= 3

⑤ set Q[3] = 40



Condition 5

steps

(2) Rear < maxsize - 1

3 < 3

False

Queue is overflow

Delete an element from the queue

QDELETE (Queue [maxsize], item)

Step 1: Repeat step 2 to 4 until front >= 0

Step 2: set item = Queue [front]

Step 3: If front = = Rear

set front = -1

set Rear = -1

else

front = front + 1

Step 4: Print, No. Deleted is, item

Step 5: Print "Que is empty or underflow

P.T.O.

Ex

10	20	30
q[0]	[1]	[2]

max size = 3

condition ①

F = 0 R = 2

Steps

Step 2: F >= 0 True

Set item = q[0]

item = 10

3 If f = R

0 = = 2 false

front = front + 1

= 0 + 1 = 1

4 Print, No is deleted / item deleted

10 deleted

	20	30
q[0]	q[1]	q[2]

F = 1 R = 2

Condition ②

steps

1 F >= 0

1 >= 0 True

② set item q[1]

item = 20

③ if F = = R

1 = = 2 False

front = front + 1

= 1 + 1 = 2

④ Print, item 20 deleted

		30
q[0]	q[1]	q[2]

F = 2 R = 2

condition ③

steps

① F >= 0

2 >= 0

True

② set item = q[2]

item = 30

③ If Front = = R

2 = = 2 True

F = -1 , R = -1

④ Item deleted

30 deleted

empty queue.

[0]	[1]	[2]

F = -1 R = -1

condition ④

steps

① F >= 0

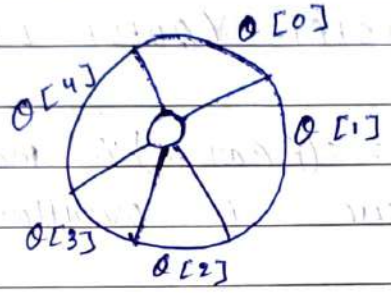
-1 = 0 false

step ⑤ Print queue is empty / underFlow.

deleted

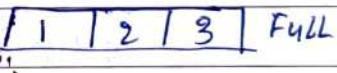
Circular Queue

A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the Last location of the queue is full.

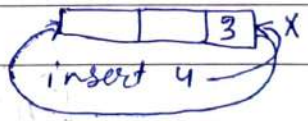


A circular queue over come the problem of unutilized space in Linear queues implemented as arrays.

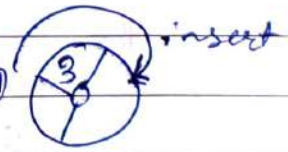
Circular queue has following conditions:



1) Front will always be pointing to the first element.



2) If Front = Rear the que will be empty.



3) Each time a new element is inserted into the queue the Rear is incremented by one.

Rear = Rear + 1

4) Each time an element is deleted from

P.T.O.

the queue the value of front is incremented by one.

Front = front + 1

Insert an Element in circular Queues

Algo ⇒ QINSERT (QUEUE [MAXSIZE], Item)

Step 1: if (front == (Rear + 1) % maxsize)
write queue is overflow & Exit.

else : take the value
if (front == -1)
set front = 0
Rear = 0

else
Rear = ((Rear + 1) % maxsize)

[Assign value] Queue [Rear] = value.
[End if]

Step 2 = Exit

Insert an Element in Circular Queues

maxsize = 3
F = -1
R = -1

let insert 10, 20, 30

if (front == -1)
F == -1
0 == -1 false

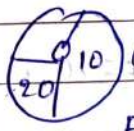
Steps

① if (front == (Rear + 1) % maxsize)
F == (R + 1) % maxsize
-1 == (-1 + 1) % 3
-1 == 0 % 3
-1 == 0 False

else
R = (Rear + 1) % maxsize
= 0 + 1 % 3
= 1 % 3
= 1

② else take value
take 10
set front = 0
Rear = 0

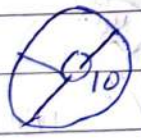
Assign value [Queue [Rear] = value
Q[R] = value
Q[1] = 20



Q[0]
10
20
Q[1]

F=0, R=1

[Assign value] Q [Rear] = value
Q[0] = 10



Q[0]
10
F=0, R=0

condition 3
step①
If F == (R + 1) % maxsize
0 == (1 + 1) % 3
0 == 2 % 3
0 == 1 False

condition ②
steps ① If (front == Rear + 1) % maxsize
if F == (R+1) % maxsize
0 == (0+1) % 3
0 == 1 % 3 False
0 == 1

else take value
take 20

else take value
take 30

if (front == -1)
F == -1
0 == -1 False

P.T.O.

else

R = (R+1) %. 3 (maxsize)

R = (1+1) %. 3

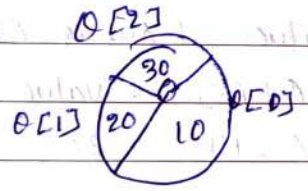
= 2 %. 3

= 1

[Assign value] : Queue [Rear] = value

Q·[R] = value

Q [2] = 30



F = 0, R = 2

condition (4)

steps (1) if (F = = (R+1) %. maxsize)

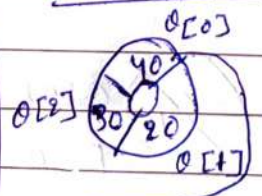
⇒ 0 = = (2+1) %. 3

0 = = 3 %. 3

0 = = 0 True

queue is overflow

Diff



inserted in empty queue

again check for insert

if (F = = (R+1) %. maxsize

1 = = (2+1) %. 3

1 = = 3 %. 3

1 = = 0 False

check same

and insert into empty.

DELETE an Element in circular queue

Algorithm → Q DELETE (QUEUE [maxsize], item)

steps: ① if (front = -1)
write queue underflow and Exit

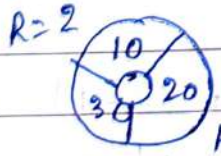
else item = Queue [Front]

if front = = Rear
set front = -1
set Rear = -1

else : Front = ((front +1) % maxsize)
[End if statement]
→ item deleted.

② exit.

Delete an element in circular queue



maxsize = 3

condition ①

steps

① if (front == -1)

0 == -1 false

else

item = Queue [front)

item = q [F]

= q [0]

item = 10

if (front == Rear)

F == R

~~set front~~ 0 == 2 false

else

front = ((front + 1) % maxsize)

F = (0+1) % 3

= 1 % 3

= 1

10 is deleted

R=2 (30, 20) F=1

Condition ②

steps

① if (front == -1)

1 == -1

false

else

item = Queue [front]

item = q [F]

= q [1]

item = 20

if (front == Rear

1 == 2

false

else

front = (front + 1) % maxsize

= (1 + 1) % 3

= 2 % 3

= 2

20 is delete

R=2 (30) F=2

Condition 3

steps ①

if (front == -1)

2 == -1

false

else

item = (Queue [front]

= q [2]

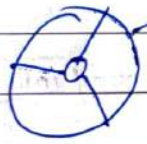
item = 30

item Else

front = (front + 1) % max size
= (2 + 1) % max size
= 3 % 3
= 0

if (front == Rear
(2 == 2)
true
F = -1
R = -1

30 is deleted



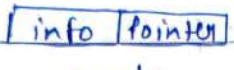
empty queue
F = -1
R = -1

Linked Lists

- A Linked List is a Linear data structure in which the elements are not stored in contiguous memory Location.
- A Linked List is a dynamic data structure.
The No of Nodes in a List is not fixed and can grow and shrink on demand.
- Each element is called a node, which has two parts.

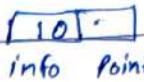
info part which stores the information and pointer which ~~stores~~ points to the next node.

next element.

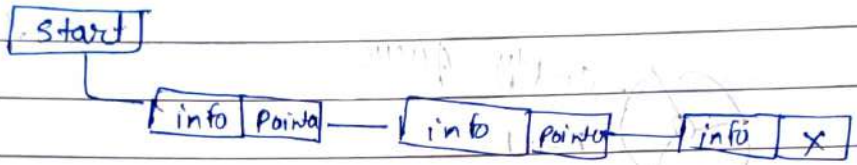


Node

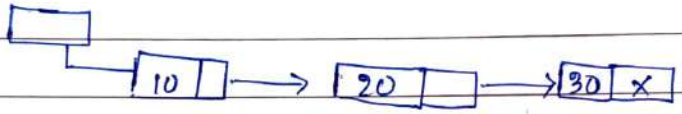
Ex.



Ex =>



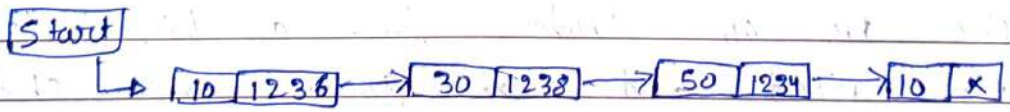
Ex



Ex.

Node

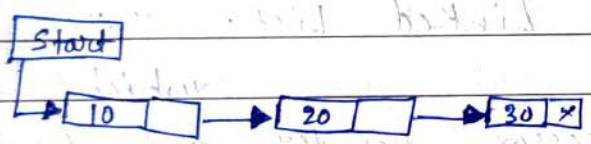
1234	10
1235	20
1236	30
1237	40
1238	50



Types of Linked Lists

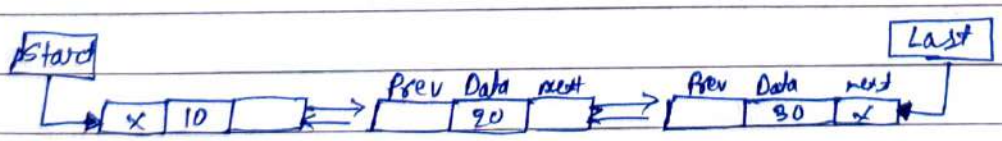
Basically there are four types of Linked list.

① Singly Linked List ⇒ It's one on which all nodes are Linked together in same sequential manner.
It's also called Linear Linked List.



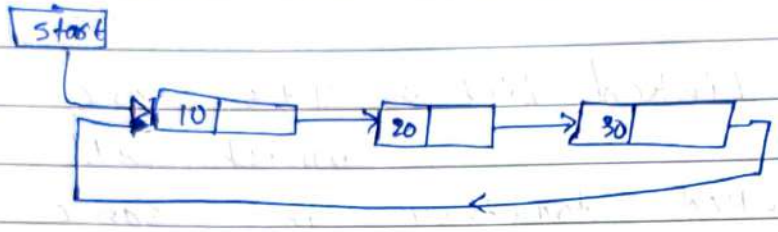
② Doubly Linked List: It's one in which all nodes are linked together by multiple ~~lists~~ Links which help in accessing both the successor node (next node) and predecessor node (previous node) within the List.

This helps to traverse the List in the forward and back ward direction.

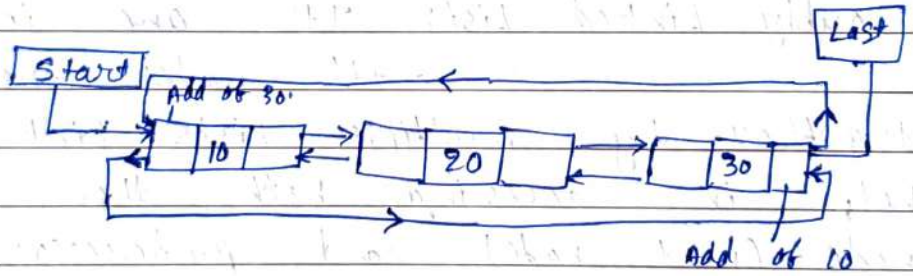


③ Circular Linked List: It's one which has no begining and no End. A singly Linked List can be made a circular linked List by simply

storing the address of the very first node in the Link field of the Last node.



(4) Circular doubly Linked List: It's one which has both the successor pointer and predecessor pointer in a circular manner.



Inserting Node in Linked List

1. Inserting at the begining of the List
2. Inserting at the end of the List.
3. Inserting at the specified position within the List.

1. Inserting A Node at the begining in the List:

Algorithm:

Algorithm:

INSERT_FIRST (START, ITEM)

Steps

Step ① [Check for overflow]
if ptr = NULL then
Print overflow
Exit

else

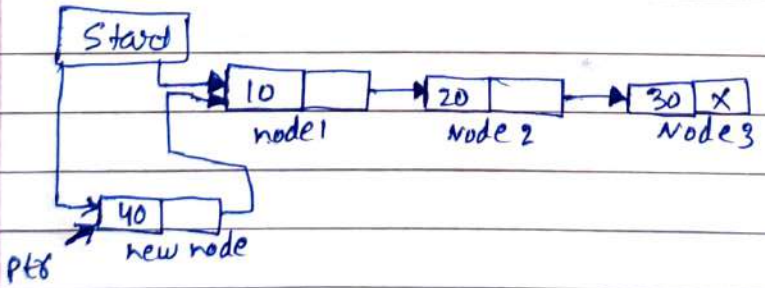
PTR = (Node *) malloc (size of (Node))

// create new node from memory and assign its address to ptr.

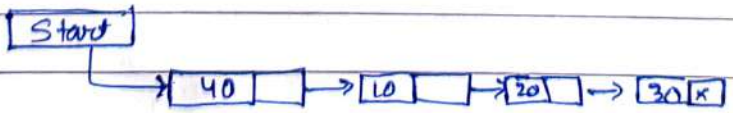
② set PTR → INFO = Item

③ set PTR → Next = START

④ Set START = PTR



After insertion



Insert A node At the End in Singly linked list

ALGORITHM:

Insert - Last (START, ITEM)

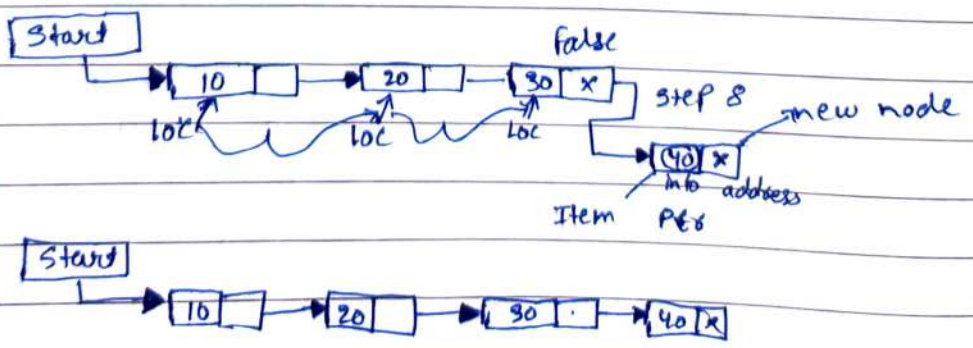
Steps:

① Check for overflow
if Ptr = NULL then
Print overflow
Exit

else
PTR = (Node*) malloc (size of (Node));

② Set PTR → Info = Item;
③ Set PTR → Next = NULL;
④ If Start = NULL and then
set START = Ptr;

else
⑤ Set LOC = Start;
⑥ Repeat step 7 until LOC → Next = NULL
⑦ set LOC = LOC → Next;
⑧ Set LOC → Next = Ptr;



Inserting A NODE at the specified Position in Singly linked List

ALGORITHM:

Insert - Location (START, Item, LOC)

steps :-

(1) Check for overflow

If Ptr = = NULL then

Print overflow

Exit

Else

Ptr = (Node *) malloc (size of (Node))

(2) Set Ptr → INFO = item

(3) If Start = NULL then

Set Start = Ptr

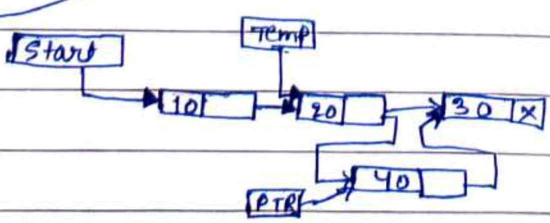
Set Ptr → Next = NULL

(4) Initialize the counter I and Pointers

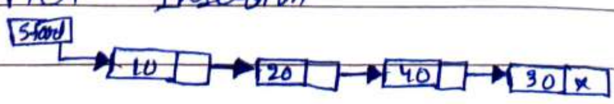
Set I = 0

Set temp = start

(after step 9)



After Insertion



(5) Repeat steps 6 and 7 until I < LOC

(6) set temp = temp → Next

(7) set I = I + 1

(8) set Ptr → Next = temp → Next

(9) set temp → Next = Ptr.

Deleting Node in Linked List

Deleting a node from the Linked List has three instances.

(1) Deleting the first node of the Linked List.

(2) Deleting the last node of the Linked List.

(3) Deleting the node from the specified position of the Linked List.

P.T.O.

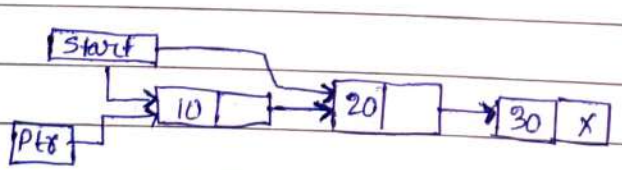
Deleting the first node in Singly Linked List

ALGORITHMS:

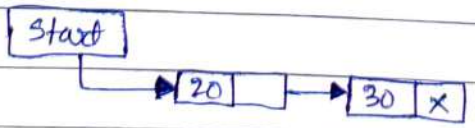
Deleted first (START)

Steps:

1. Check for underflow
   if start = NULL, then
   Print Linked List is Empty
   Exit
2. Set PTR = START
3. Set START = START → Next
4. Print element deleted is ptr → info
5. free (Ptr).



After deletion



Deleting the Last node in Singly Linked List

ALGORITHM:

Deleting (START)

Steps:

① Check for underflow
If Start = NULL then
Print LinkList is Empty
Exit

② if Start → Next = NULL then
set Ptr = Start
set Start = NULL
Print element deleted is = PTR → Info
free (PTR)
End if

③ set PTR = START

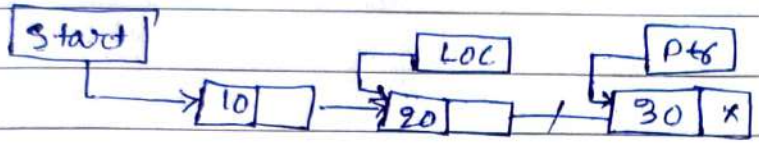
④ Repeat step 5 and 6 until
PTR → Next != NULL

⑤ set LOC = PTR

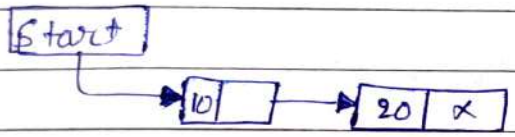
⑥ set PTR = PTR → Next

⑦ set LOC → Next = NULL

⑧ free (PTR)



After deletion



Deleting the Node from specified Position In Singly Linked List

ALGORITHM:

Delete - Location (START, LOC)

STEPS:

① check for underflow
if PTR = NULL then
print underflow
Exit

② Initialize the counter I and pointers
set I = 0;
set ptr = start;

③ Repeat step 4 to 6 until I < LOC

④ set temp = PTR

⑤ set PTR = PTR → Next

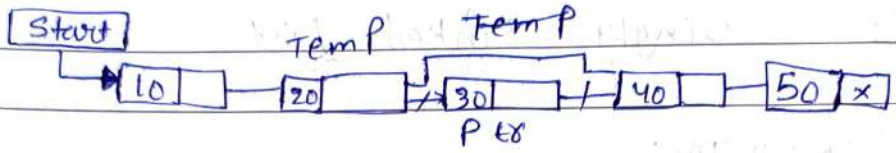
⑥ set I = I + 1

⑦ Print element deleted is = ptr → info

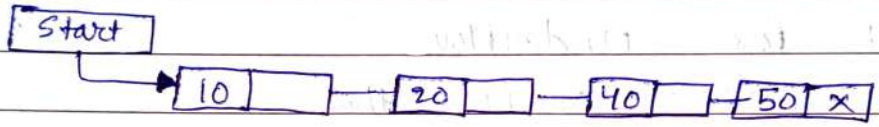
⑧ set Temp → Next = ptr → Next

P.T.O.

(9) Free (Ptr)



After deletion



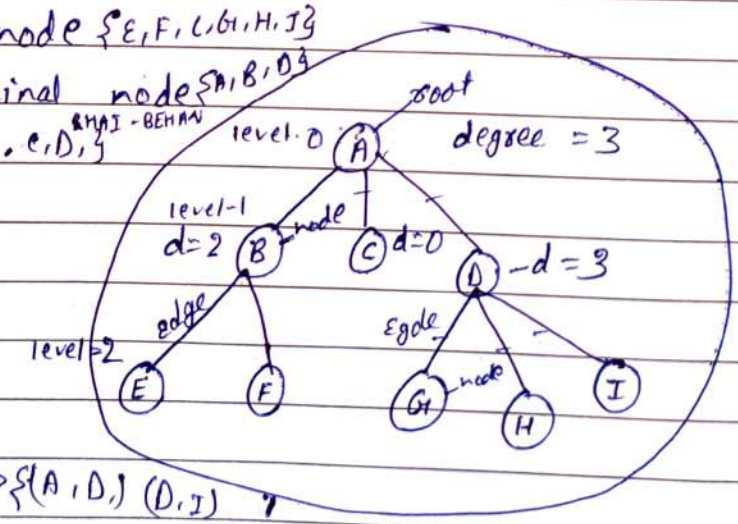
TREES IN DATA STRUCTURE

- A Tree is a non-linear Data Structure in which items are arranged in sorted sequences.

- It's a heirarchical data structure which stores the information naturally in the form of hierarchy style.

Tree Terminology:

1. Root
2. Node
3. Degree of a Node
4. Degree of a Tree
5. Terminal node {E, F, C, G, H, I}
6. Non-Terminal node {A, B, D}
7. Siblings {b, c, D,} BHAI-BEHAN
8. Level
9. Edge
10. Path
11. Forest

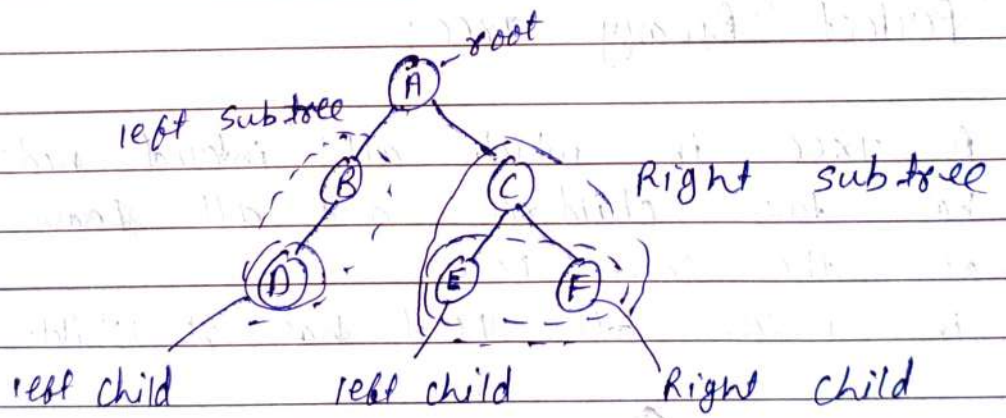


without root all tree is forest

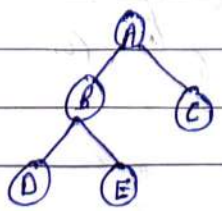
BINARY TREES

- Binary Tree is a finite set of data item which is either Empty of consists of a single item called root and two disjoint binary tree called the Left subtree and right subtree.

- In Binary tree, Every node can have matimum of 2 children which are known as Left child and right child.



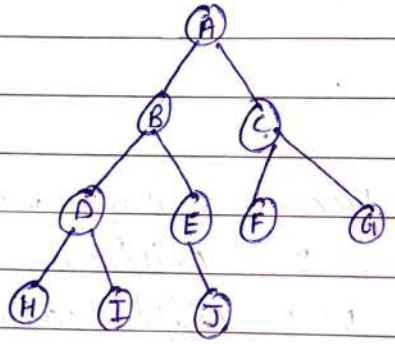
(1) Full Binary Tree: A Binary Tree is full if Every node has 0 or 2 child.



P.T.O.

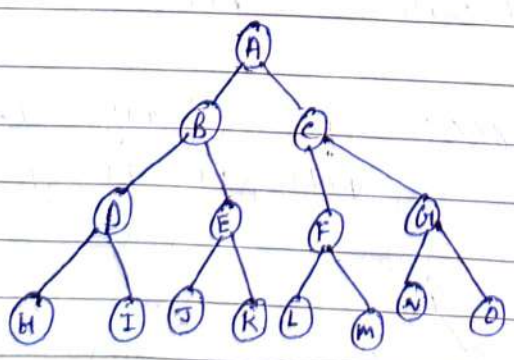
(2) Complete Binary Tree:

A Binary tree is completly ~~filled Except Possibly the~~ Binary Tree if all levels are completely filled Except Possibley the Last level and the Last level has all keys as Left as Possible.



(3) Perfect Binary Tree:

A Tree in which all internal nodes has two children and all Leaves are at the same Level.
in which all Level has 2^n child.



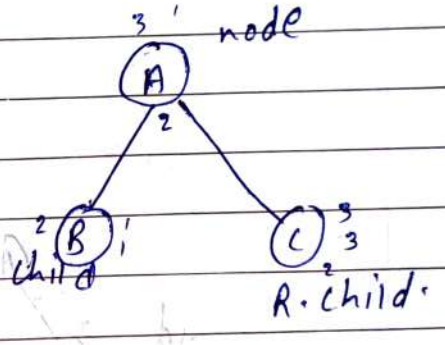
lev 0 = 2^0 = 1
1 = 2^1 = 2
2 = 2^2 = 4
3 = 2^3 = 8

Traversal of Binary Tree

It is a way in which each node in the tree is visited exactly once in a systemetic manner.

Three types :

1. Pre order NL R
2. In Order L N R
3. Post order L R N



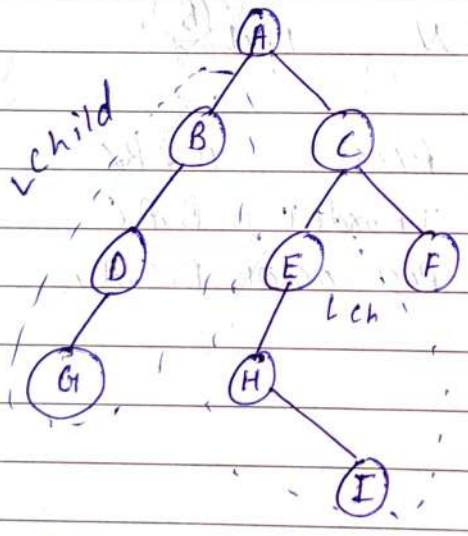
Pree A, B, C
Inorder B, A, C
Post order B, C, A,

P.T.O.

Preorder Traversal Binary Tree

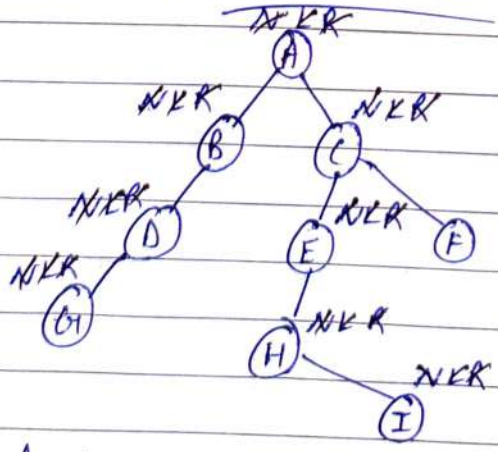
Preorder: N L R

Ex.



Preorder = A, B, D, G, C, E, H, I, F

Short way

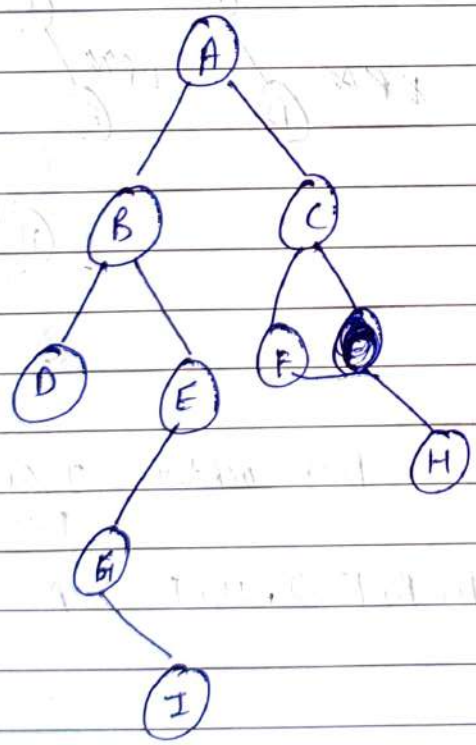


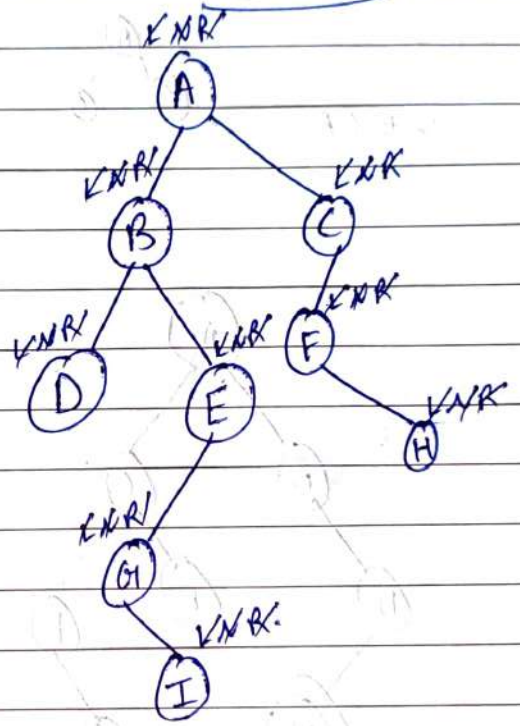
Preorder: A, B, D, G, C, E, H, I, F

In order Traversal Binary Tree

Inorder : LNR

Short



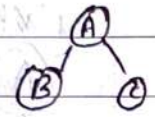


Inorder D, B, G, I, E, A, F, H, C

Inorder : D, B, G, I, E, A, F, H, C

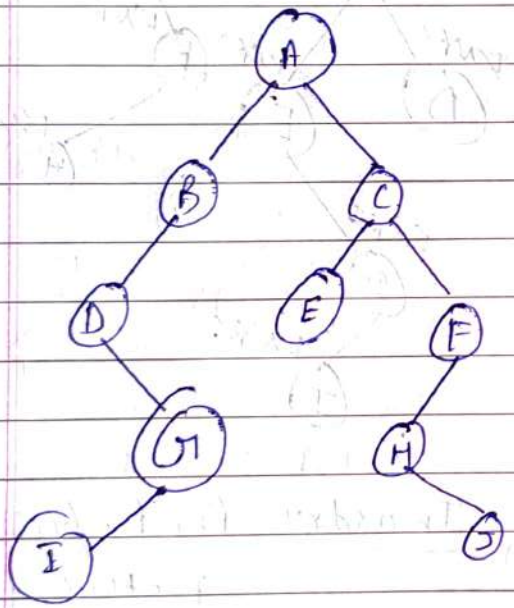
POST - order Traversal in Binary Tree

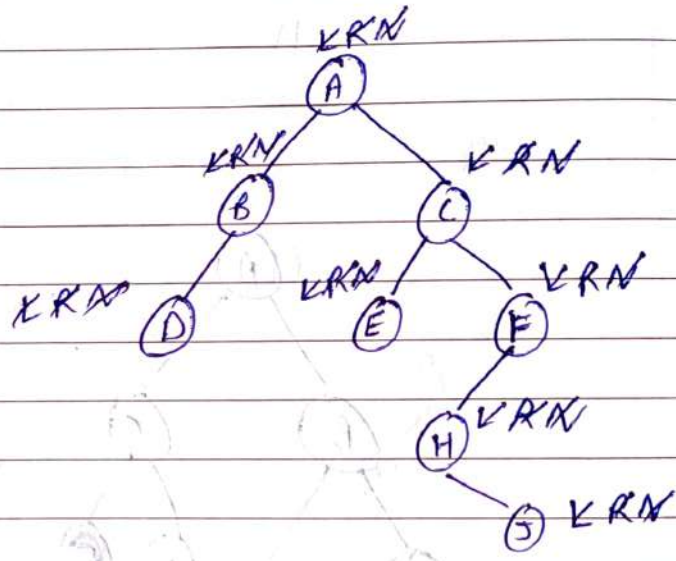
POST order L R N



⇒ B C A

Short





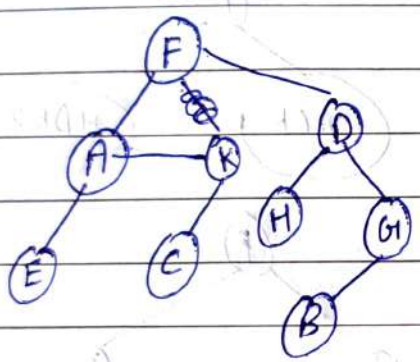
POST order : I, G, D, B, E, J, H, F, C, A

POST order: I, G, D, B, E, J, H, F, C, A

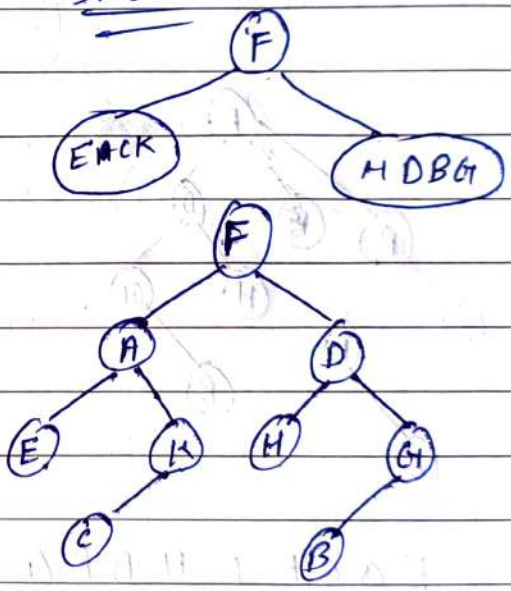
Draw a tree using inorder and Preorder traversal

LR Inorder : E A C K F H D B G

LR Preorder : F A E K C D H G B



Short

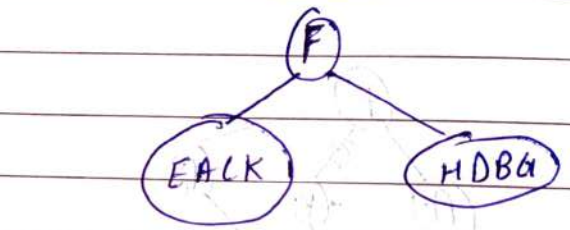


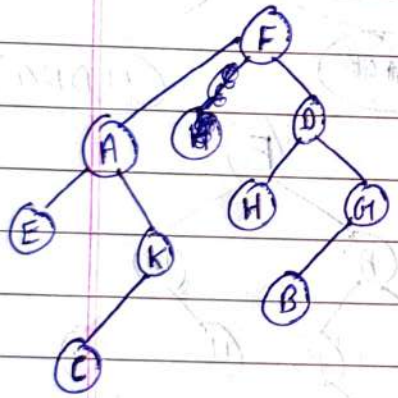
Draw a tree using inorder and postorder traversal

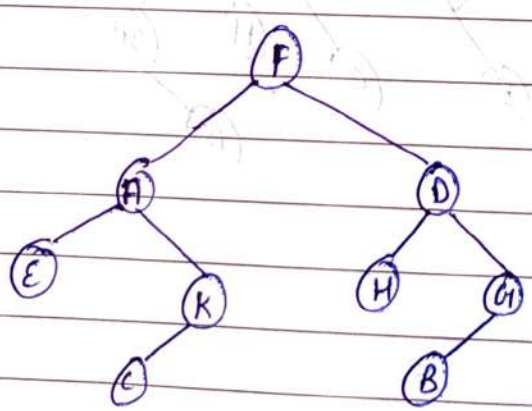
LNR Inorder : E A C K F H D B G

LRN Postorder : E C K A H B G D F

short







E A C K F H D B G

Binary search in Data structure

This search technique searches the given item in minimum possible comparisons. In this searching first we had to sort the Array Element Then following operation perform

(1) First Find the middle element of the array.

(2) Compare the mid element with an item.

(3) There are three cases:

(a) If it is a desired element then search is completed.

(b) If it's Less than desired item then search only the first half of the Array.

(c) If it's greater than the desired element search in the second half of the array.

P.T.O.

ALGORITHM:

Step 5:

① Set beg = LB, end = UB and mid = int $\left(\frac{beg+end}{2}\right)$

② Repeat steps 3 and 4 while beg <= End and a[mid] != item

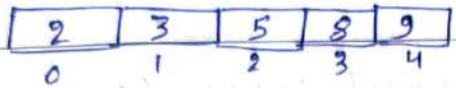
③ if item < a [mid] then
    set End = mid - 1
  else
    set beg = mid + 1

④ set mid = int $\left(\frac{beg + End}{2}\right)$ go to setep 2

⑤ if a[mid] = item then
    set LOC = mid
  else
    set LOC = NULL

⑥ Exit:

Ex: Search Element (8)



Step (i) beg = 0 End = 4 mid = int $\left(\frac{0+4}{2}\right)$ = 2

Step (ii) beg <= end & a [mid] != item

0 <= 4 & a[2] != 8

True 5 != 8 True

Step (iii) if item < a [mid]

8 < 5 False

Else

set beg = mid + 1

beg = 2 + 1 = 3

go to step (ii)

mid = int $\left(\frac{3+4}{2}\right)$ = 3

Now beg = 3, end = 4

beg <= end & a [mid] != item

3 <= 4 True a[3] != item

8 != 8 False

Jump to step (5)

8 = 8

Set Loc = mid

Loc = 3

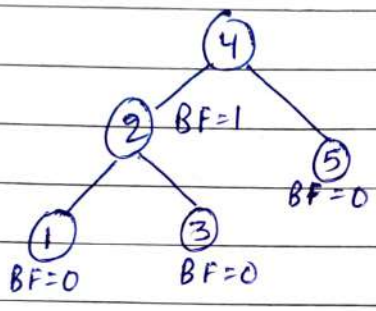
Exit

P.T.O.

AVL Tree

AVL tree is Balance BST or height Balance tree

Balance factor = Height of Left Subtree - Height of Right Subtree



BF = 2 - 1 = 1

*Balance factor : height - of - left subtree

Condition of AVL Tree :-

If B.F ⇒ 0, 1, -1 then it is Balanced tree

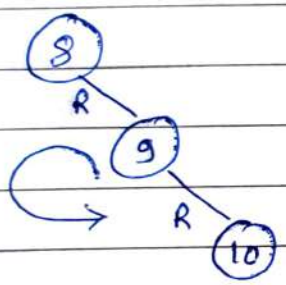
If another found then we have to rotate the tree.

(i) Left
(ii) Right
(iii) Left - left
(iv) Right - Right

AVL Tree ROTATION

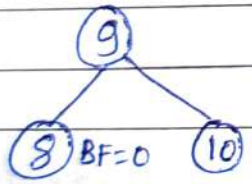
1. Right - Right
2. Left - Left
3. Right - left
4. Left - Right

(1) (Ex) 8, 9, 10



BF = 0 - 2 = -2

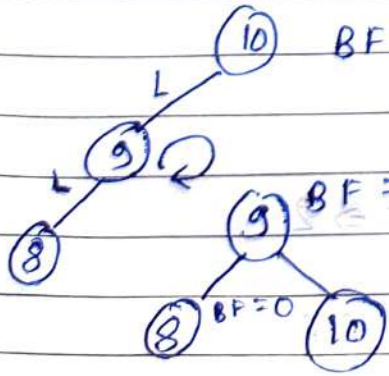
Then,



BF = 1 - 1 = 0

One rotation
Right - Right Rotation.

(2) (Ex) 10, 9, 8

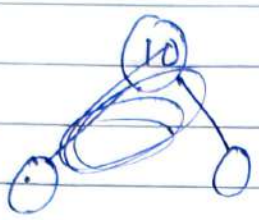


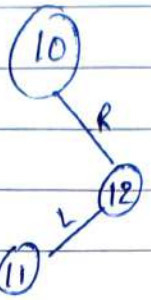
BF = 2 - 0 = 2

BF = 1 - 1 = 0

One rotation
left left rotation.

(3) Ex 10, 12, 11

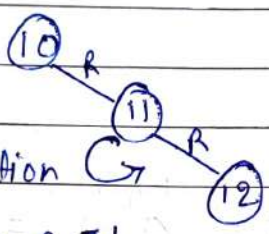




BF = 0 - 2 = -2

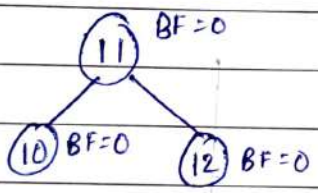
RL → RR

Then,



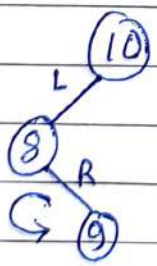
First / one rotation

RR → RL



second rotation

4 Ex 10, 8, 9

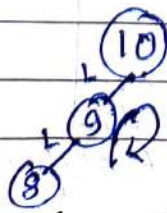


BF = 2 - 0 = 2

→ LR → LL

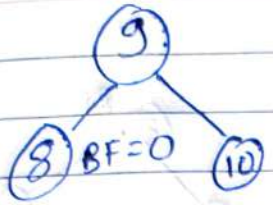
First rotation

Now,



BF = 2-0 ⇒ 2

→ LL → RR



second rotation.

Creating AVL TREE

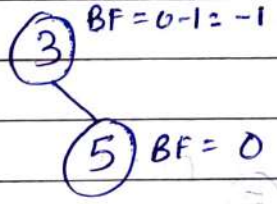
B.F = h of l − h of R

Create AVL Tree ⇒

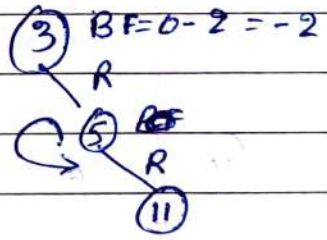
Ent, 3, 5, 11, 8, 4, 1, 12, 7, 2, 6

① 

② 5 =

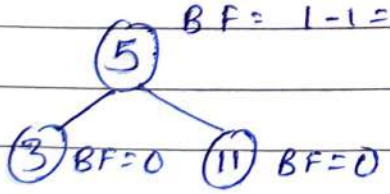


③ 11

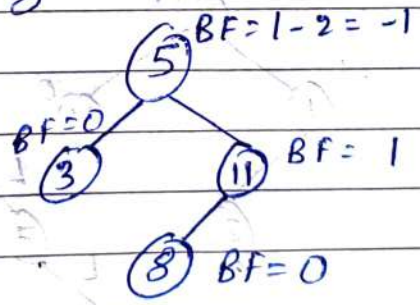


after rotation

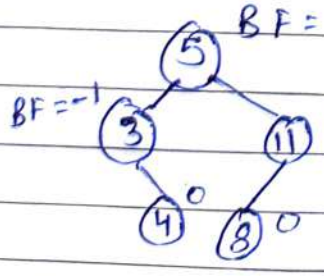
RR → RL



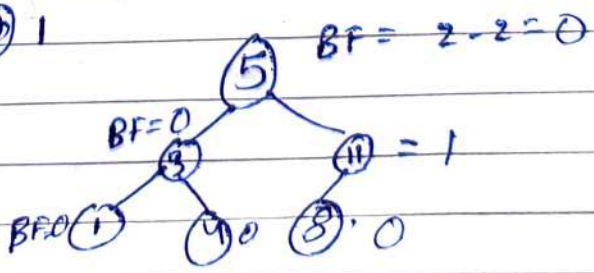
④ 8



⑤ ④

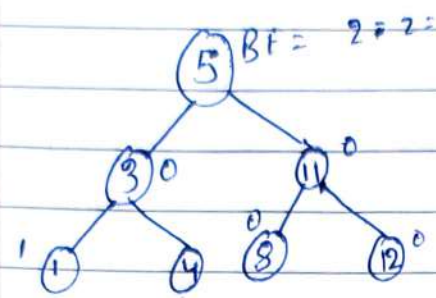


⑥ 1

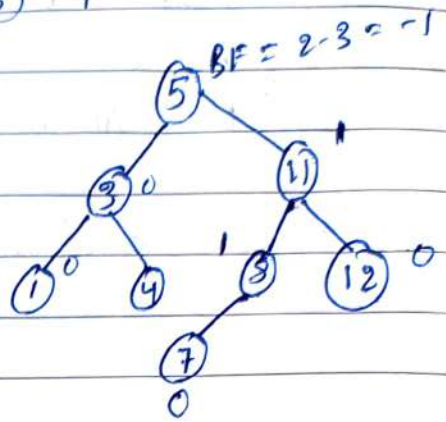


P.T.O.

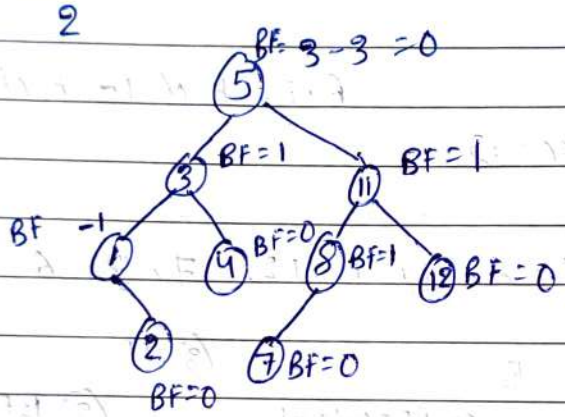
7) 12


BF = 2-2 = 0
5
3
11
1
4
8
12

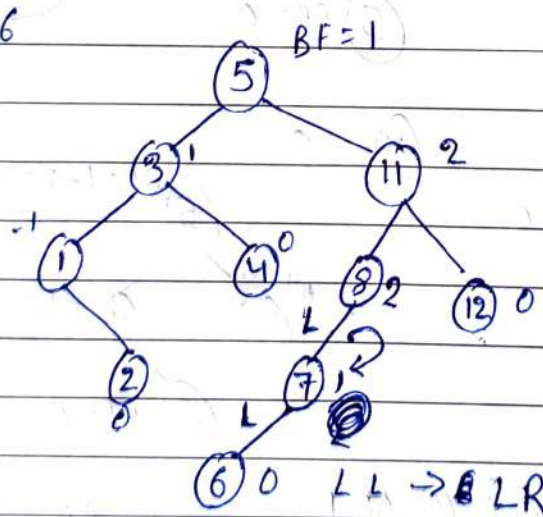
8) 7

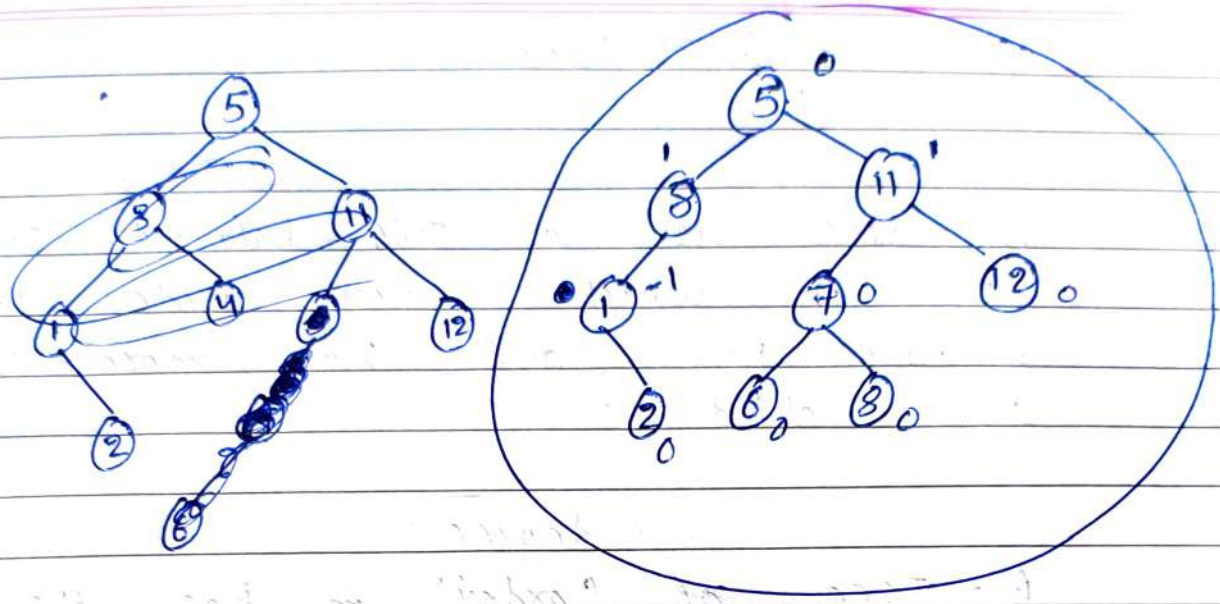

BF = 2-3 = -1
5
3
11
1
4
8
12
7

9) 2


BF = 3-3 = 0
5
3
11
1
4
8
12
2
7
BF = 1
BF = 1
BF = -1
BF = 0
BF = 1
BF = 0
BF = 0
BF = 0

10) 6


BF = 1
5
3
11
1
4
8
12
2
7
6
L
L
LL → LR



Actual AVL Tree / output

B - Tree

B - tree is a self - balanced Search Tree in which every node contains multiple keys and has more than two children.

degree
B - Tree of (order) m has the following properties :-

1. All Leaf nodes must be at same level.

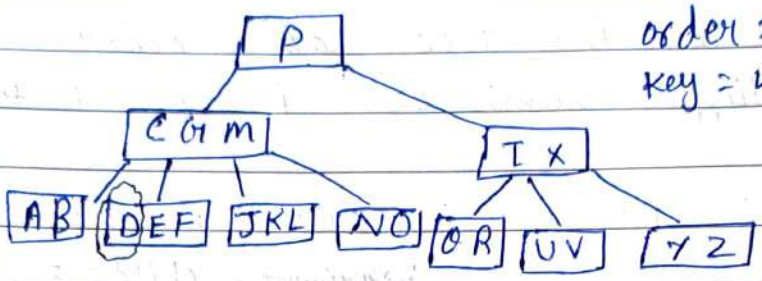
2. All the nodes except root must have atleast $\lceil m/2 \rceil - 1$ keys and maximum of $m - 1$ keys.

3. Each node has a maximum of m children and a minimum of $m/2$ children.

4. All the key value in a node must be in Asending order.

B - Tree Deletion

Ex

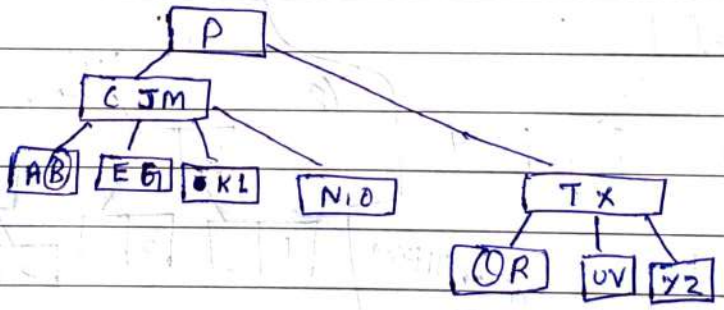


order = 5
key = 4

Case ① Delete D condi: more than two delete I directly

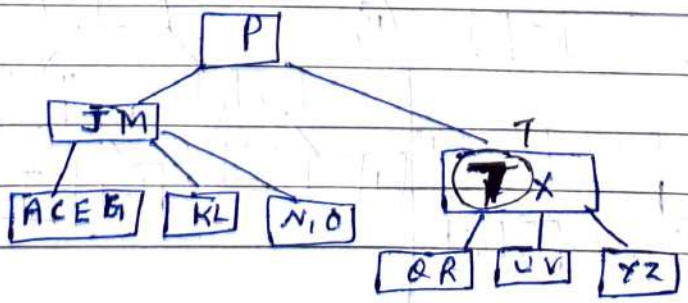
Case ② Delete F

condition: Less than three fill the value with upper node which is parallel to it and next sibling's value will replace with the value of node.



Case 3: Delete B

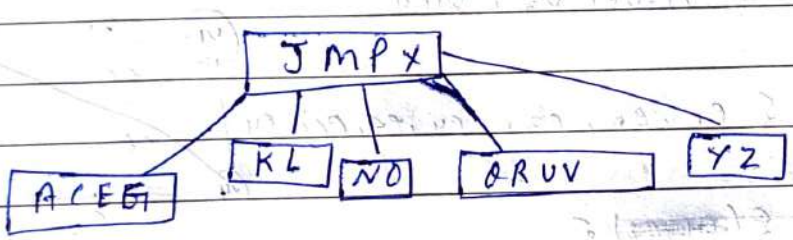
condition: Deleted have also 2 keys (child) then the parent will replace to deleted and child connect to the next sibiling.



Case. 4: Delete T

condition: if Node and its child also have two key's then all the nodes will combined by their parent / root node:

If Node's child have three keys then deleted key will be replaced to their child.



Ans.

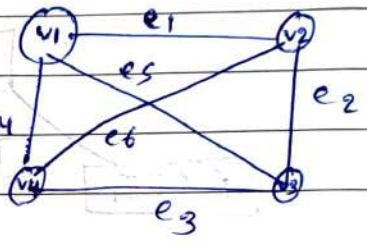
Introduction to Graphs

- A graph 'G' is an ordered pair of a set 'V' of vertices and a set E of edges

$$G = (V, E)$$

Ex $V = \{ v_1, v_2, v_3, v_4 \}$

$E = \{ e_1, e_2, e_3, e_4, e_5, e_6 \}$



~~$E = \{ (\quad, \quad) \}$~~

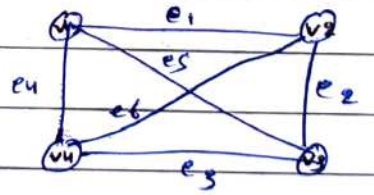
~~$E = \{ (v_1, v_2), (v_3, v_4), (v_5, v_6), \ldots \}$~~

$E = \{ (v_1, v_2), (v_2, v_3), (v_3, v_4) \cdots \}$

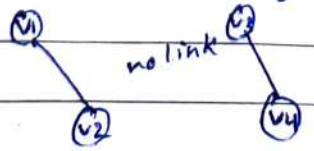
Types of Graphs

(1) Connected Graph:
In connected graph two edges connected to each other.

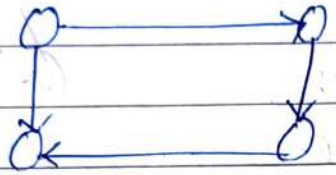
Ex.



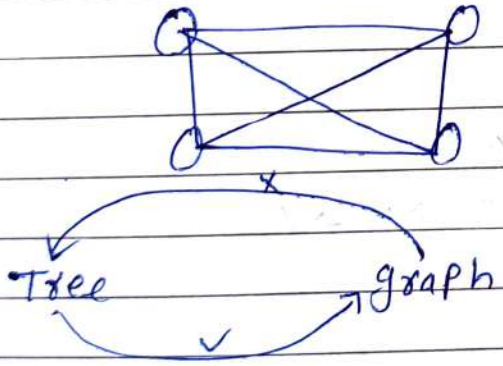
(2) Un connected Graph:
In unconnected graph There is no link to connect two edges.



① Directed Graph:
In Directed graph we have given the direction by which we know the way to go by ~~edges~~. Vertices.



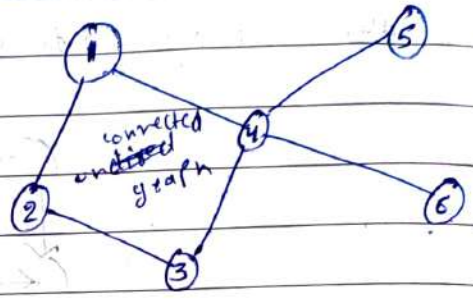
② Undirected Graph:
There is no direction given to go all vertices are connected directed



All tree will be graph because no cycle create.
NO graph will be tree because circle will create. here

Basic Terminologies of Graph in DS

1. Adjacent vertices
2. Path
3. Cycle
4. Degree
5. Complete graph
6. Weighted graph



(1) Adjacent vertices

2 = 1, 3

4 = 1, 3, 5, 6.

(2) Path.

1 → 6 =

(2.) 1 → 4 → 6

(2.) 1 → 2 → 3 → 4 → 6

(3) Cycle

1 → 2 → 3 → 4 → 1 Starting end same

(4) Degree other node connected by that node

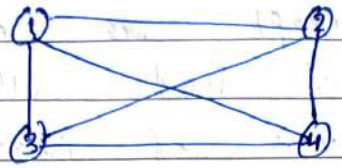
1 ⇒ 2 degree

4 = 4 degree

2 = 2 degree

(5) Complete graph: All node connected.

$$\text{Edges} = \frac{n(n-1)}{2}$$



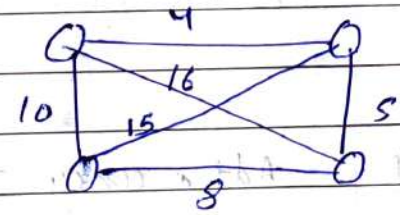
$n = 4$

So,

$$\frac{4(4-1)}{2} = \frac{4 \times 3}{2} = 6$$

Edges = 6

(6) Weighted graph:



Graph Representation in D.S.

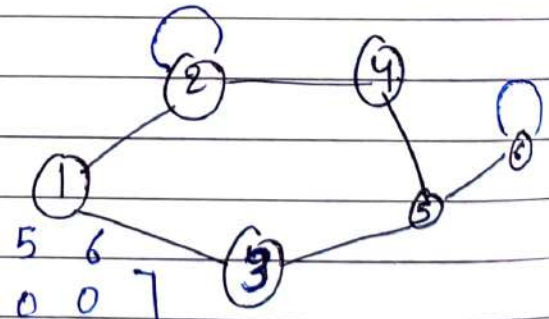
* Graph is a mathematical structure and finds its Application in many Areas of interest in which problems need to be solved using computers

There are two types of graph Representation.

(1) Adjacency matrix Representation

(2) Adjacency List Representation.

(1) Adjacency matrix Representation; non connected graph.

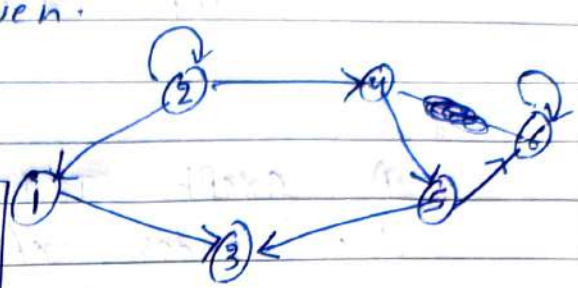


v i j = $v_{1,2} \Rightarrow 1$ present

$v_{1,2} \Rightarrow 0$ not present

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	1	0	1	0	0
3	1	0	0	0	1	0
4	0	1	0	0	1	0
5	0	0	1	1	0	1
6	0	0	0	0	1	1

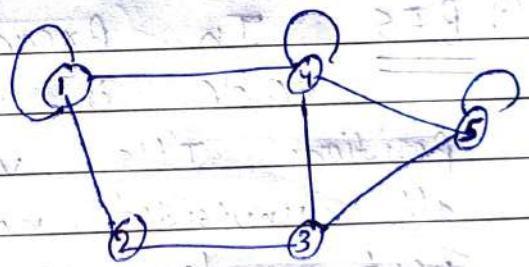
Let direction given.

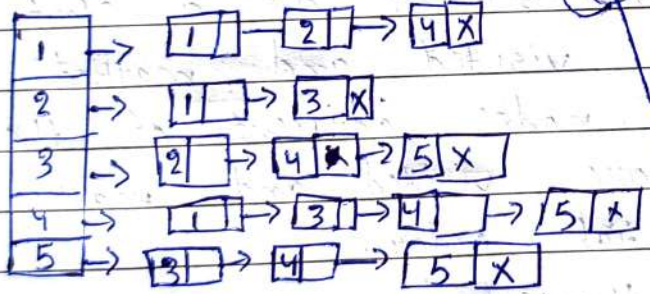


	1	2	3	4	5	6
1	0	0	1	0	0	0
2	1	1	0	1	0	0
3	0	0	0	0	0	0
4	0	0	0	0	1	0
5	0	0	1	0	0	1
6	0	0	0	0	0	1

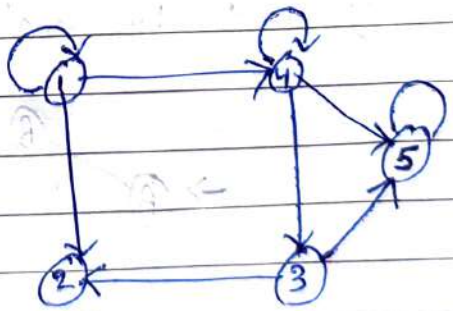
② Adjacency List Representation

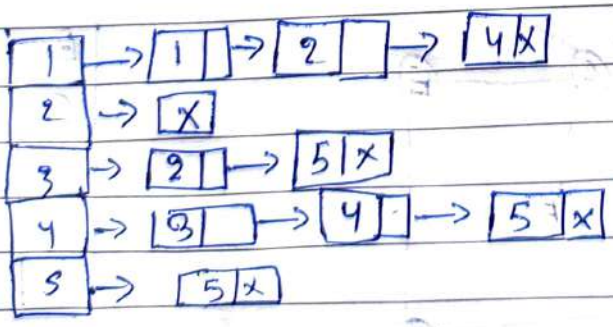
Non connected





Direct graph





GRAPH Traversal in D.S.

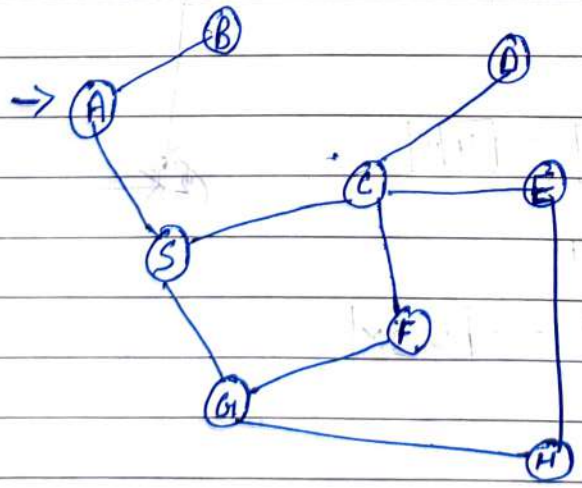
A graph Traversal means visiting all the nodes of the graph.

There are two types of graph Traversal.

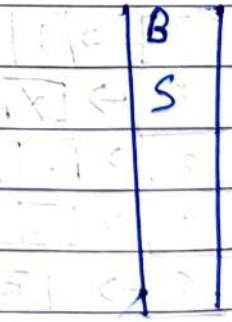
1) Breadth first Traversal / search (BFS)
2) Depth first Traversal / search (DFS)

(I) BFS In Breadth first search, One node is selected as a start position. It's visited and marked, then all unvisited nodes adjacent of the next node are visited and marked in some sequential order.

(I) level by level traversal

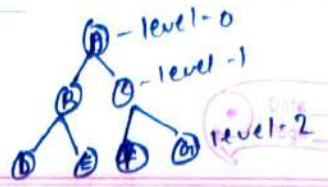


queue status



output: A, B, S, C, G, D, E, F, H

in tree form



Now B

No, adjacent node of B

Now B, remove, S will take place

Queue status

S

Now S

C, G1

Queue status

C
G1

Now C

D, E, F.

G1
D
E
F

Now, G1

F, H

E
F
H

Now, D

no adjacent

Now, E

F
H

no

Now F

no

H

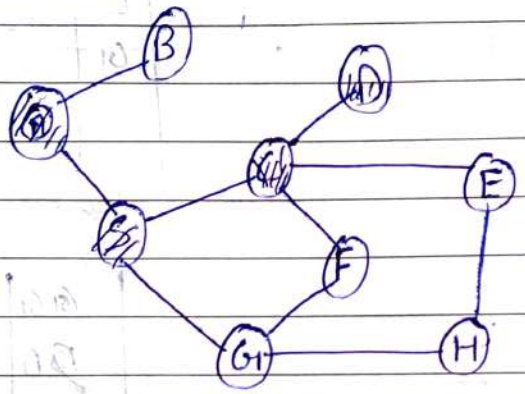
Now H

no

Queue Status

(11) Depth First Search (DFS)

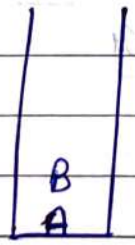
DFS => Depth first search (DFS) follows first a path from the starting node to an ending node, then another path from the start to the end. and so forth until all nodes have been visited.



output : A, B, S, C, D, E, H, G, F

Ans

Stack :- (LIFO)



Now B

no adjacent

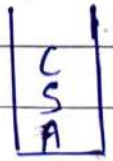
B POP

stack

Now S

C,

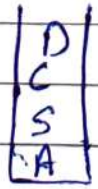
stack



Now C

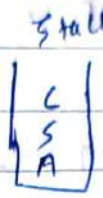
D

stack



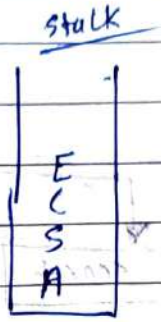
Now D

no adjacent

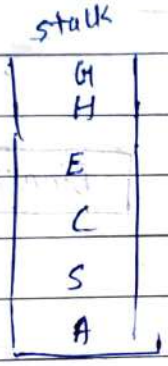


Now again C

E → H

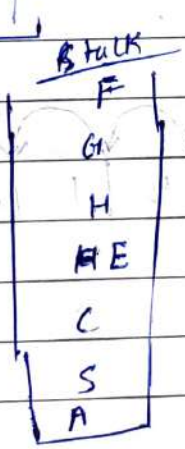


Now H



Now G

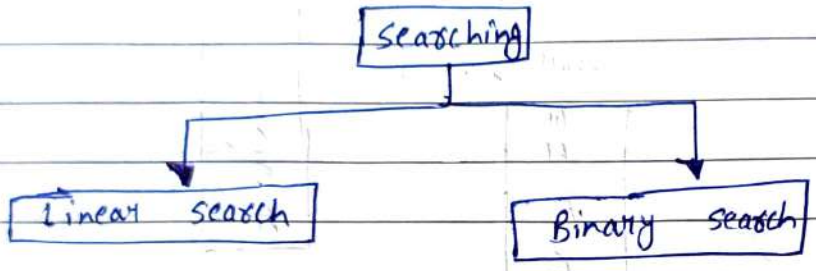
→ F



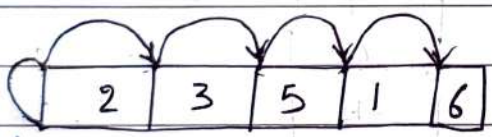
all will pop one by one because all visited and no adjacent remain

Introduction to searching in D.S

★ Searching is a Process to finding an element within the List of elements stored in any order.



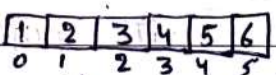
Linear search :



Find 1

Binary search

First Sort

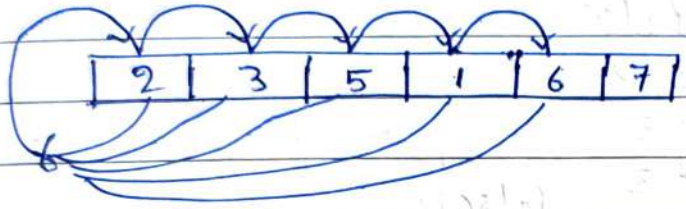


Find mid = F + L /2

$= \frac{0+5}{2} = 2.5 = 2$

Linear search in Data structure

In Linear search, we access each element of an array / List one by one sequentially and see whether it is desired element or not.



ALGORithm:

(i) $i = 0$
(ii) if $i > n$; go to step 7 [n = max size of Array
(iii) if $A[i] = x$; go to step 6
(iv) $i = i + 1$
(v) Go to step 2
(vi) Print Element x found at i
(vii) Print Element not found
(viii) Exit

P.T.O

(Ex)

2	4	6	3	1	5
0	1	2	3	4	5

Steps:

(I) $i = 0$, $x = 3$, $n = 6$

Steps

(II) $0 > 6$ false

(III) $A[i] = 3$

$A[0] = 3$

$A\ 2 = 3$ false

(IV) $i = i + 1 \Rightarrow 0 + 1 = 1$

(V) go to step 2 $1 > 6$ false

(III) $A[i] = 3$

$A[1] = 3$

$4 = 3$ false

(IV) $i = i + 1 \Rightarrow 1 + 1 = 2$

(V) Go to step 2

(II) $2 > 6$ false

(III) $A[i] = 3$

$A[2] = 3$

$6 = 3$ false

(IV) $i = i + 1 \Rightarrow 2 + 1 = 3$

(V) Go to step (II)

(II) $3 > 6$ false

(III) $A[i] = 3$

$A[3] = 3$

$3 = 3$ True

Go to step 6

(vi) Print 3 , 3

(vii) Exit

SORTING IN DATA STRUCTURE

Sorting: Sorting refers to the operation of Arranging data in some given sequence such as increasing order or decreasing order.

Type of sorting:

1) Bubble sort
2) selection sort
3) Insertion Sort
4) quick sort
5) heap sort
6) merge sort
7) Radix or Bucket sort.

P.T.O.

(1) Bubble sort :-

In Bubble sort, each element is compared with its adjacent element. If the first element is larger than the second one then the position of the elements are interchanged, other wise it is not changed.

(Ex)

1) Pass

10	10	10	10	10
14	14	2	2	2
2	2	14	11	11
11	11	11	14	6
6	6	6	6	14
	no change	swap	swap	swap

2) Pass

10	
2	(in first pass greater value
11	will be at last)
6	
14	

Pass 2 ⇒

10	2	2	2	
2	10	10	10	(in 2nd pass 2nd
11	11	11	6	largest will be at
6	6	6	11	2nd last.)
14	14	14	14	
swap	no change	swap		

Pass 3

2 ← no swap	2	2
10 ← swap	10 ← swap	6
6	6 ←	10
11	11	11
14	14	14

(3rd largest completed

Output

2
6
10
11
14

P.T.O.

Bubble sort Algorithm with Example

Algo

Step 1 → Initialization I = 0

step 2 → Repeat step 3 to 5 until I < N

step 3 → set J = 0

step 4 → Repeat step 5 until J < N - I - 1

step 5 → If A [J] > A [J + 1] then

set Temp = A [J]
set A [J] = A [J+1]
set A [J+1] = temp

step 6 ⇒ Exit.

Program :

```
# include <stdio.h>
# include <conio.h>

 void main ()
{


    for (i =0; i<N; i++)
    {
      for ( j=0; J < N - I -1)
      {
        if ( A [j] > A [J+1] )
        {
            set Temp = A [J]
            set A [J] = A [J+1]
            set A [J+1] = temp;
        }
      }
    }
getch ( );
}
```

P.T.O.

Ex

8	1	7	2	5
a[0]	1	2	3	4

$N = 5$

Steps

① $J = 0$

② $0 < 5$ True

③ set $J = 0$

④ $J < N - i - 1$

$0 < 5 - 0 - 1$

$0 < 4$ true

⑤ if $A[J] > A[J+1]$

$A[0] > A[0+1]$

$A[0] > A[1]$

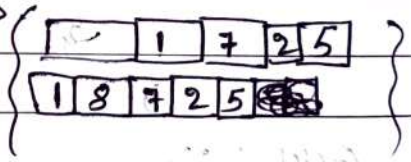
$8 > 1$

True

set temp = 8

set $A[J] = A[J+1]$

set $A[J+1] =$ temp



Condition ②

Now $J = 1$

Steps

① $J = 0$

② $0 < 5$ True

③ set $J = 1$

④ J < N - i - 1

J < 5 - 0 - 1

J < 4 True

⑤

If A[J] > A [J+1]

A[1] > A [2]

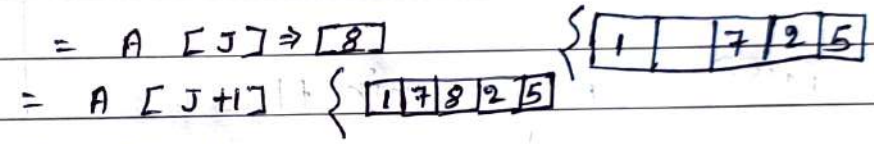
8 > 7

True

set Temp = A [J] ⇒ [8]

set A[J] = A [J+1]

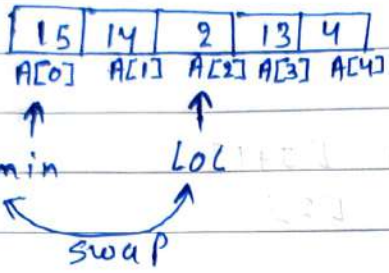
set A[J+1] = Temp.



To be continued until complete Thankyou

Selection Sort in Data structure

Ex

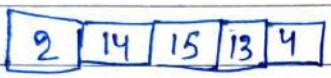


max size = 5

① Make a min 1st element

② Location = Small

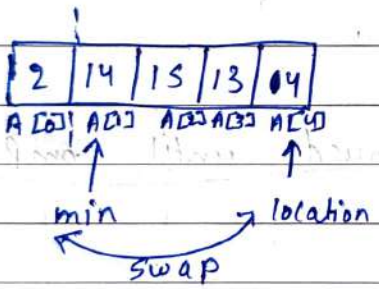
Pass ①



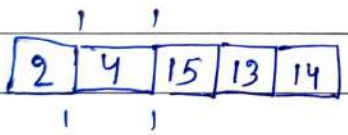
LOC = 2

Pass ②

min = min + 1



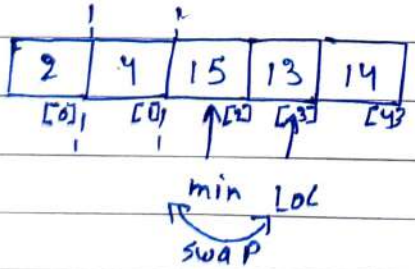
LOC = 4



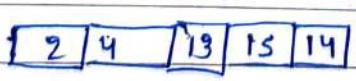
{LOC = 4}

Pass 3

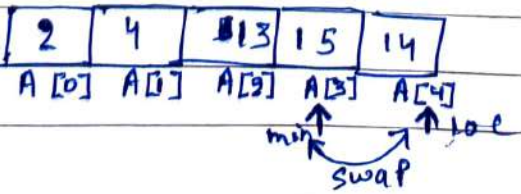
min = min + 1



LOC = 3

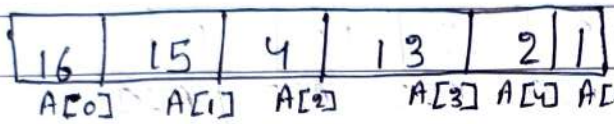


Pass 4



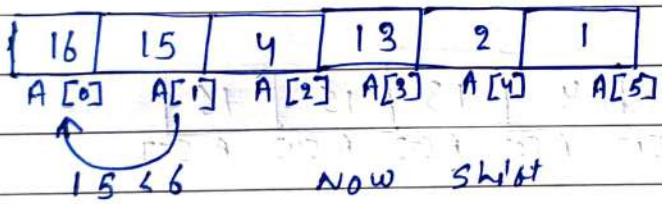
~~Sorting~~ Sorted

INsertion sort in data structure

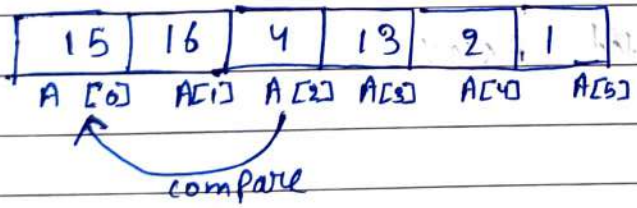
(Ex) 

max size = 6

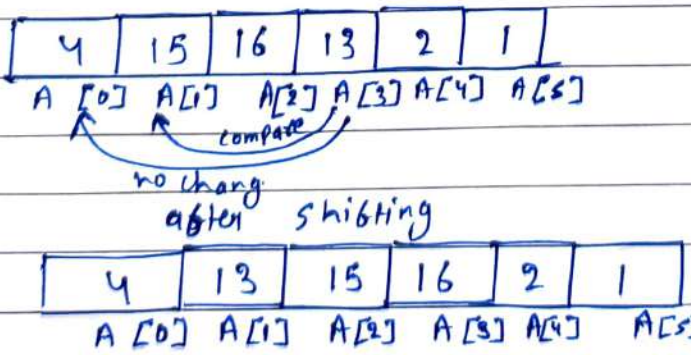
(1) Start with 2nd element
(2) compare with Back / previous (next) element
if small then shift



Pass 2

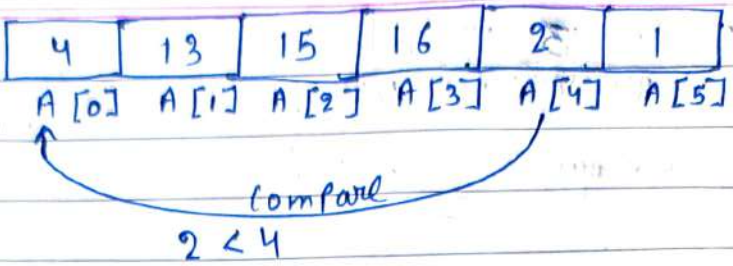


Pass 3

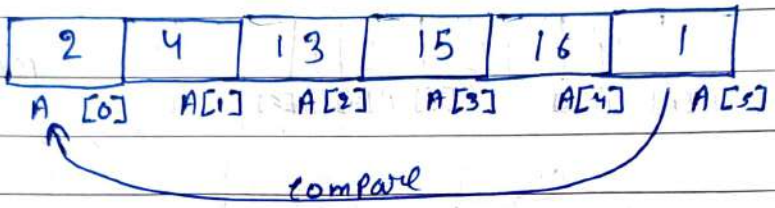


P.T.O.

Pass 4



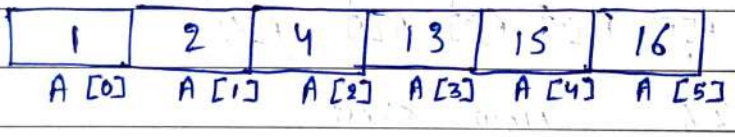
after shifting



Pass 5

1 < 2

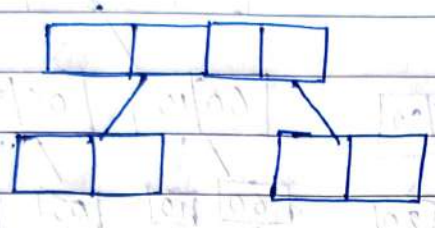
after shifting.



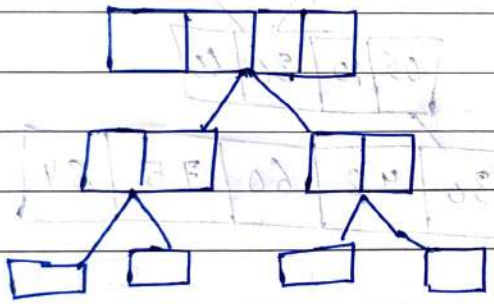
Sorted Array

Merge sort in Data structure
(Divide, conquer & combined)

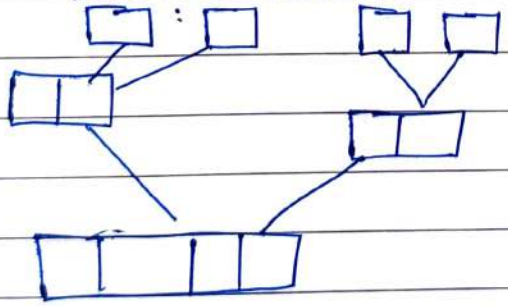
(1) Divide rule into two part



(2) conquer (separate)



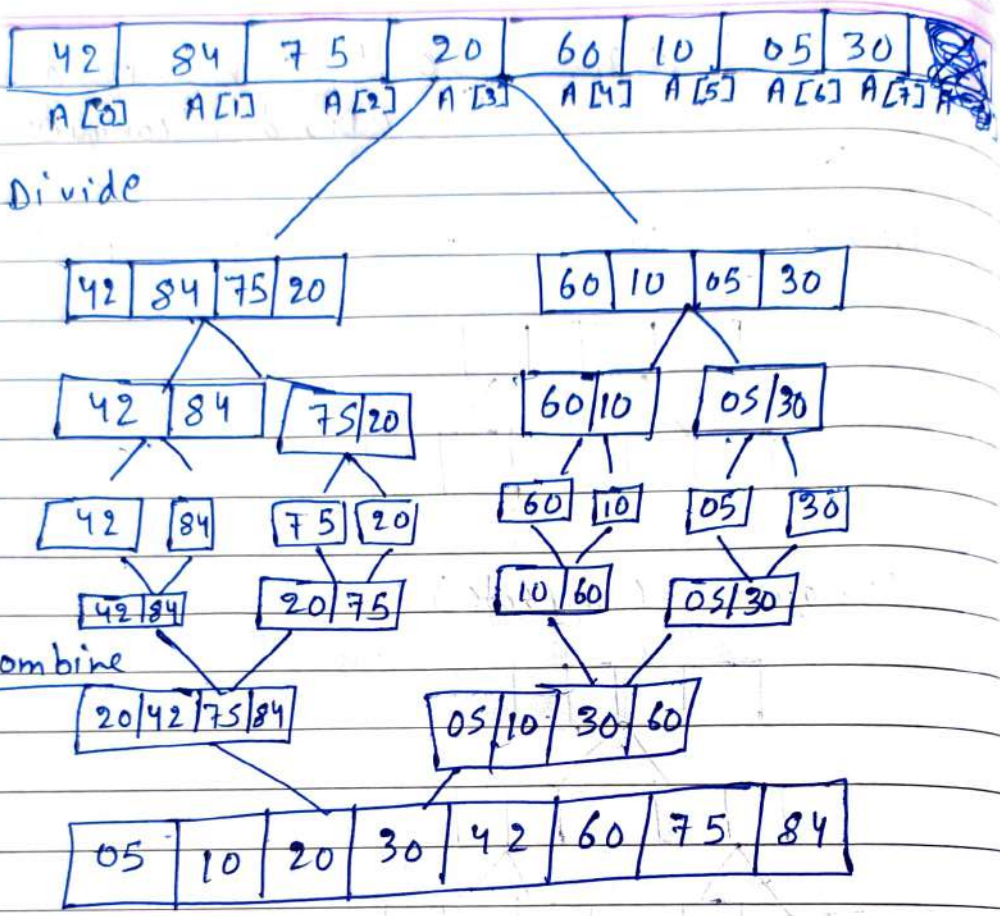
(222) merge sort (combined)



output

P.T.O.

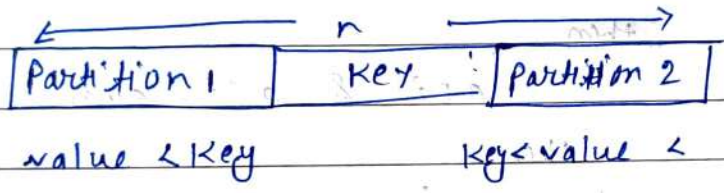
Ex


42 84 75 20 60 10 05 30
A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7]
Divide
42 84 75 20
60 10 05 30
42 84
75 20
60 10
05 30
42
84
75
20
60
10
05
30
conquer
42 84
20 75
10 60
05 30
Combine
20 42 75 84
05 10 30 60
05 10 20 30 42 60 75 84
Sorted Array

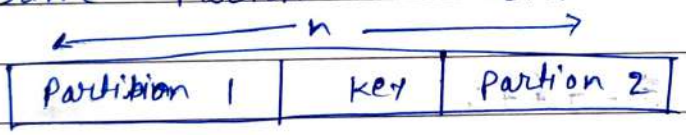
Quick Sort in Data structure

(i) Let [Key] element

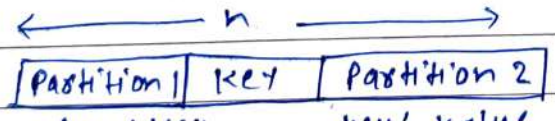
(ii) Divide array in two part ~~mid~~ Key be in mid left side less Right side greater than mid.



(iii) Same Partion in Partion 1



(iv) Same Partion in Partition 2



(v) Output will generate with sort

P.T.O

Ex)

i	J								
42	84	75	20	60	10	90	50	05	30

↑ Key

max size = 10

First value = i next value = j

Compare key with J

1st condition if a[J] < key →

increment i = i++

then

a[i] & a[j] swap

then

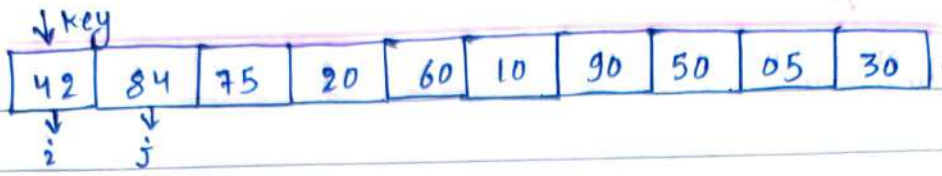
j = J++

Else

j = J++;

if J == maxsize - 1

a[i] & key value swap



mat size = 10

key = 42 a[j] = 84

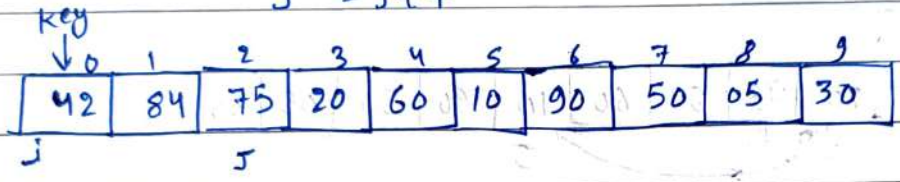
condition ①

a[j] < key

84 < 42 false

else part

j = j++



condition ①

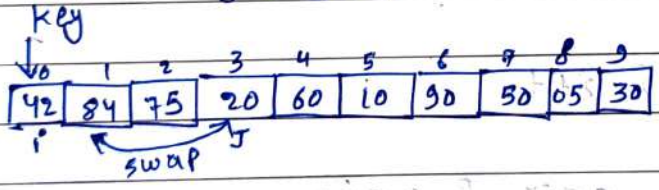
a[j] < key

75 < 42

False

else part

J = J++



condition ①

a[j] < key

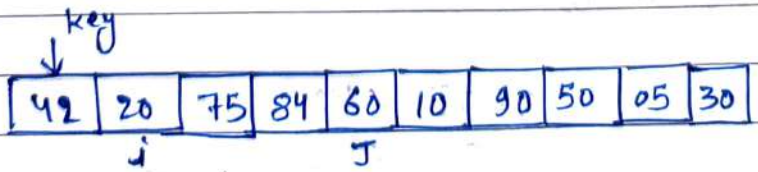
20 < 42 True

i = i++

a[i] & a[J] Swap value

J = J++ ;

Then



P.T.O

Condition ①

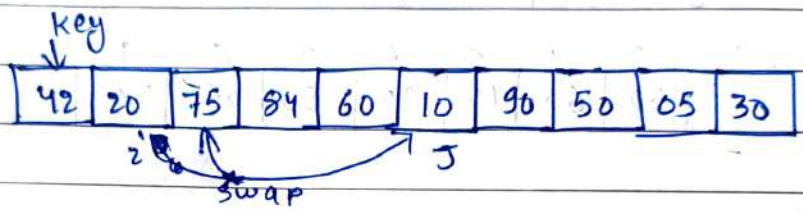
a[J] < key

False

60 < 42 false

else part

J = J++



condition ①

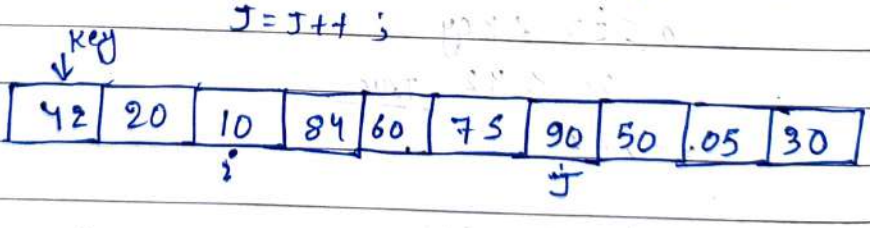
a[J] < key

10 < 42 True

i = i++

a[i] & a[j] swap

J = J++ ;



~~Now~~ Now

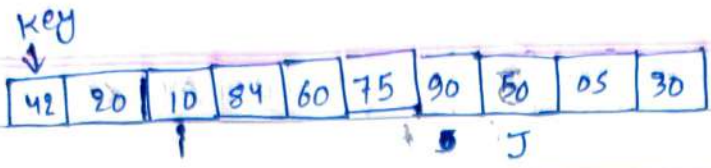
condition ①

a[J] < key

90 < 42 False

else

J = J++



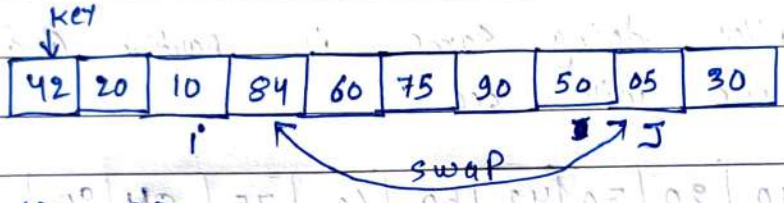
condition ①

a[J] < key

50 < 42 False

else part

J = J++



condition ①

a[J] < key

05 < 42

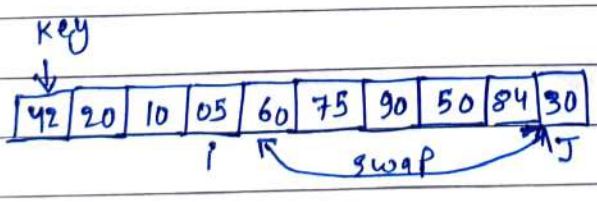
True

i = i++

a[i] & a[j] swap

J = J++;

Now



condition ①

a[J] < key

30 < 42

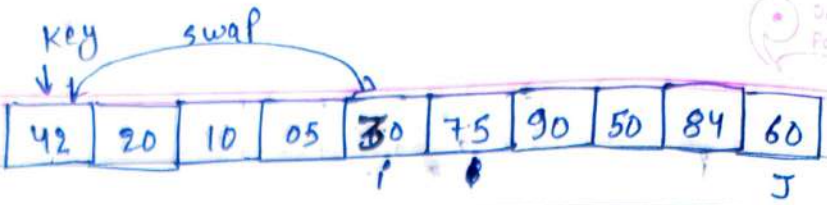
True

i = i++

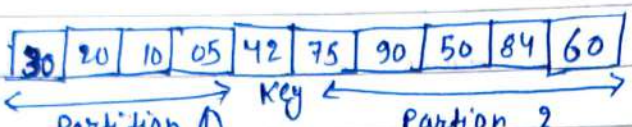
a[i] & a[J] swap

J

P.T.O

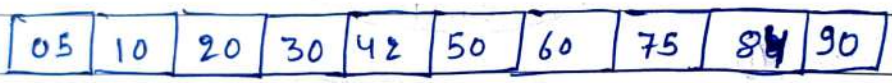


if J = mat size - 1
a[i] & key swap



value < key value > key

after doing same in partion (1) & (11) we will see



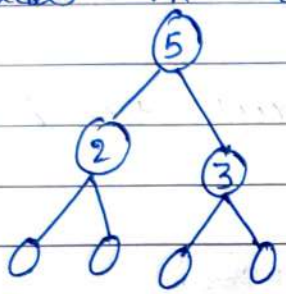
Heap Sort in Data Structure

(i) **Max heap**

Parent / Father

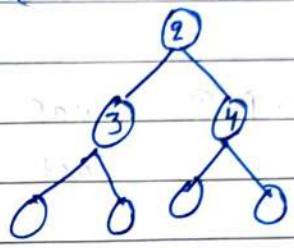
father ≥ child

deals in complete binary tree



min heap

father ≤ child



Max heap

(Ex)

10	15	5	8	25
a[0]	a[1]	a[2]	a[3]	a[4]

Max heap

step (i) Create heap

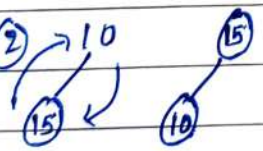
step (ii) Delete heap

step (iii) if value less than swap until satisfy.

step(i) :-

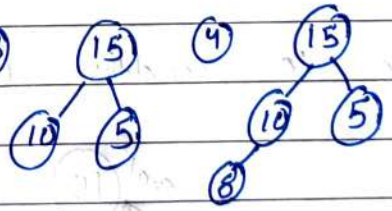
(1) 10

(2) 10

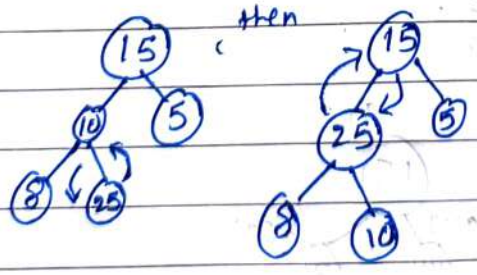


(3) 15

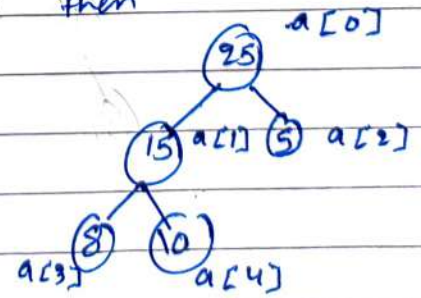
(4) 15



(5) 25



then



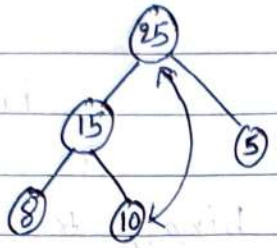
Max heap satisfy

P.T.O.

Step 2:

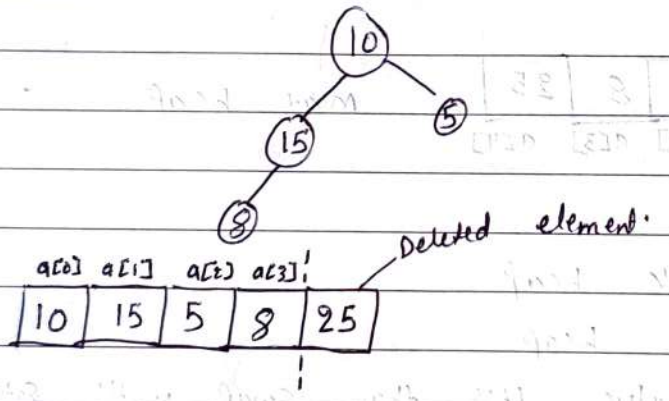
25	15	5	8	10

for deleting element in heap

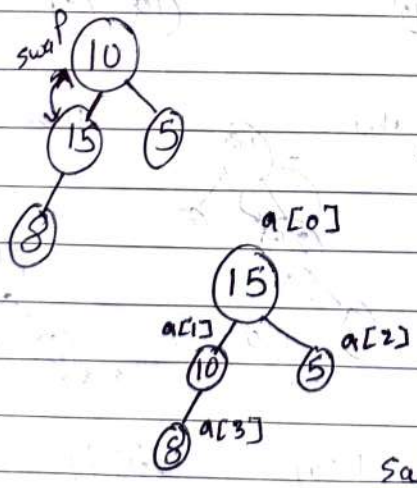


step ① Swap last element with root than delete last element

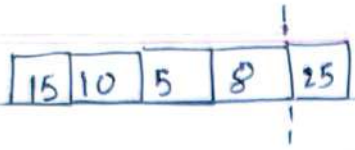
Then



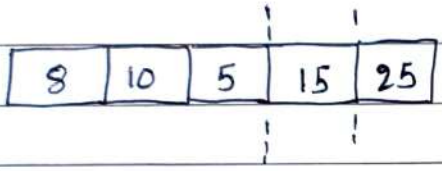
go to condition ①



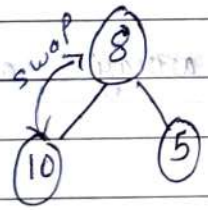
Satisfy

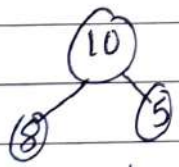


step ② delete 8 after swapping

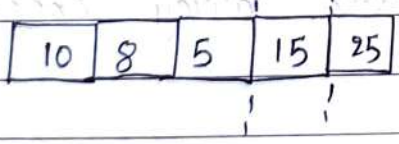


go to condition ①

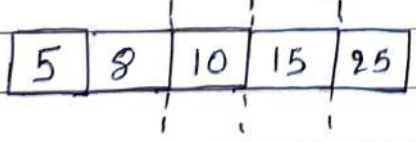




satisfy

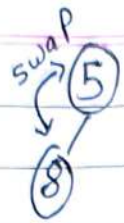


step ② delete 5

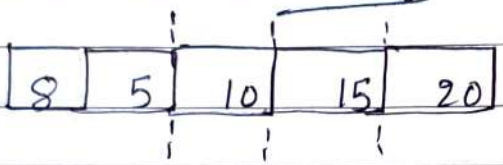


P.T.O.

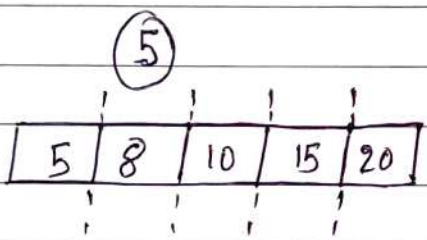
Step ①



satisfy

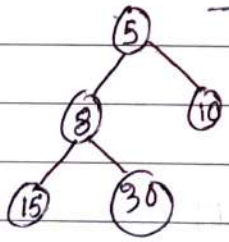


Step ② delete 5



Sorted array order

Proved



Sorted array created