Introduction to DS

A Data structure is a way to store & organize data in a computer, so that it can be used efficiently.

Random data

① difficult in searching
eg. words in random order.

② college list students → dept wise / class wise etc.

Structured data

eg. in dictionary (arrange words)
⇒ searching / insertion is easier.

operation

- traversing
- searching
- sorting
- insertion
- deletion

(all becomes easy for a structured data).

Data structure is used by →

-) OS
-) compiler
-) DBMS
-) Data communication etc. (related to data).

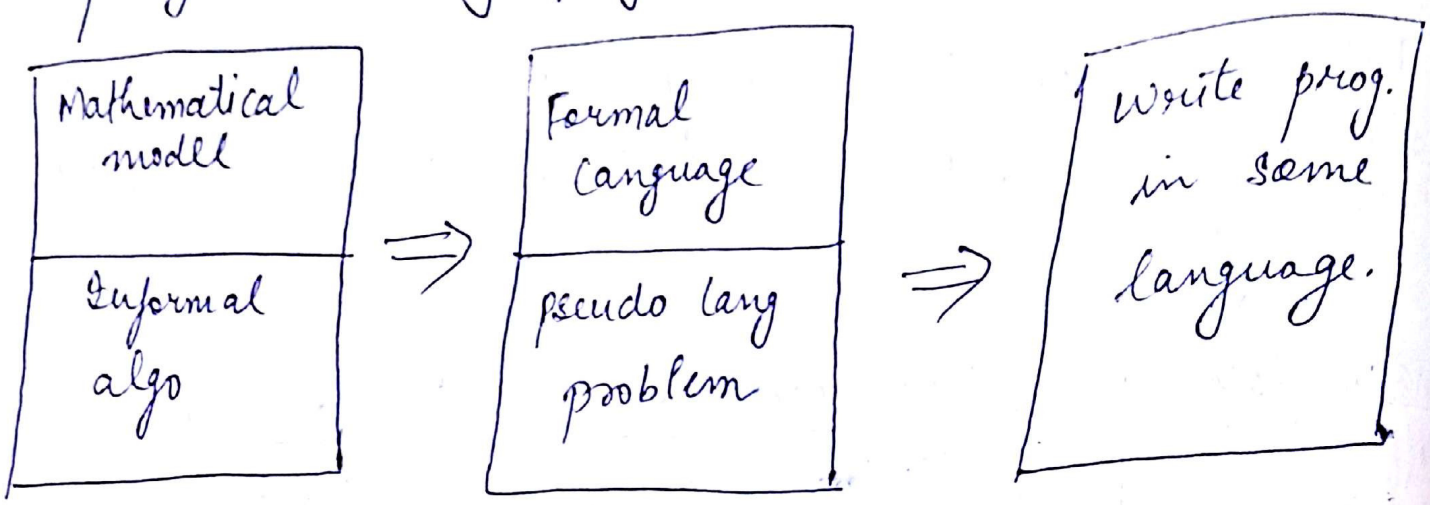
Programming methodologies & design of algo

It deals with different methods of designing program.

* **Program:** An implementation of an algorithm in some programming language.
* **Algo:** It is a step-by-step finite sequence of instr, to solve well defined computational problem.
* **DS:** way of organizing data..

DS = organized data + opr

Steps for creating prog:—



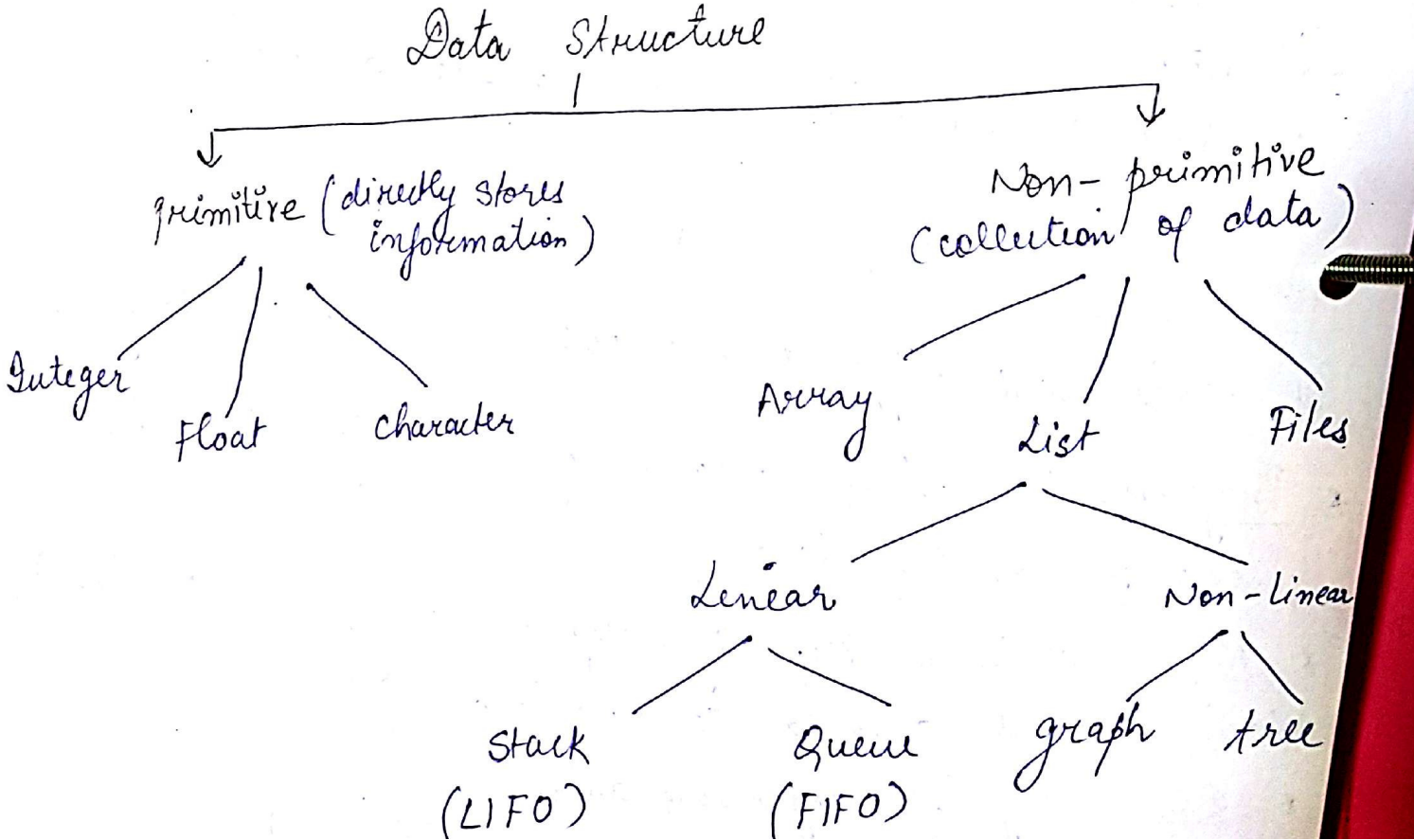
Ques. Identify those students who passed the exam (50% & above).

Soln:

```
m = 60
if (m >= 50)
"Student pass")
```

step1: Enter marks
step2: if marks >= 50
print "pass"
step3: Repeat 1 & 2
step4: End

⇒ write in c lang



Abstract Data type: → we are not concerned about the internal interface.

we directly use them as per our need.

Stack / Queues are ADT

eg. driving a car

ADT is a logical description of how we view the data & opr that are allowed without knowing their implementation."

Array: It is a collection of similar data type. (int / char / float)

eg. int x_1, x_2, x_3;

but in array => int arrayname [size]

eg. int a[5]; // declaration of array

int a[5] = {4,6,8,5,3} // initialization of array

0 4 // indexing in C

No. of elements = UB − LB + 1 // lower & upper bounds

= 4 − 0 + 1 = 5.

eg. LB = −2, UB = 3, find no. of elements

N = 3 − (−2) + 1 = 6

1-D Array

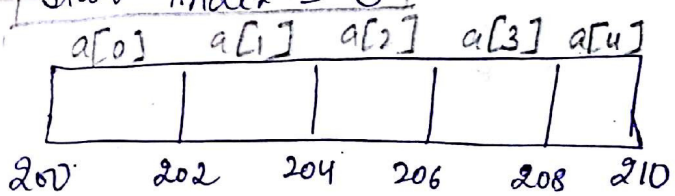
- linear array : list of finite number n of homogeneous data elements.

int A[LB] --- A[UB]

address calcul^n :

int A[5]

Start index = 0



(base address)

A[i] = Base address + i (data type size)

A[2] = 200 + 2(2)

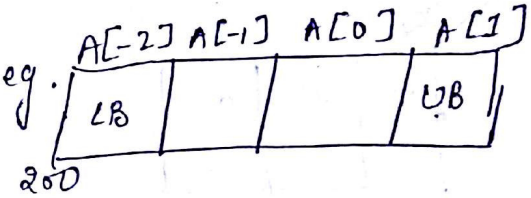
= 204.

Start index = LB.

now,

A[K] = Base add. A[LB] + (K − LB) × size

eg.



A[0] = 200 + (0 − (−2)) × 2

= 200 + 4

= 204

Q. A[-4:3]

$N = 3 - (-4) + 1 = 8$

For 2D arrays,

eg. A[-2:2, 2:22]

Soln, Find L_1 and L_2

Total elements = $L_1 \times L_2$

$\therefore L_1 = 2 - (-2) + 1 = 5$

$L_2 = 22 - 2 + 1 = 21$

$N = L_1 \times L_2 = 5 \times 21 = \underline{105}$.

Ques. A[2:8, -4:1, 6:10], find elements

$L_1 = 8 - 2 + 1 = 7$

$L_2 = 1 - (-4) + 1 = 6$

$L_3 = 10 - 6 + 1 = 5$

$\therefore N = 7 \times 6 \times 5$

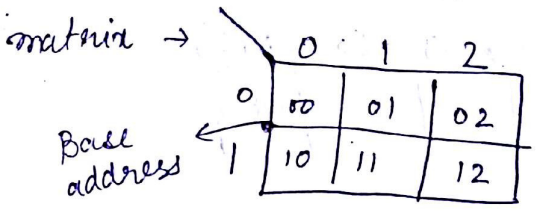
$= 210$

Array Representation

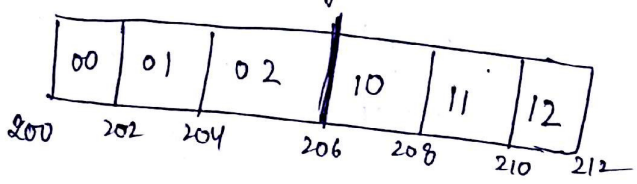
2-D array representation, also known as "matrix"

① Row-Major

eg. int A[2][3] // elements = $2 \times 3 = 6$



For row major, first rows are traversed.



eg. A[m][n]

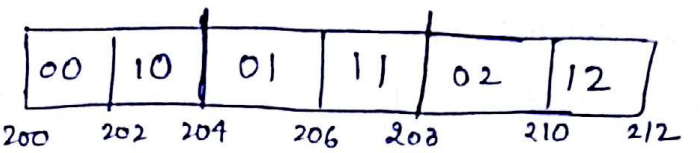
address:

$A[i][j] = Base + (i \times n + j) \times size$

eg. $A[1][0] = 200 + (1 \times 3 + 0)2$

$= 200 + 6 = 206$

(2) column major (4)



$A[i][j] = Base + (j \times m + i) \times size$

$A[0][1] = 200 + (1 \times 2 + 0) \times 2$

$= 200 + 4 = 204.$

Sparse Matrix (2-D array) : majority of elements are Zero. very few non-Zero elements.

→ it reduces scanning time.

eg. A[100][100] : elements = 10,000

(we need to scan all 10000 elements to find where values exist. Instead, we use Sparse Matrix).

Representation < Three column form

Linked List

(1) 3 columns = rows, col, values

eg.

	0	1	2	3
0	0	2	0	0
1	4	0	3	0
2	0	1	0	1
3	0	2	0	0

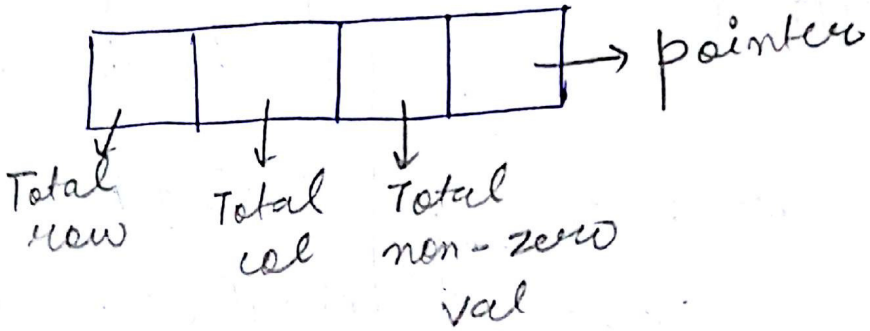
4×4

1st row: Total r, c & values →

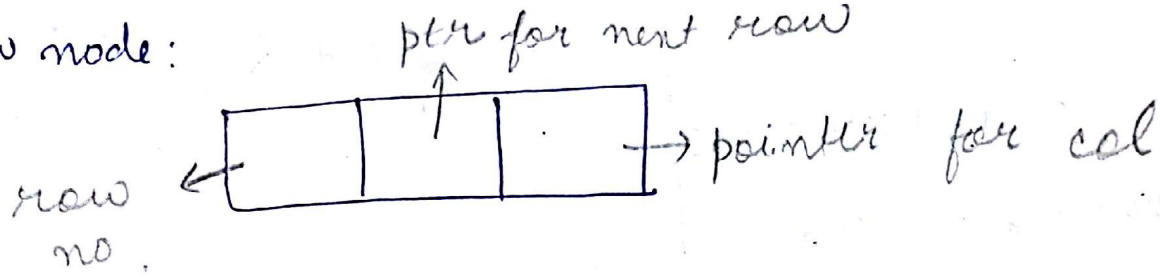
Rows	col	value
4	4	6
0	1	2
1	0	4
1	2	3
2	1	1
2	3	1
3	1	2

(2) In Link List representation, we require three structure.

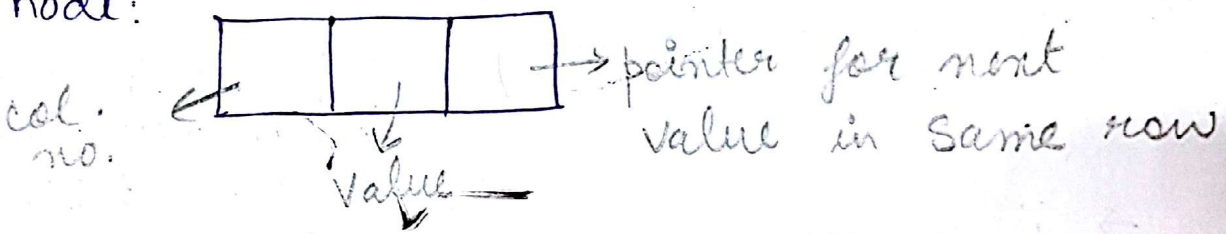
a) Head node :



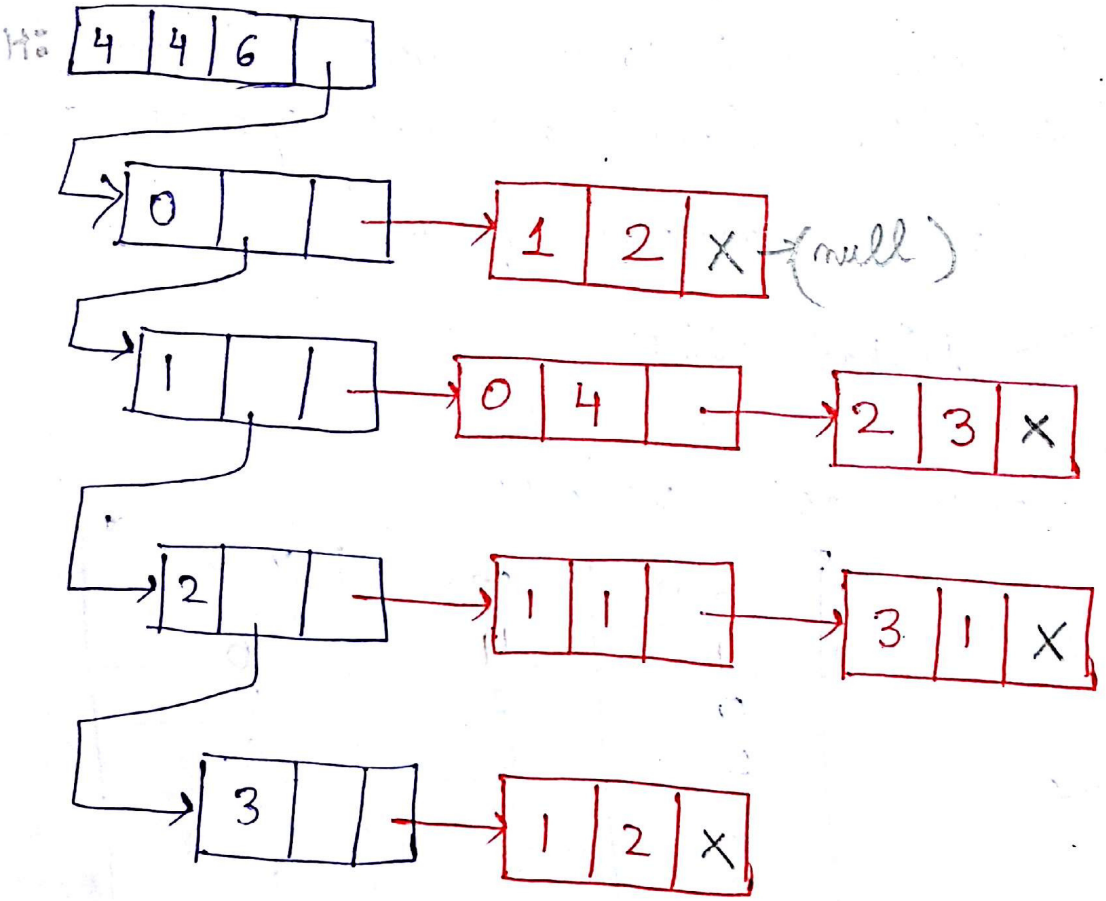
b) Row node :



c) column node :

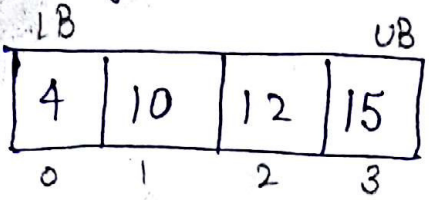


eg : same →



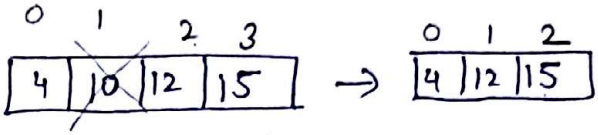
ray operations:

Traversing ▸ Deleting ▸ Insertion



Step 1: Repeat for i = LB to UB
2: Process A[i]
[End of loop]
3: Exit

(2)



Step 1: Set Item := A[K] // to be deleted
2: Repeat for i = K to N-1
Set A[i] := A[i+1]
[end loop]
3: Set N := N-1
4: Exit

(3) Insertion

•) Insertion in unsorted array

Let K be the loc^n where item is to be inserted.

K = 2

i = 2

Step1: Set i = N
2: Repeat while i ≥ K
3: Set A[i+1] := A[i] // moving the element → Right
4: Set i := i - 1
[end loop]
5: Set A[K] := item.
6: N := N+1;
7: Exit

Date

Array-

Linear Array- Int A[K]

Array Length = UB - LB + 1
(Lower bond, Upper bond)

LOC(A[K]) = Base address + w(K-LB)

Location of Kth element of array A whose address is to be calculated

Ques) Base = 200, LB = 1932, Sale[K] = 1965, w = 4 words/memory unit

LOC(Sale[1965]) = 200 + 4{1965 - 1932}

= 200 + (4 x 33) = 332

→ Operations on Data Structures -

1) TRAVERSING-

Step 1 Set K := LB [Initialize counter]
Step 2 Repeat steps 3 & 4 for K ≤ UB
Step 3 Apply process to LA[K] [Visit element]
Step 4 K := K + 1 [Increase counter]
[End of Step 2 loop]
Step 5 Exit

2) INSERTION-

INSERT (LA, N, K, ITEM)-
(N: length of array, K: location)

Step 1 Set J := N [Initialize counter]
Step 2 Repeat steps 3 & 4 while J ≥ K

Step 3 Set LA[J+1] := LA[J] → # This steps moves elements to next position [Move] to create vacant spot for new element
Step 4 Set J := J-1 [Decrease counter]
[End of step 2 loop]
Step 5 Set LA[K] := ITEM
Step 6 Set N := N+1
Step 7 Exit

3) DELETION-
DELETE(LA, N, K, ITEM) → position to be deleted
Step 1 Set ITEM := LA[K] , counter # Element to be deleted is stored as ITEM only to save it at an alternate location. This allows us to print the value of deleted (at the end of program) if required
Step 2 Repeat for J = K to N-1
Set LA[J] := LA[J+1]
[End of loop]
Step 3 Set N := N-1
Step 4 Exit

4) A) LINEAR SEARCH-
LINEAR [DATA, N, ITEM, LOC]

Step 1 Set DATA[N+1] = ITEM # Only to check at end if a match is found
Step 2 Set LOC = 1 [Initialize counter]
Step 3 Repeat while DATA[LOC] ≠ ITEM
Set LOC := LOC+1
[End of loop]
Step 4 If LOC = N+1, then set LOC := 0 [Unsuccessful]
Step 5 Exit

Worst case complexity: $f(n) = n+1$

Date / / /

4.B) BINARY SEARCH-

Location { BEG = LB
END = UB
MID = INT $\left(\frac{BEG+END}{2}\right)$

BINARY (DATA, LB, UB, ITEM, LOC)

Step 1 Set BEG := LB, END := UB, MID := $\left(\frac{BEG+END}{2}\right)$

Step 2 Repeat steps 3 & 4 while BEG ≤ END and DATA[MID] ≠ ITEM

Step 3 If ITEM < DATA[MID] then:
Set END := MID - 1
else:
Set BEG := MID + 1
[End of If structure]

Step 4 Set MID := { INT $\left(\left(\frac{BEG+END}{2}\right)\right)$

[End of step 2 loop]

Step 5 If DATA[MID] = ITEM then:
Set LOC := MID
else
Set LOC := NULL

Step 6 Exit

∴ Worst case complexity : $f(n) = \lceil \log_2 n \rceil + 1$

→ Algorithm Notations -
Finite step-by-step instructions

1) Identifying Number - To identify algorithms & distinguish them from other algorithms
eg. Algo 4.1 Insertion of elements
no. description (brief)

2) Steps, control & Exit - Numbering the steps
eg. numbering of steps
Step 1 ***
Step 2 go to step 7 → control
Step 3 ***
⋮
Step n Exit / End / Stop
when loops are to be used

3) Comments - To simply or explain any particular step
always enclosed in square brackets

4) Assignment statement - To assign values to variables
# a := 5 ⇒ assigns value 5 to a
a = b ⇒ a equals b

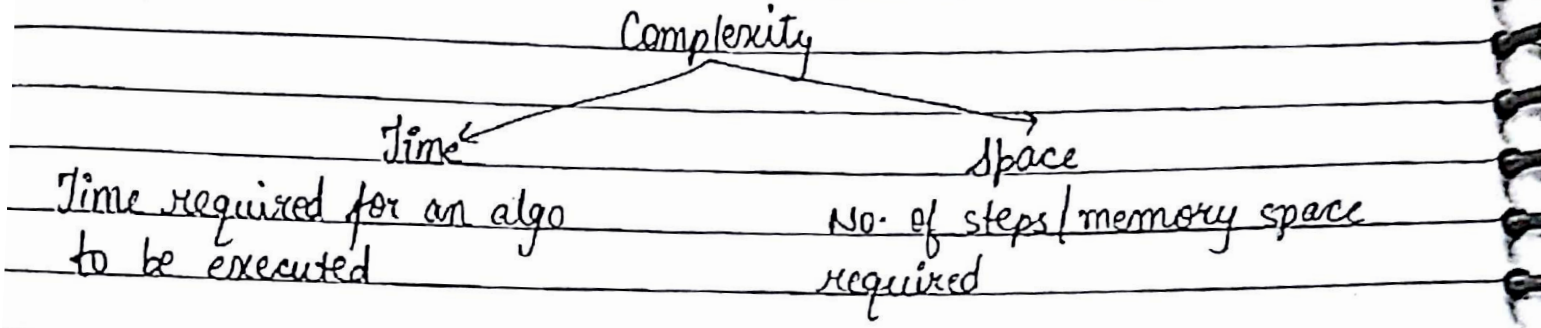
5) Variable Names - Always written in CAPITAL LETTERS

6) Input and Output - Read: (Input)
Write: (Output)

Complexity – The choice of an algorithm depends on its complexity

↓

Measure of effort required to execute an algorithm



Time required for an algo to be executed

No. of steps / memory space required

→ Time-Space Trade Off –

- Depends on time & space complexity
- It is not possible for both these complexities to be less for any particular algorithm.

→ Complexity –

Suppose 'M' is an algorithm and 'n' is the size of the input data. The complexity of algo M is fn f(n) which gives the running time and storage space requirement of algo M in terms of 'n' that is the size of input data.

→ Big O Notation –

let the complexity of an algo be f(n).

Big O Notation is the characterization scheme that allows us to measure the properties of algos such as performance and memory required.

In this notation, we eliminate the constant factors.

Therefore, the complexity fn increases as n increases.

eg. $f(n) = 2n^2 + 3n$
$O(n) = n^2$

eg. $f(n) = 3n^3 + n^2 + 2n$
$O(n) = n^3$

eg. $f(n) = 3n^3 + 1n^2 + \log n$
$O(n) = \log n$

eg. $f(n) = 4n^3 + \log n + 3e^n$
$O(n) = e^n$

eg. $f(n) = 2$
$O(n) = 1$

Three cases of algo complexity -

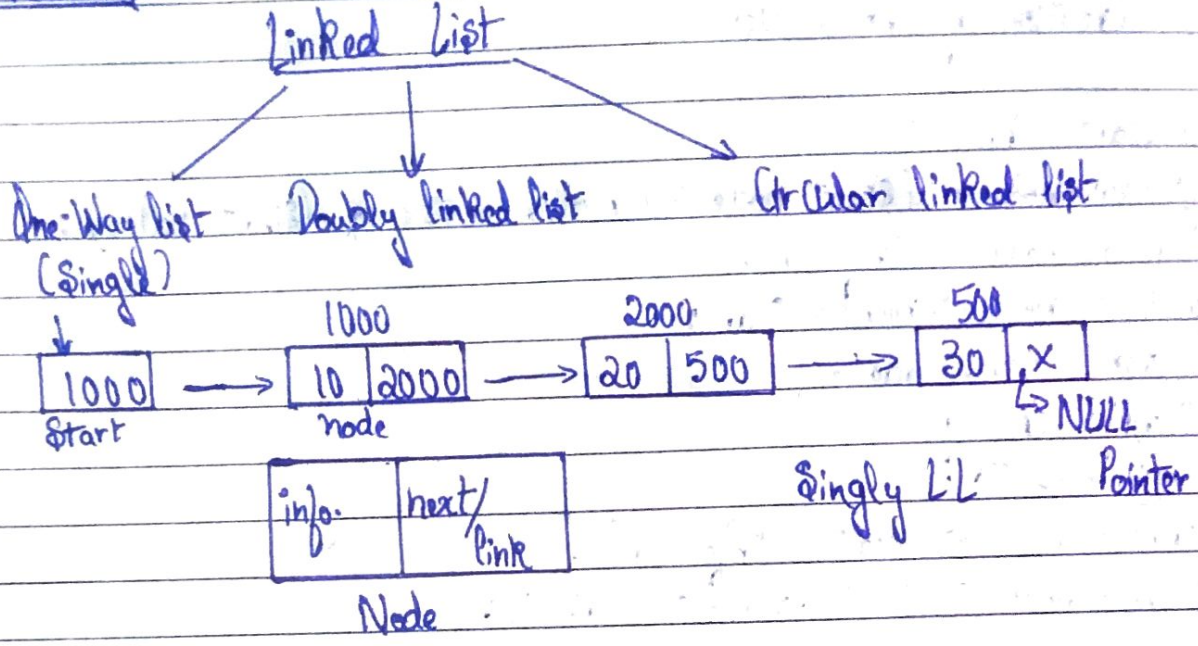
1) Worst case: Max. value of $f(n)$ for every input
ie. algo runs correctly only on a few input sets

2) Average case: Expected value of $f(n)$

3) Best case: Min. value of $f(n)$

Date 14/9.....

Data Structures



Linked list is a data structures in which insertions & deletion can be performed at any time.

* NULL Pointer indicates end of linked list

Structure of node ;; Syntax

```
struct node
{
    int info.;
    struct node * next;
} * start, * node, * new node;
```

Syntax of malloc

```
ptr = (cast type *) malloc (size of (byte - size));
```

Saral

Date .....

Creation of Linked List

(i) start = NULL;
(ii) new_node = (struct node *) malloc (sizeof (struct node));
(iii) new_node → info = m;
(iv) new_node → next = start;
(v) start = new_node;
(vi) while ch = 'y'
(viii) → (ii)
(viii) new_node → next → info = m1;
(ix) new_node → next → next = NULL;

Array	Linked List
(1) Arrays are stored in contiguous memory locations.	Elements are not stored in contiguous memory locations; can be stored anywhere in memory.
(2) It has static memory allocation i.e memory allocation happens at compile time.	It supports dynamic memory allocation. ie. memory allocation happens at run time
(3) Size of the array has to be specified at the time of declaring the array.	There is no need for specifying the size of linked list.
(4) It is a collection of homogeneous (similar) data type.	It is a collection of node (data & address)
(5) It occupies less memory than linked list.	It occupies more memory as it stores both data & the address of next node.
(6) It has a fixed size	It doesn't has a fixed size
(7) Elements can be accessed easily.	Elements accessing requires the traversal of whole linked list.

Spiral

Date .....

(8) Insertion & deletion is slower.	It is faster than Array to do insertion & deletion operation
(9) Array can't be used as a stack or Queue	Once the L.L is defined, can be used as array, stack, queue, SLL & DLL
(10) Elements of an array are not dependent on each other.	Elements are dependent on each other as each node stores the address of its next node

Multi-D Array

↳ Array of Array

Row by Method →

$$Base(A) + W\left[\{((E_1 L_2 + E_2) L_3 + E_3) L_4 \text{ ------- } + E_{N-1}\} L_N + E_N\right]$$

Column Method →

$$Base(A) + W\left[\{(((E_N L_{N-1} + E_{N-1}) L_{N-2}) + \text{-----} E_3) L_2 + E_2\} L_1 + E_1\right]$$

here L_i is the number of indices calculated as

L_i = (Upper bound − lower bound + 1)

E_i → effective index, $E_i = \underbrace{K_i}_{\text{subscript}} - \text{lower bound}$

Q Suppose a[5,-1,8] is 3D array. Base(a) = 200, w = 4 ; a(2:8, -4:1, 6:10)

Sol^n → $l_1 = 8-2+1 = 7$

$l_2 = 1+4+1 = 6$

$l_3 = 10-6+1 = 5$

Total elements

$l_1 \cdot l_2 \cdot l_3 = 210$

Date .....

Now; $E_1 = 5-2 = 3$, $E_2 = -1+4 = 3$, $E_3 = 8-6 = 2$

Hence LOC $a[5,-1,8]$ = Base(a) + W[$(E_1 l_2 + E_2) l_3 + E_3$]
= 200 + 4[$(3 \cdot 6 + 3) 5 + 2$]
= 200 + 4 (107)
= 200 + 4
=

LinRed List

Traversal of Linked List

① NODE = START
② While NODE ≠ NULL Repeat 3 & 4
③ Process node [info]
④ Set Node = Node [Next];
End of Step 2 loop
⑤ Exit

Count the no. of elements in linked list

① Count = 0;
② NODE = START
③ While NODE ≠ NULL (Repeat 4 & 5)
Count ++;
④ set Node = Node [Next];
end of step 2 loop
⑤ No of elements = Count
⑥ Exit

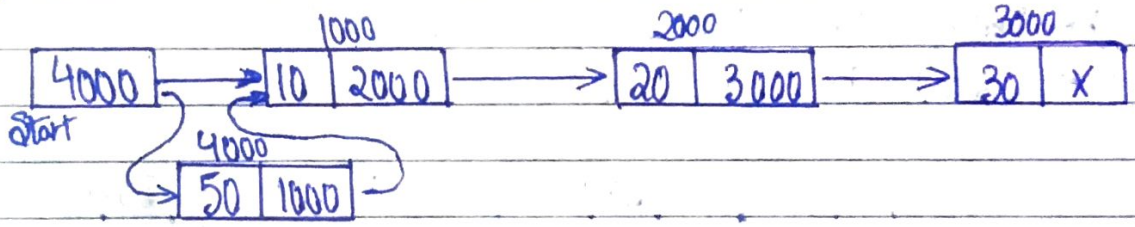
Date .....

Insertion in Linked list ⟶ 3 types

- Beginning
- Middle
- End

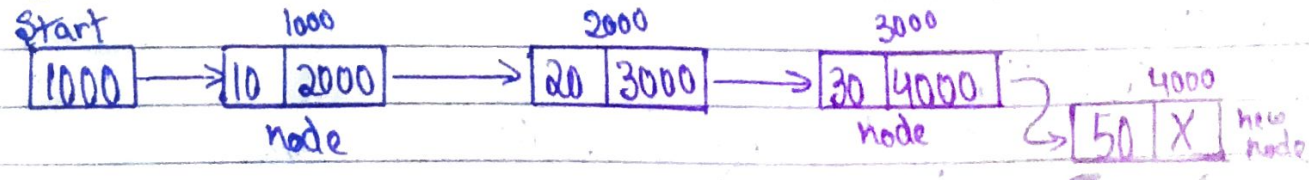
⇒ Insertion in Beginning

1. Create a new node say newNode.
2. Set info [newNode] = item
3. Next [newNode] = start
4. Start = newNode
5. Exit



⇒ Insertion at End

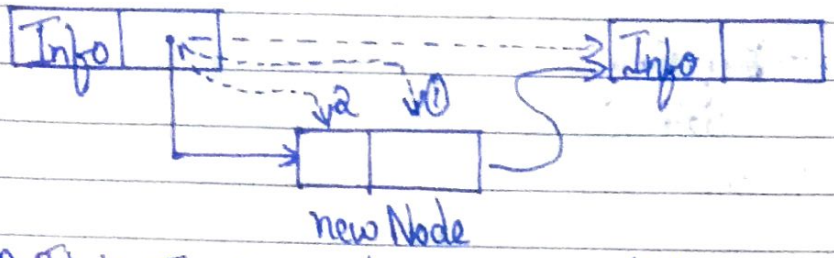
1. Node = start
2. Create a new node say newNode
3. Set info [newNode] = item
4. Repeat while (node [next] ≠ NULL)
   1. node = node [next]
5. Next [newNode] = next [node]
6. next [node] = next [node]
7. Exit



Spiral

Date .....

Insertion at specified location



① Input Insert_node // Position 5000 ← Position = 3

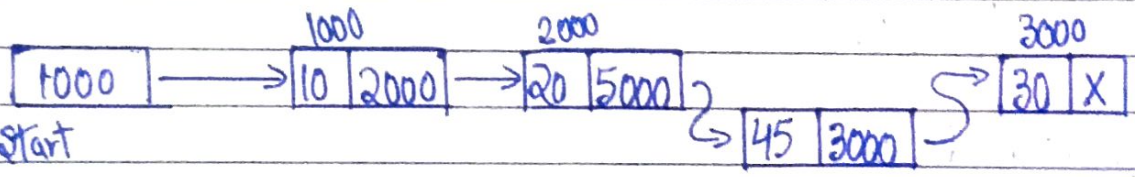
② Create a new node say new Node & 45 | | (new node)
info [new Node] = item

③ Node = start and count = 1

④ Repeat while count < Insert_node - 1
  (i) Node = Next [node]
  (ii) count ++

⑤ Next [new Node] = node [next]
  next [node] = new Node

⑥ Exit



⇒ Deletion from Linked list

- first node
- last node
- & specified location

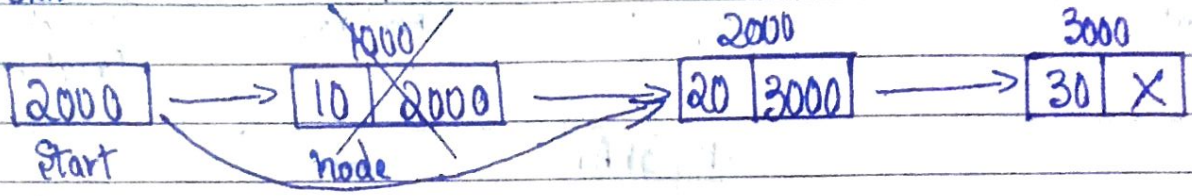
* Deletion of first node

① NODE = start

② If NODE = NULL
  output "underflow" & exit

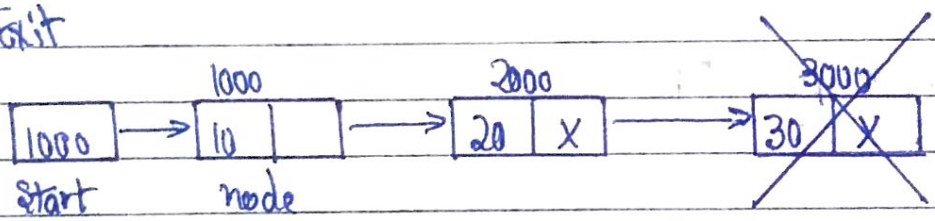
Date .....

else
    Start = next [node]
    free the space associated with node
⑤ Exit



* Deletion of last node

① NODE = Start
② If NODE = NULL output "underflow" & exit
③ Repeat while next [next [node]] ≠ NULL
        NODE = next [node]
④ Next [node] = NULL
⑤ Exit



[Note : Draw Back of SLL : Backward Traversal is not possible]

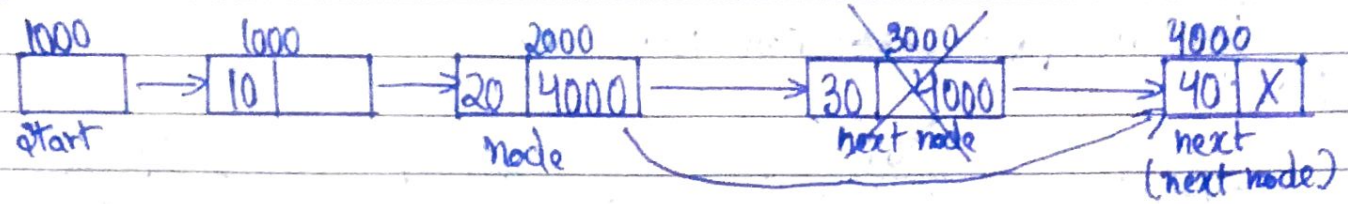
* Deletion of a specified node

① NODE = Start, count = 1
② Input the position, delete _ node
③ If NODE = NULL output "Underflow" & Exit
④ Repeat while (count < delete_node - 1)
        (i) NODE = next [node]
        (ii) count = count + 1

Date .....

⑤ Next [node] = next [next [node]]

⑥ Exit



PROGRAM

* Program for single linked list

```
#include <stdio.h>
#include <malloc.h>
  struct node
        {  int info;
           struct node * next;
        } * start
  int main ()
  {  int ch, n, m, pos, i;
     start = NULL;
     while (1)
     {
       printf ("1. Create list \n");
       printf ("2. Add at Beginning \n");
       printf ("3. Display \n");
       printf ("4. Count \n");
       printf ("5. Reverse \n");
       printf ("6. Quit ");
      scanf ("%d", &ch);
       switch (ch)
      {
```

Date .....

```
case (1):
    printf ("Enter no of nodes");
    scanf ("%d", &n);
    for (i=0; i<n; i++)
{
    printf ("Enter element :");
    scanf ("%d", &m);
    create_list (m);
}
    break;
case 2:
    printf ("Enter element");
    scanf ("%d", &m);
    add at beg (m);
    break;
case 3:
    display ();
    break;
case 4:
    count ();
    break;
case 5:
    rev ();
    break;
case 6:
    exit ();
default:
    printf ("Wrong Choice \n");
}   // End of switch //
}   //End of while //
}   // End of main () //
```

Spiral

Date .....

```
create_list (int data)
{ struct node * node, * new_node;
  new_node = (struct node *) malloc (size of (structure));
  new_node -> info = data;
  new_node -> next = NULL;
  if (start == NULL)
     start = new_node;
  else
     { node = start;
       while (node -> next != NULL)
          node = node -> next;
          node -> next = new_node;
       }
  }
add to beg (int data)
     { struct node * new_node;
       new_node = malloc (size of (struct node));
       new_node -> info = data;
       new_node -> next = start;
        start = new_node;
     }
  display ()
{
    struct node * node;
     if (start == NULL)
       {
        printf ("list is Empty \n");
        return;
       }
       node = start;
```

Date .....

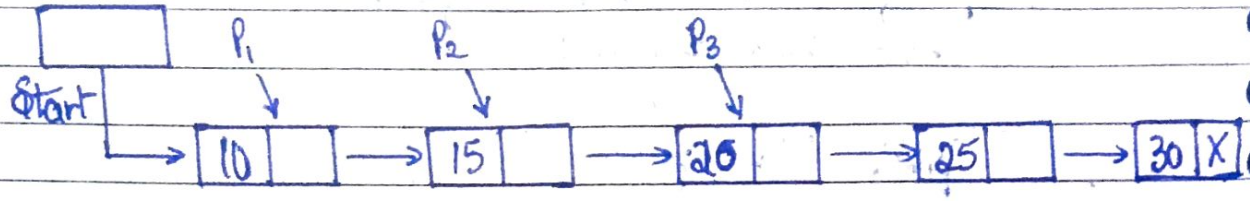
```
    while (node != NULL)
  { printf ("%d", node -> info);
    node = node -> next;
  }
    printf ("\n");
}
  count ()
{  struct node * node = start
   int cnt = 0;
   while (node != NULL)
   { node = node -> next;
     cnt ++;
   }
   printf ("No of elements are %d \n", cnt);
}  // End of count //
  rev()
{
  struct node *p1, *p2, *p3;
    if (start == NULL)
      return;
   p1 = start;
   p2 = p1 -> next;
   p3 = p2 -> next;
   p1 -> next = NULL;
   p2 -> next = p1;
while p3 != NULL
  { p1 = p2;
    p2 = p3;    p3 = p3 -> next;    p2 -> next = p1;
  }
   start = p2;
}  // End of Rev() //
```

{ cnt = count }

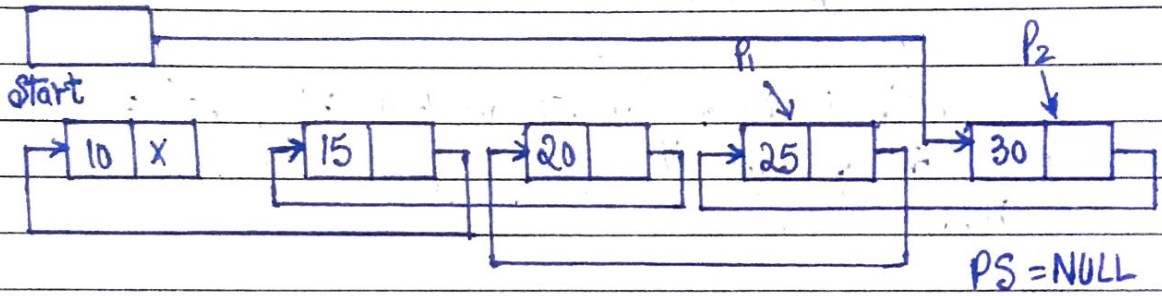
Spiral

Date 25/9.....

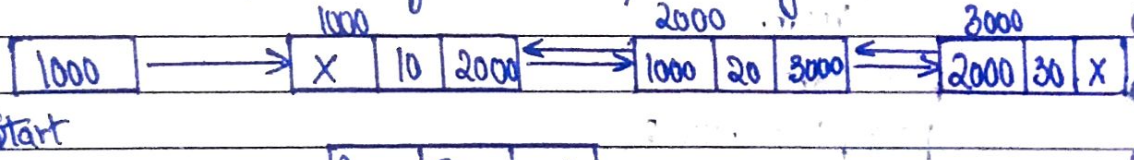
Reverse Function

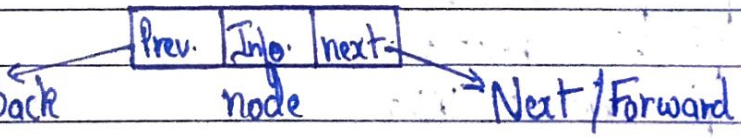


1. First node becomes last node
2. Last node becomes first node
3. Link of second node will point to first node
4. Link of third node will point to second & so on.
5. Link of last node will point to previous node of last node in linked list.



Doubly Linked list / Two way list





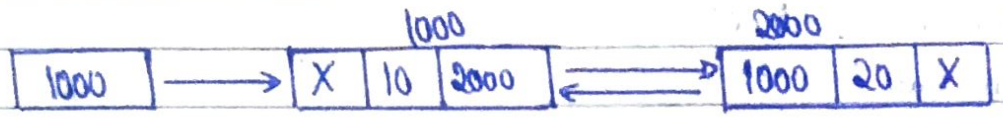
* Creation of Doubly linked list →

1. Start = NULL
2. Start = NODE, Previous [node] = NULL, next [node] = NULL

Spiral

Date .....

(3) Input the info in node, Info [node] = Item
(4) Repeat while choice is 'yes'
(i) Create a next node i.e next [node]
(ii) Previous [next [node]] = node
(iii) Node = Next [node]
(iv) Info [node] = Item; next [node] = NULL
(5) Exit.



Traversal → Forward
Backward

Forward Direction

(1) Node = Start
(2) Repeat while Node ≠ NULL
(i) Process the info // Print info
(ii) Node = next [node]
(3) Exit.

Backward Direction

(1) Node = Start
(2) Repeat while next [node] ≠ NULL
Node = next [node]
(3) Repeat while Node ≠ NULL
(i) Process the info.
(ii) Node = Previous [node]
(4) Exit

Date .....

Insertion in D.L.L.

* Insertion at Beginning

1. Create a node, new_node
2. node = start, next [new_node] = start
3. Previous [node] = new_node;
   Previous [new_node] = NULL
4. start = new_node
5. Exit

* Insertion at End

1. Create a new_node,
2. node = start
3. Repeat while next [node] ≠ NULL
   (i) node = next [node]
4. (i) next [node] = New_node,
   prev [new_node] = node
   (ii) next [new_node] = NULL
5. Exit

* Insertion at specified location

1. Create a newnode, say new_node
2. node = start, count = 1, Input Insert_node
3. Repeat while count < Insert_node - 1
   (i) node = next [node]
   (ii) count = count++

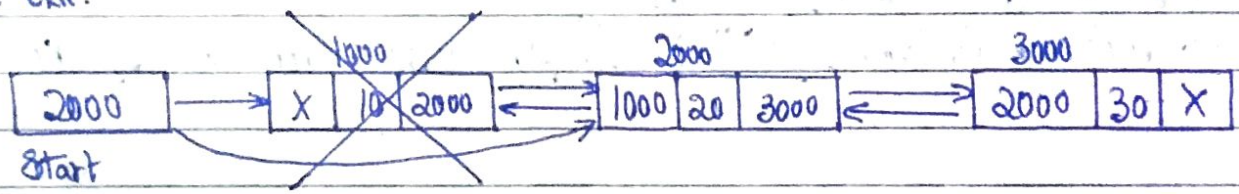
Date .....

4. (i) next [new_node] = next [node]
   (ii) previous [next [node]] = new_node
   (iii) next [node] = new_node
   (iv) previous [new_node] = node
5. Exit

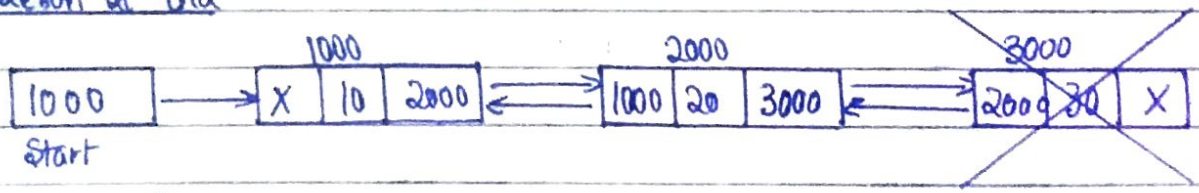
Deletion of D.L.L

* Deletion of first node

1. node = start
2. If NODE = NULL
   output "Underflow" & exit
3. else
   start = node [next]
   previous [next [node]] = NULL
   Free the space associated with node
4. Exit.



* Deletion at End



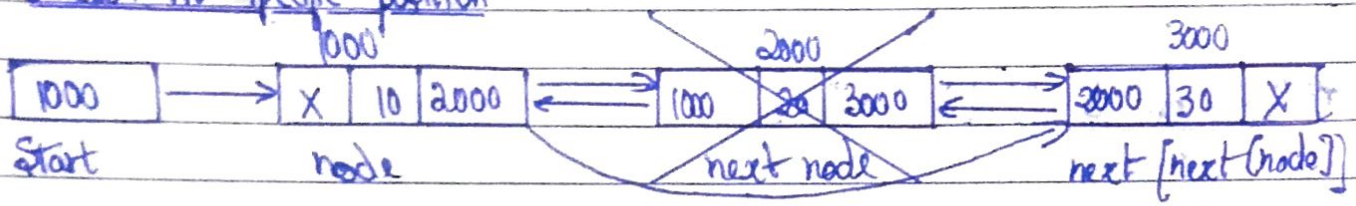
(1) node = start
(2) If node = Null output "Underflow" & exit
(3) Repeat while next [next [node]] ≠ NULL
    node = next [node]
    // Here this is ~~transfer~~ traversal

Spiral

Date .....

④ next [node] = NULL
previous [next [node]] = NULL.
⑤ Exit

* Deletion At Specific position



① node = start, count = 1
② Input the position, delete_node
③ If node = NULL, then output "Underflow" & exit
④ Repeat while [count < delete_node - 1]
  (i) node = next [node]
  (ii) count = count + 1
⑤ (i) next [node] = next [next [node]].
  (ii) pervious [next [node]] = NULL
  (iii) next [next [node]] = NULL (iv) pervious [next [next [node]]] = node
⑥ Exit

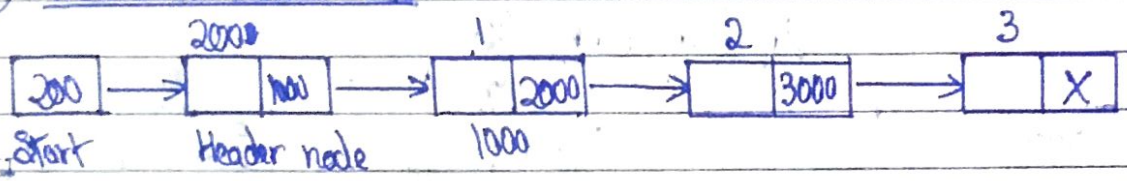
Date 13/10.....

Header LinkedList

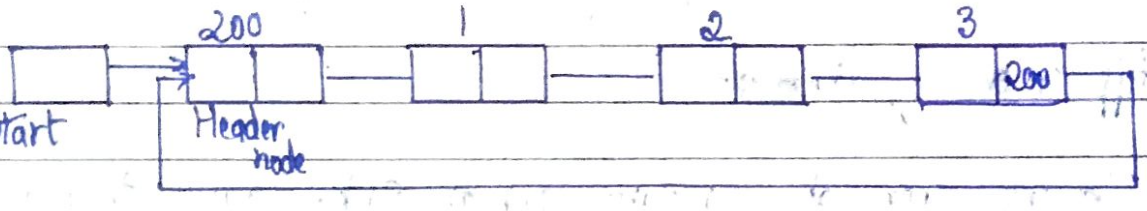
Header LinkedList is the one which contains a special node called 'Header node', it contains any type of special info in a linked list

Header LinkedList is of 2 types → Grounded, Circular

① Grounded Header List



② Circular Header List



last node will point to the header node

Creation & Traversal of Grounded header list

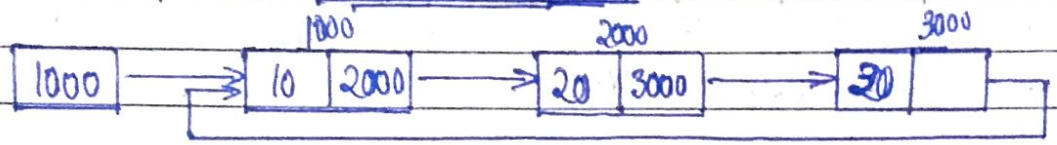
① Start = NULL
② Create the header node, say node
③ Start = node; i = 0
④ Info [node] = i, next [node] = NULL
⑤ Repeat while choice is 'yes'
  (i) Create a node [next [node]]
  (ii) node = next [node]
  (iii) info [node] = item
  (iv) next [node] = NULL, i = i+1
⑥ Node = start, info [node] = i

Date .....

Traversal → ① Node = next [start]
② Repeat while node ≠ NULL
(i) process info [node].
(ii) node = next [node]
③ Exit

Note → If it is Circular Header, then replace 'NULL' with 'start' rest algo is same.

Circular Linked List



Change "NULL" to 'start' of single LinkedList, for Circular LinkedList

Application of Linked list →

① To implement other data structures such as stacks, queue, trees, graphs.
② To perform arithmetic operations on non-integer.
③ To manipulate polynomials
④ Implement Sparse Matrix

POLYNOMIAL ARITHMETIC

Circular LinkedList is generally used in manipulation of Polynomials

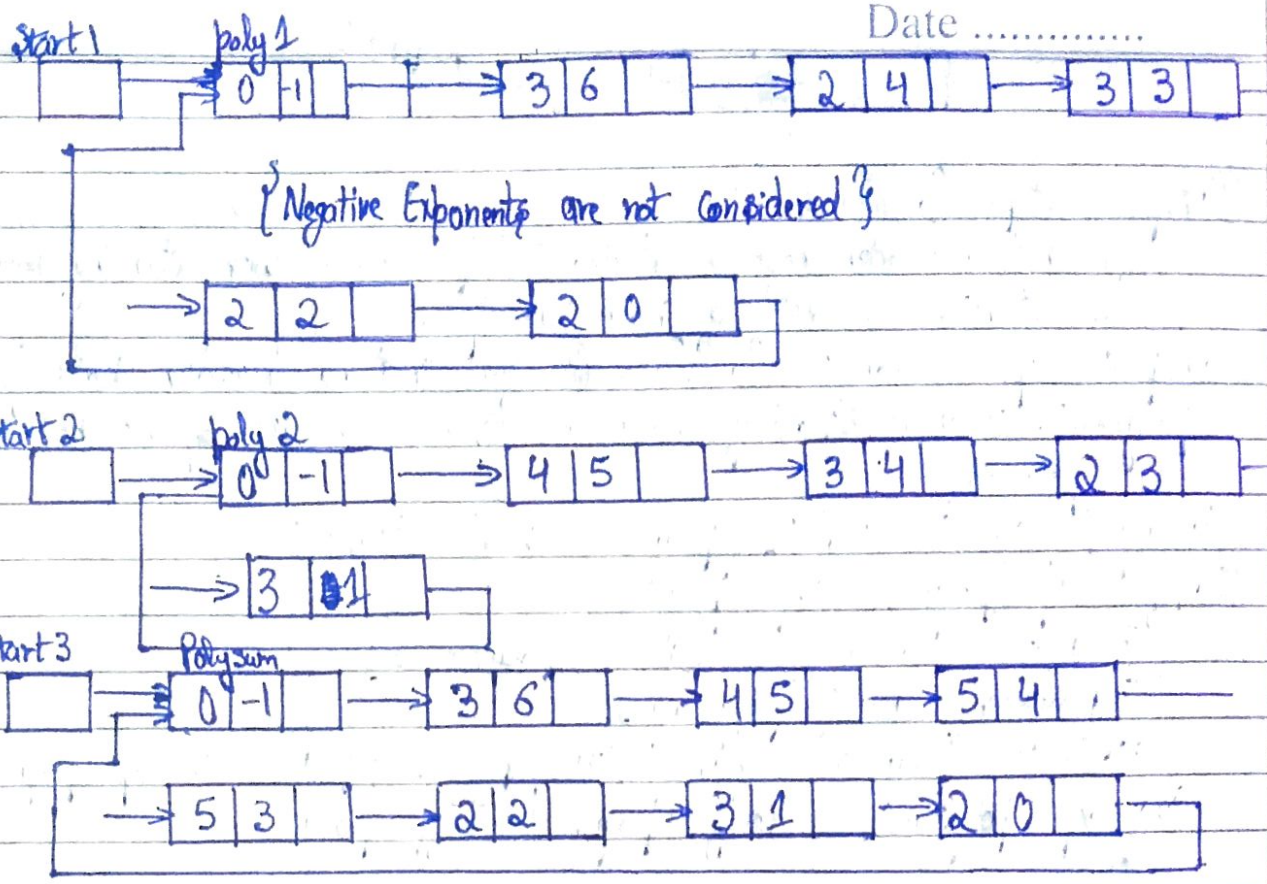
Eg → Poly 1 $f_1 = 3x^6 + 2x^4 + 3x^3 + 2x^2 + 2$

Poly 2 $f_2 = 4x^5 + 3x^4 + 2x^3 + 3x$

Poly Sum $f_3 = 3x^6 + 4x^5 + 5x^4 + 5x^3 + 2x^2 + 3x + 2$

Coeff.	Exp.	ptr.

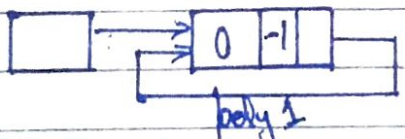
Date .....



Algorithum Steps →

① Create header node for polynomial, start 1 = NULL & Create a new node say, poly 1

② poly 1 = start 1, coeff [poly 1] = 0
exp [poly 1] = -1
next [poly 1] = start 1



③ Create nodes for polynomial
Repeat while choice is 'yes'

(i) Create a new node, [next [poly 1]], (ii) poly 1 = next [poly 1]

(iii) Enter Coeff [poly 1] and exp [poly 1]

(iv) next [poly 1] = start 1

Spiral

Date .....

(4.) Repeat steps ①, ② & ③ for creating header node & entering 2nd polynomial with starting node as start 2 & rest of node as poly 2.

(5.) Repeat steps similar to ①, ② & ③ for creating header node for polynomial 3. With starting node as start 3 & rest of the node as polysum.

(6.) Poly 1 = next [start 1], poly 2 = next [start 2]

(7.) If exp (poly 1) = exp (poly 2)

(i) Create a newnode, [Next [polysum]]

(ii) polysum = next [polysum]

(iii) next [polysum] = start 3 (iv) exp[polysum] = exp [Poly 1] or exp [poly 2]

(iv) exp [polysum] = coeff [poly] + coeff [poly]
Coeff

(vi.) poly 1 = next [poly 1] and poly 2 = next [poly 2]

else if

exp [poly 1] > exp [exp 2]

(a) perform (i) (ii), (iii) as above

(b) exp [polysum] = exp [poly 1]

(c) coeff [polysum] = coeff [poly 1]

(d) poly 1 = next [poly 1]

else (i), (ii), (iii)

(e) perform as above

(f) exp [polysum] = exp [poly 2]

(g) coeff [polysum] = coeff [poly 2]

(h) poly 2 = next [poly 2]

Spiral

Date 13/10....

Stack

Stack is a way of organizing data having fixed rules.

Operations — Push [Insertion]
Pop [Deletion]

LIFO → [Last In First Out]

Array Representation

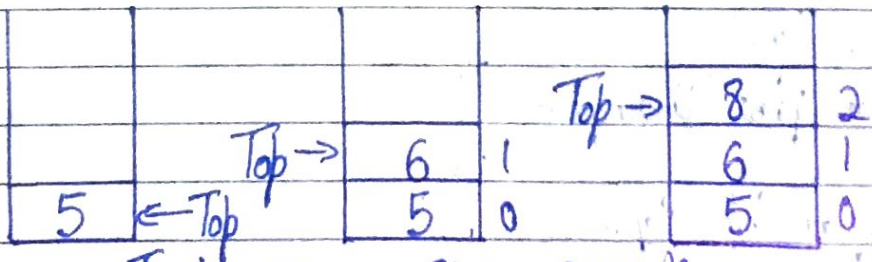
- Push (Stack (name of array), top (ptr. pt. is top most element), max_size, item)

1. If top = max_size then print "overflow" & return
2. Set top = top + 1
3. Set Stack [top] = item
4. return

- Pop (Stack, top)

1. If top = -1, then print "underflow" & return
2. item = stack [top]
3. top = top - 1
4. return

* Stack follows LIFO



If top = max_size → overflow
no further insertion

Spiral

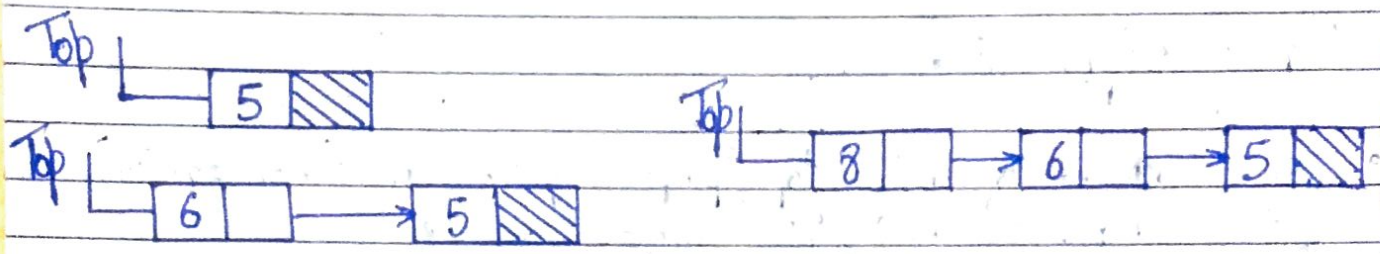
Date .....

If top = -1, the underflow
No more deletion

* Stack can be represented by — Array
                              — Linked list

Application → Tower of Hanoi

* Linked list Representation



PUSH

(i) Top = NULL
(ii) Repeat while choice is 'yes'
  (a) Create a new node say New_node
  (b) Info [New_node] = item
  (c) Next [New_node] = Top
  (d) Top = New_node
(iii) Exit

POP

(i) Repeat while choice is 'yes'
  (a) If Top = NULL then print "Underflow" & Exit
  (b) Node = Top & display info [Top]
  (c) Top = Next [node]
(2) Exit

Date .....

Polish Notation

1. Infix → If operator is b/w the operands. Ex- A+B
2. Postfix → If operator is after the operands. Ex- AB+
3. Prefix → If operator is before the operands. Ex- +AB

* Precedence [Imp]

Exponentiation-[$, ↑, ^]	Highest	3	Left to Right
Multiply & Division-[*, /]	Next Highest	2	Left to Right
Addition & Subtraction-[+, -]	Lowest	1	Left to Right

Evaluation of Postfix Expression

1. Add a right parenthesis ')' at the end of P (Postfix Expression)
2. Scan P from left to right & repeat steps 3 & 4 for each element in P until the right parenthesis ')' is encountered
3. If an operand comes, push it on-to the stack.
4. If an operator (X) then —
   a. Remove top 2 elements from stack, say A is top & B is next to top element.
   b. Evaluate B (X) A & place result back on stack
5. Set value equal to top element of stack.
6. Exit

P = 5, 6, 2, +, *, 12, 4, /, -,) → Added

(X) → represent operater.

Date .....

Postfix Evaluation

Expression	Stack
5	5
6	5, 6
2	5, 6, 2
+	5, 8
*	40
12	40, 12
4	40, 12, 4
/	40, 3
-	37

Imp Conversion of Infix to Postfix

1. Push '(' onto stack & add ')' to the end of Q (Infix)
2. Scan Q from left to right & repeat steps 3 to 6 for each element of Q until the stack is empty
3. If an operand, add it to P (postfix)
4. If a left parenthesis, push it onto stack.
5. If an operator (X) then —
   a. Repeatedly POP from stack & add to P each operator (on top of stack) which has same or higher precedence than (X).
   b. add (X) to stack.
6. If a right parenthesis appears then —
   a. Repeatedly POP from stack & add to P each operator (on top of stack) until a left parenthesis is encountered.
   b. Remove the left parenthesis (Don't add right parenthesis to P)
7. Exit.

Date .... Added

Q = A + (B * C - (D / E ↑ F) * G) * H)

S. No.	Expression	Stack	P (Postfix)
1.	A	(	A
2.	+	(+	A
3.	(	(+(	A
4.	B	(+(	AB
5.	*	(+(*	AB
6.	C	(+(*	ABC
7.	-	(+(-	ABC*
8.	(	(+(-(	ABC*
9.	D	(+(-(	ABC*D
10.	/	(+(-(/	ABC*D
11.	E	(+(-(/	ABC*DE
12.	↑	(+(-(/↑	ABC*DE
13.	F	(+(-(/↑	ABC*DEF
14.	)	(+(-	ABC*DEF↑/
15.	*	(+(-*	ABC*DEF↑/
16.	G	(+(-*	ABC*DEF↑/G
17.	)	(+	ABC*DEF↑/G*-
18.	*	(+*	ABC*DEF↑/G*-
19.	H	(+*	ABC*DEF↑/G*-H
20.	)		ABC*DEF↑/G*-H*+

Spiral

Date .....

Conversion of Infix to Prefix

1. Push ')' onto stack & add '(' to the end of R (Prefix)
2. Scan R from right to left & repeat steps 3 to 6 for each element of R until the stack is empty
3. If an operand, add it to R (Prefix)
4. If a right parenthesis, push it onto stack.
5. If an operator (X) having less precedence, then -
   (a) Repeatedly POP from stack & add to R each operator (on top of stack) which has same or higher precedence than (X)
   (b) add (X) to stack
6. If a left parenthesis then -
   (a) Repeatedly POP from stack & add to R each operator (on top of stack) until a left parenthesis is encountered.
   (b) Remove the left parenthesis.

Date .....

Added

Q → (A $B * C - D + E / F / (G + G) → Added

Reverse → (G+G) / F / E + D - C * B $ A)

S.No	Expression	Stack	P (Prefix)
1.	(	((	
2.	G	((	G
3.	+	((+	G
4.	G	((+	G G
5.	)	(	G G +
6.	/	(/	G G +
7.	F	(/	G G + F
8.	/	(//	G G + F
9.	E	(//	G G + F E
10.	+	(+	G G + F E //
11.	D	(+	G G + F E // D
12.	-	(+-	G G + F E // D
13.	C	(+-	G G + F E // D C
14.	*	(+-*	G G + F E // D C
15.	B	(+-*	G G + F E // D C B
16.	$	(+-*$	G G + F E // D C B
17.	A	(+-*$	G G + F E // D C B A
18.	)		G G + F E // D C B A $ * - +

Now, Simply Reverse it to get Prefix

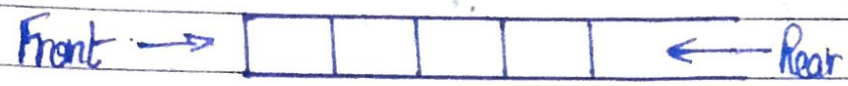
⇒ + - * $ A B C D // E F + G G

Spiral

Queue

Date 17/10/23

It is a linear list in which insertions are performed at one end while deletion from other



Deletions - Front
Insertions - Rear

FIFO principle
First In First Out

Queue has namely 2 Types → Linear, Circular

Limitation of Linear Queue

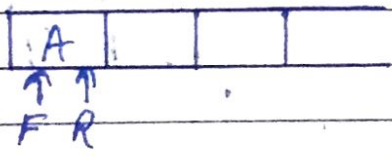
In Linear Queue we can't add data if Rear = N-1

Ex →

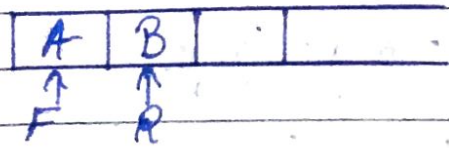
In case of Queue, elements are inserted at one end while deleting from other end because of this a no of insertion & deletion process may cause overflow condition even if there are 3 locations at the starting of the queue.

Example - Consider a queue with 4 elements & operations are performed -

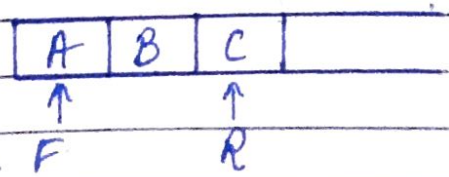
(i) Insert A



(ii) Insert B

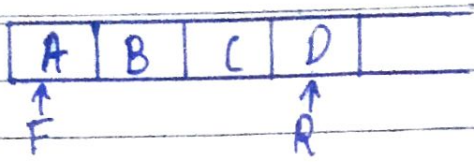


(iii) Insert C

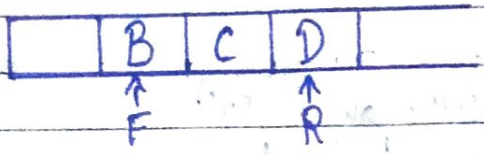


Date .....

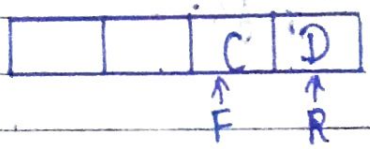
(iv) Insert D



(v) Delete A



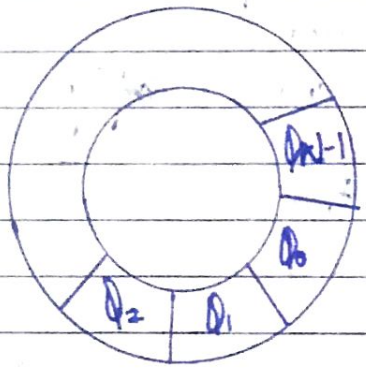
(vi) Delete B



(vii) Insert E will cause overflow even if 2 locations are free / empty.

To overcome, we follow → Circular Queue

Circular Queue is the one in which Q[0] follows Q[N-1]



Circular Queue removes the problem of linear queue, where elements can be inserted at position Q0 after QN-1.

Date .....

Array Representation

• Insertion in Linear Queue

① If Rear = N-1 then write 'overflow' & exit
② If Front = -1 (initially Queue Empty).
    Set Front = 0, Rear = 0
  else
    Set Rear = Rear + 1
③ Set Queue [Rear] = Item
④ Exit

• Insertion in Circular Queue

① If FRONT = 0 & Rear = N-1 or if FRONT = Rear + 1
  then write 'overflow' and exit
② If FRONT = -1 (initially Queue is empty)
    Set FRONT = 0, Rear = 0
  else if REAR = N-1 then
    set Rear = 0
  else
    set Rear = Rear + 1
③ Set Queue [Rear] = Item
④ Exit

Date .....

• Deletion in Linear Queue

1. If FRONT = -1 then write 'underflow' & Exit
2. Set Item = Queue [FRONT]
3. If FRONT = REAR Then [only one element], Set FRONT = -1 & REAR = -1
4. Exit

• Deletion in Circular Queue

1. If FRONT = -1, then write 'underflow' & exit
2. Set Item = Queue [FRONT]
3. If FRONT = REAR then [only one element]
   Set FRONT = -1 and REAR = -1

else if FRONT = N-1 then set FRONT = 0

4. Exit

Linked List Implementation

• Insertion

1. If FRONT = NULL (Queue empty)
   - a. Create a node, queue
   - b. next [queue] = FRONT
   - c. FRONT = queue
   - d. Rear = Front

   [Note: If we interchange these 2 by (4 & b) it will become circular]

   else
   - a. Queue = FRONT
   - b. while Queue ≠ Rear
     Queue = Next [queue]
2. Create a new node and info [new-node] = Item

Date .....

d. new [new_node] = next [queue]
e. next [queue] = new_node & Rear = new_node
2. Exit

Deletion

1. If FRONT = NULL then 'underflow' & exit
   else if
   FRONT = Rear then
   Set FRONT = NULL & Rear = NULL
   else
   Queue = FRONT
   Process Info [Queue]
   Front = next [Queue]

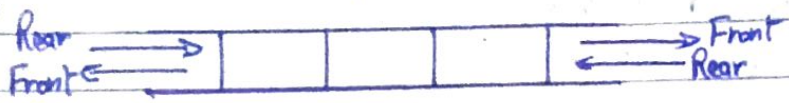
   Next [Rear] = Front ——> { This is an addition for circular linked list else it is a linear. }
2. Exit

Dequeue

Date .....

[Double Ended Queue]

Double Ended Queue is one in which insertion & deletion can be performed at both ends.



(i) Input Restricted Dequeue
(ii) Output Restricted Dequeue
— [Types of Dequeue]

(i) Input Restricted Dequeue → In this, input from one end while deletion from both ends.

(ii) Output Restricted Dequeue → In this, Output from one end while insertion from both ends.

PRIORITY QUEUE

It is the one in which insertion & deletion can be performed at any point depending on priority. Elements with highest priority are processed first, It is used in process scheduling.
while performing deletion elements with highest priority are deleted first.

2 ways to represent Priority Queue in memory are —

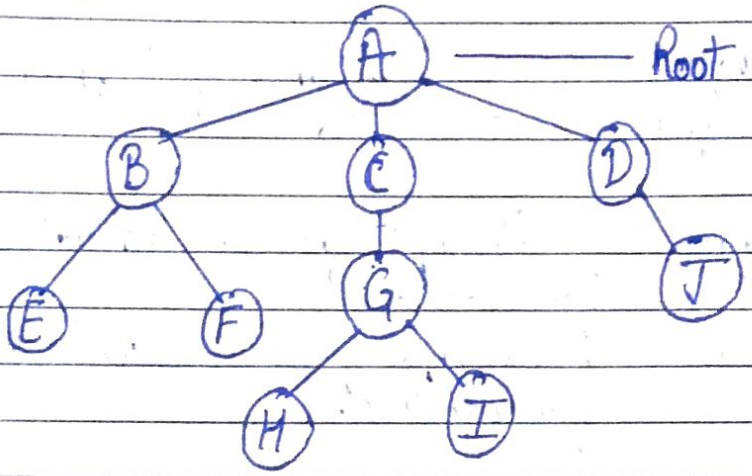
1. One Way List
2. Multiple Queue

Date 18/10

Trees

It is a non-linear Data Structure which has ~

1. A special node called root node
2. Set of subtrees T_1, T_2, T_3 ----



- A is the parent of B, C & D
- B, C & D are child nodes for A

* Nodes that have common parents are called siblings. Ex- B, C, D

* Leaf node / Terminating Node / External node

→ Nodes that don't have any child. Ex ~ E, F, H, I, J.

* Internal Nodes

→ Nodes that have atleast one child are called Internal nodes.
Ex → A, B, C, D, G

Date .....

* Ancestors / Predecessors

They are all the nodes that falls in path while traversing from that node to root node.
Ex- If node is E, then ancestor are B & A

* Successors

All the nodes that falls in path while traversing from nodes to leaf node. Ex → Successors of B are E & F

* Degree of nodes

It is the no. of child nodes of a node which are known as Degree of that node.

deg (A) = B

NOTE : Degree of Leaf node is always zero.

* Degree of tree

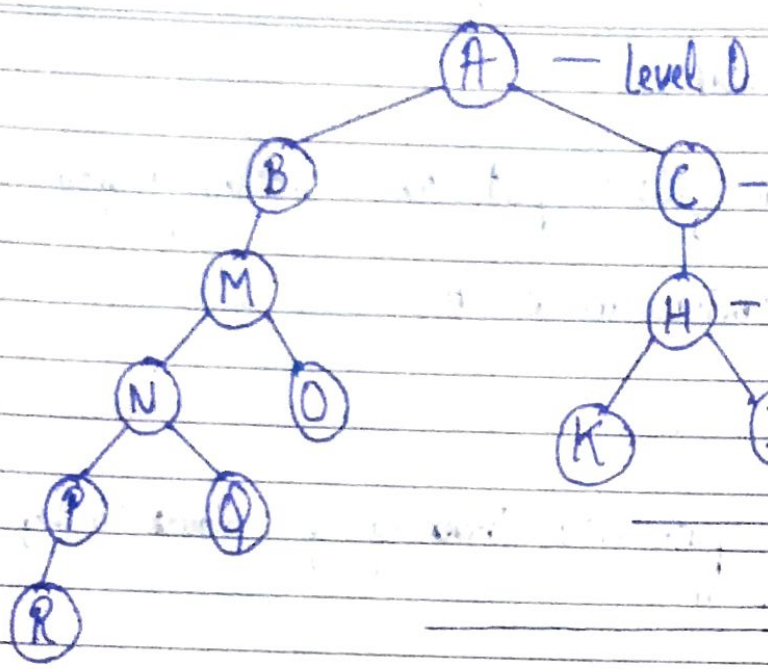
It is the maximum degree of node in the tree. Max deg = 3, so deg (tree) = 3

* Height of Tree

It is the maximum ~~degree~~ no. of node in the ~~tree. Max deg~~ longest branch or

$$\boxed{\text{height} = \text{max} - \text{levels} + 1}$$

Date .....



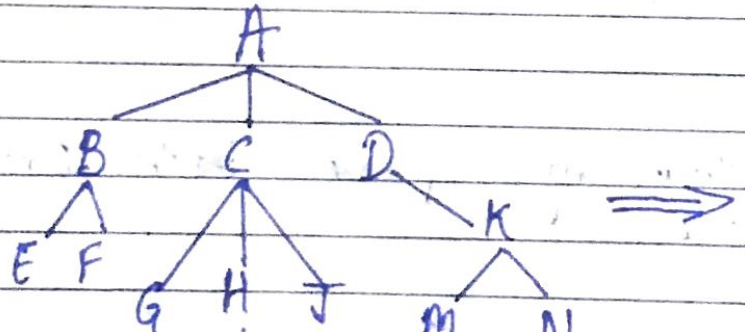
→ 2 way to calculate height of tree

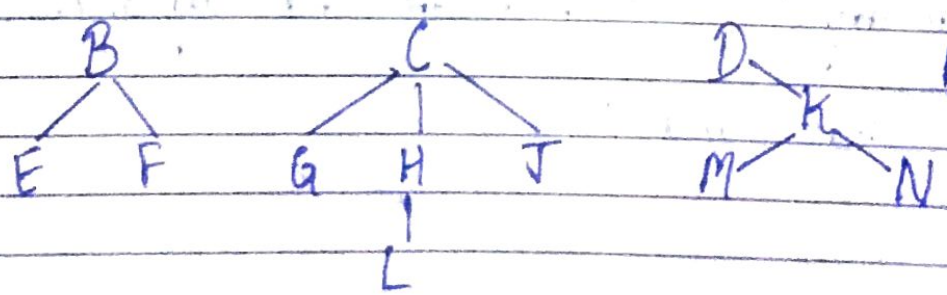
So height = 6
(Max no. of node in the longest branch/node)

{Longest = Number of Node
So; height = 6}

2nd Way → Height = Max level + 1
= 5 + 1 = 6

FOREST → It is the set of n greater than/equal to 0. $n \geq 0$, disjoint tree where n represent no. of nodes in the tree. & if we remove the root node, we get a forest.



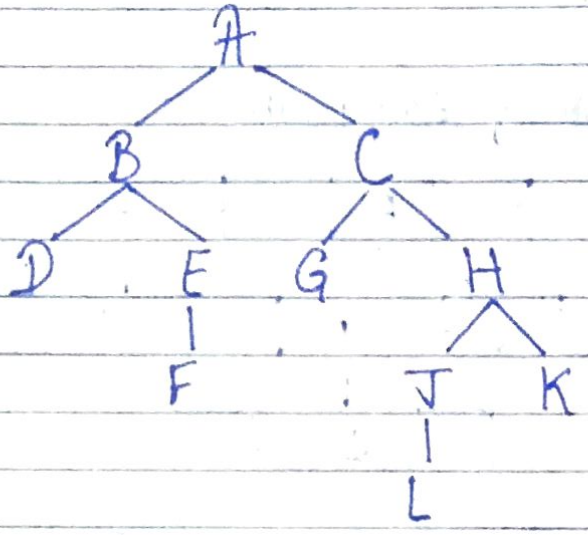


FOREST of 3 Tree

Date .....

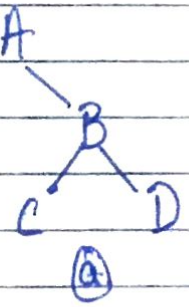
Binary Tree

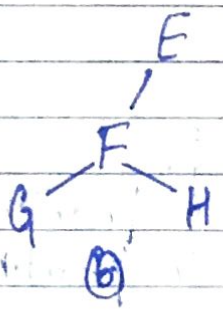
If every node of a tree can have atmost degree 2 then it is said to be a Binary Tree.

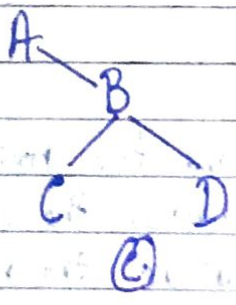


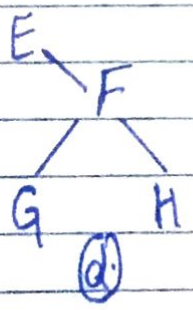
- Two binary trees T & T' are <u>similar</u> if they have same structure or shape.
- Two binary trees T & T' are <u>copies</u> if they are structurally identical, and the nodes have the same value.

Ex-



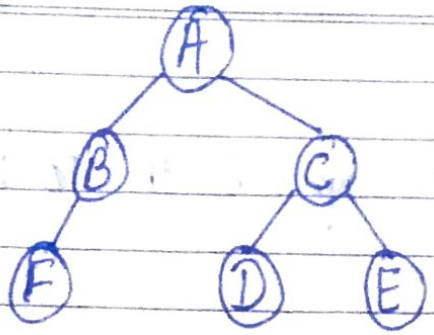






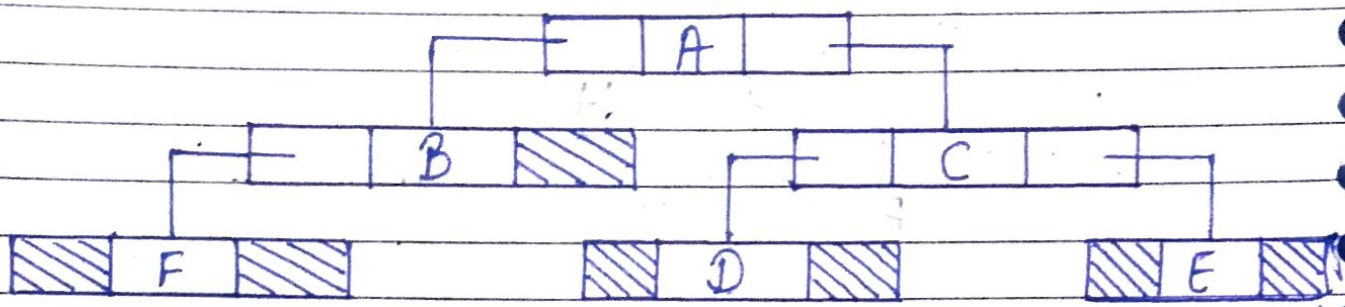
(a), (c), (d) are similar
(a) & (c) are copies

Date .....



LPTR → Left ptr
RPTR → Right ptr

Linked list Representation of Binary Tree –



Tree Traversal

S → Root Tree
L → Left Subtree
R → Right Subtree

- Pre Order [SLR]
- In Order [LSR]
- Post Order [LRS]

Pre Order →

- Process the root node i.e S.
- Traverse the left subtree in preorder.
- Traverse the right subtree in preorder.

In Order →

- Traverse the left subtree inorder.
- Process the root node i.e. S.
- Traverse the right subtree inorder.

Spiral

Date .....

Post Order →

- Traverse the left subtree in post order.
- Traverse the right subtree in post order.
- Process the root node i.e. $.

Traversal Algorithm < Recursive Algorithm
Non-Recursive Algorithm

Recursive Algorithm for Tree Traversal

R PREORDER (T) – Given a binary tree where root node address is given by T.

(i) [Process the root node]
If T ≠ NULL
then write (Data (T))
else
write (Empty Tree)
return.

(ii) [Process left Subtree]
If LPTR ≠ NULL
CALL RPREORDER [LPTR (T)]

(iii) [Process Right Subtree]
If RPTR ≠ NULL
CALL RPREORDER [RPTR (T)]

(iv) Return

RINORDER (T)

(i) If T = NULL then
write ("Empty Tree")
return

Spiral

Date .....

② [Process left subtree]
LPTR (T) ≠ NULL
CALL R_INORDER (LPTR(T))
③ [Process the root node]
write (Data (T))
④ [Process the right subtree]
CALL R_INORDER (RPTR(T))
⑤ Return

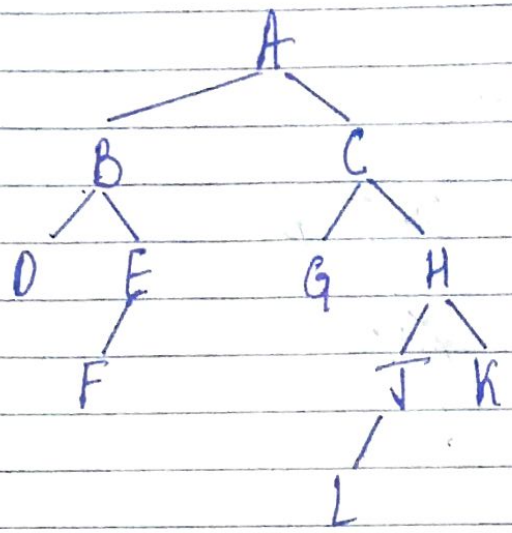
R_POSTORDER (LRS)

① If T = NULL then
write ("Empty tree")
return
② [Process left subtree]
If LPTR (T) ≠ NULL
CALL L_POSTORDER (LPTR (T))
③ [Process the right subtree]
RPTR(T) ≠ NULL
CALL R_POSTORDER (RPTR(T))
④ [Process the root node]
write (Data (T))
⑤ Return

Date .....

Tree Traversal Questions

(i)

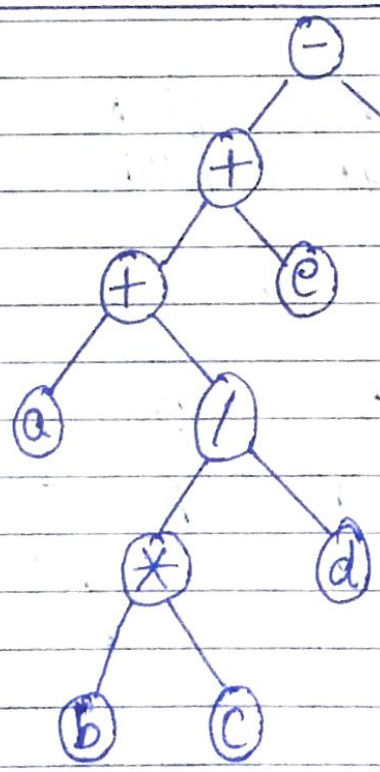


[SLR] PREORDER - ABDEFCGHJLK

[LSR] INORDER - DBFEAGCLJHK

[LRS] POSTORDER - DFEBGLJKHCA

(ii)



[SLR] PREORDER -> - + + a / * b c d e f

[LSR] INORDER - a + b * c / d + e - f

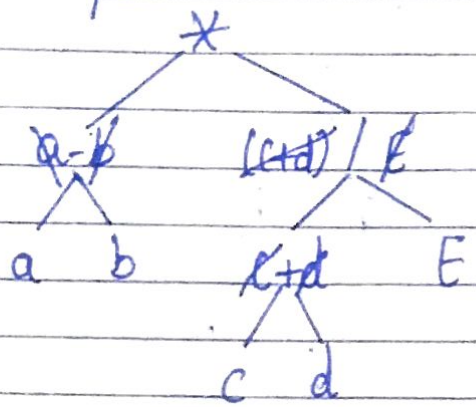
[LRS] POSTORDER - a b c * d / + e + f -

Spiral

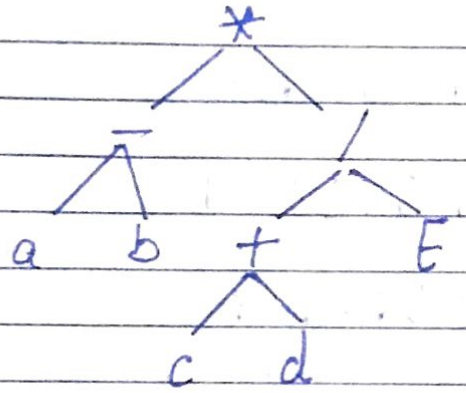
Date .....

Expression Tree

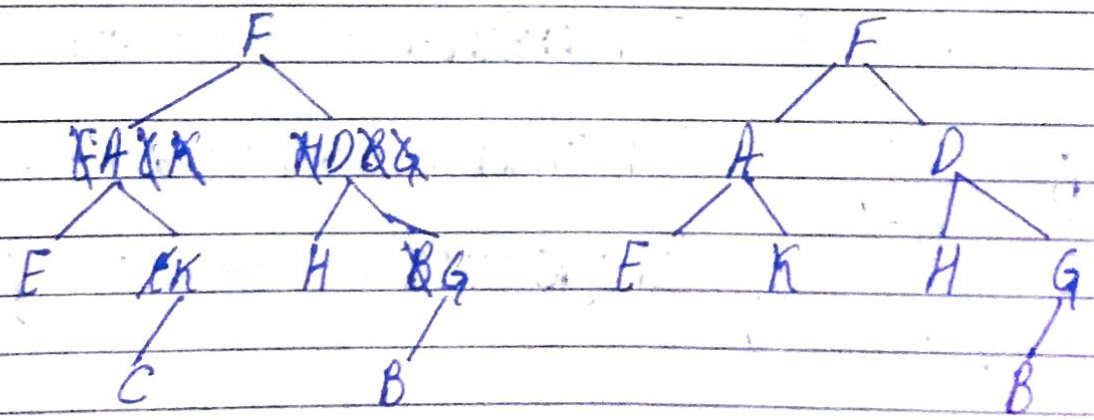
(a-b) * ((c+d) / E)



Final -



Imp Ques → Inorder – EACKFHDBG [LSR]
Preorder – FAEKCDHGB [SLR]



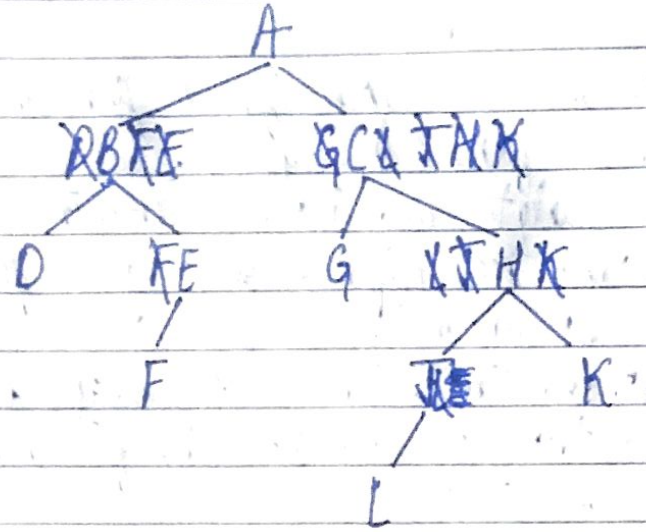
Spiral

Date .....

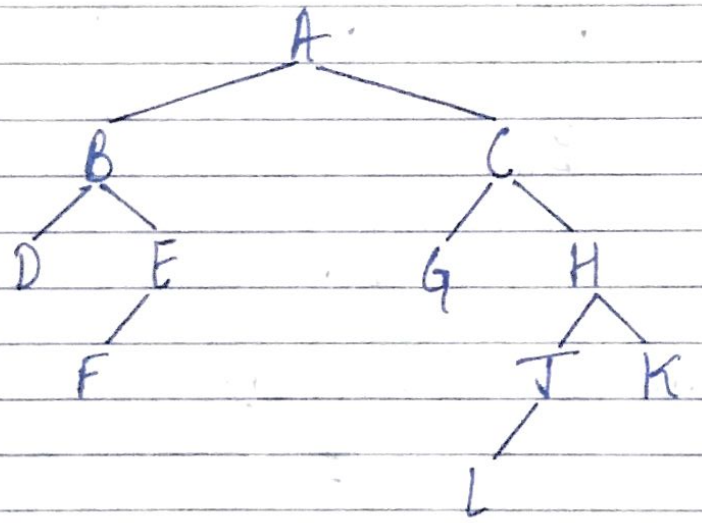
PostOrder → [L R S]

Q Preorder → A B D E F G H J L K [S L R]

InOrder → D B F E A G C L J H K [L S R]



Final →



Date 20/10/.....

Binary Search Tree [BST]

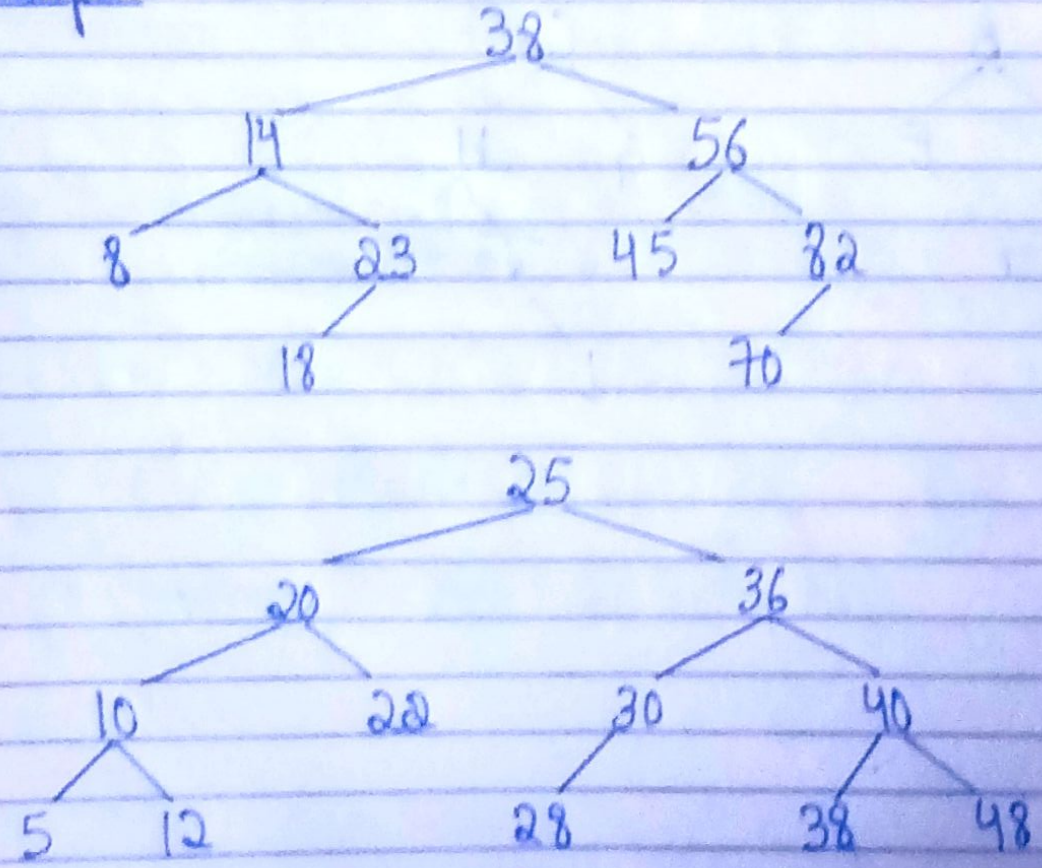
- It's also known as Binary Sorted Tree

The structure enables one to search for & find an element with an average running time $f(n) = O(\log_2 n)$.

- It enables to easily delete and insert elements.
- The definition of binary tree depends on a given field whose values are distinct & may be ordered.
- Suppose T is a binary tree. Then T is called a binary search tree [or Binary sorted tree] if each node N of T has the following property:-

The value of N is greater than every value in the left subtree of N & is less than every value in right subtree of N

Example →



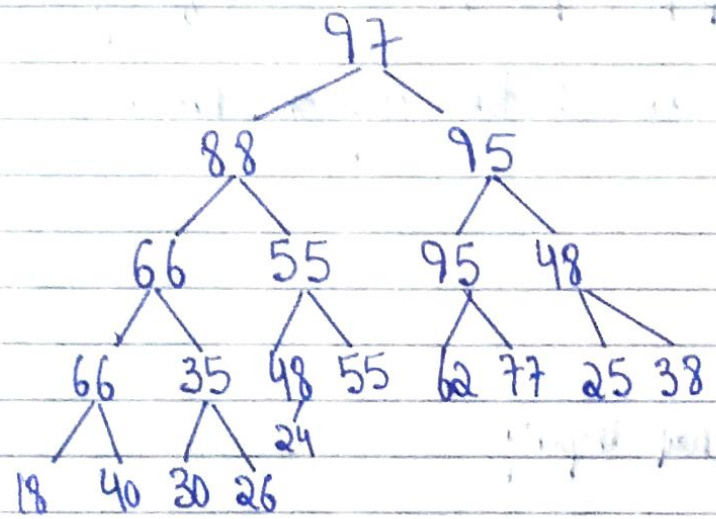
Spiral

Date 23/10.....

Heap / Heap Sort

Suppose H is a Complete Binary Tree with n elements then, H is called Heap / Maxheap. The heap is used in an elegant sorting algorithm called heapsort. If each node N of H has the following property: The value of N is greater than or equal to the value of each of the children of N. ~~Maxheap if~~ then that's known as Maxheap.

Minheap is when the value at N is less than or equal to the value to any of the children of N.



97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Right & Left Children

Tree [2K+1] Tree [2K] → Ex- 66, → 4th

so, T[2(4)+1] T[2[4]] = T[8]

T[9]

Date .....

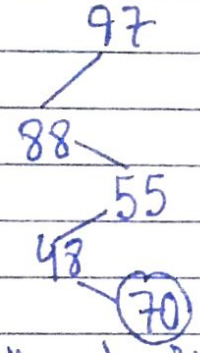
Parent of any root node Tree [T] is Tree [T/2] Integer Division

$$T\left[\frac{12}{2}\right] = T[6]$$

Insertion into Heap

① First adjoin ITEM at the end of H. So that H is still a complete tree but not necessarily a heap.

② Let the ITEM rise to its appropriate place, so that H is finally a heap

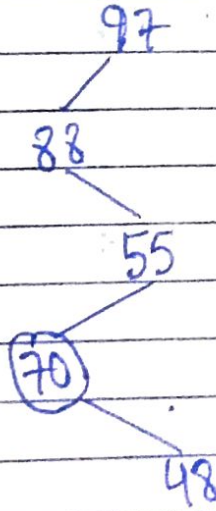


→ We will take only one branch

③ Since it doesn't support Heap Property.

70 > 48

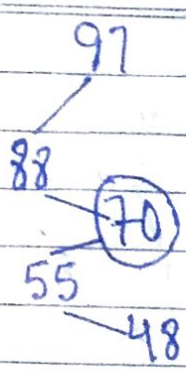
So, interchange 48 & 70



④ again 70 > 55

So, interchange 55 & 70

Date .....

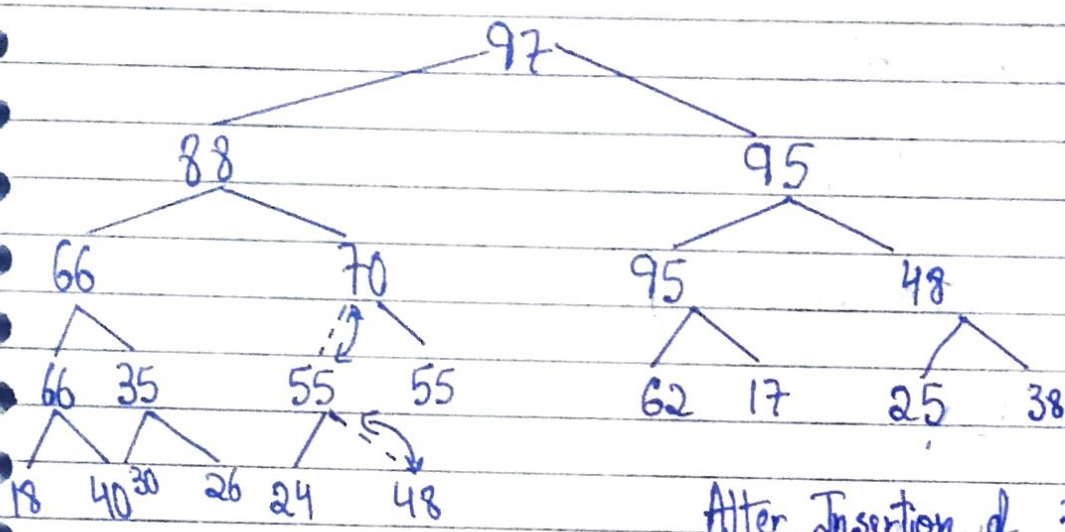


(c.) Now Heap property is satisfied.

Since 70 < 88

So No Interchange

Now place this branch into that tree →



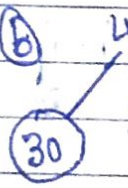
After Insertion of 70

Build a Heap

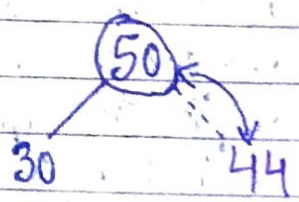
→ 44, 30, 50, 22, 60, 55, 77, 55

(a.) 

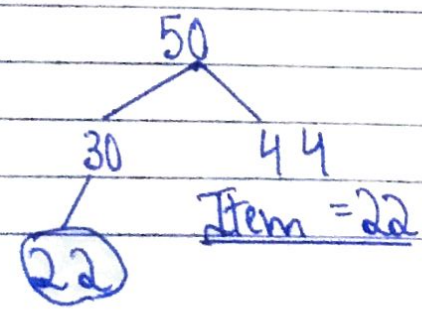
Item = 44

(b.) 

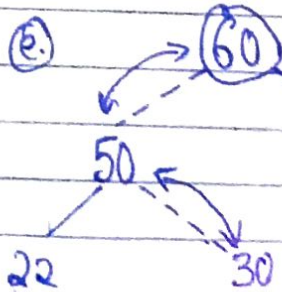
Item = 30

(c.) 

Item = 50.

(d.) 

Item = 22

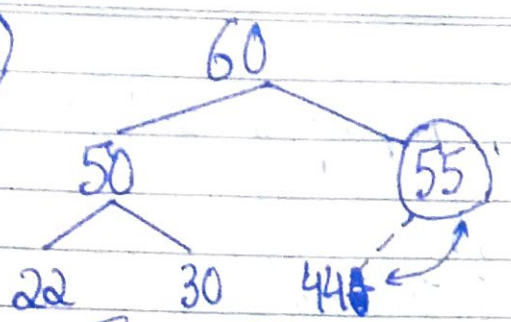
(e.) 

Item = 60

Spiral

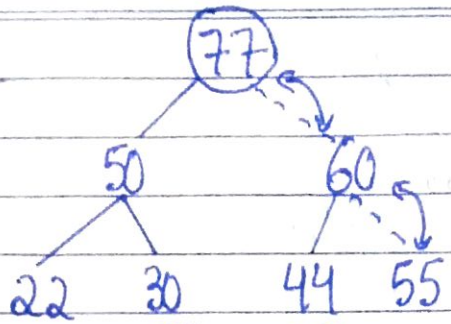
Date .....

(f)



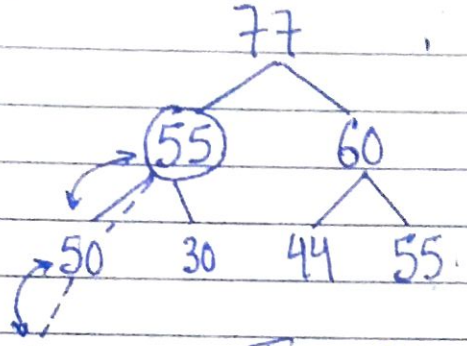
Item = 55

(g)



Item = 77

(h)



Item = 55

Deletion from Heap

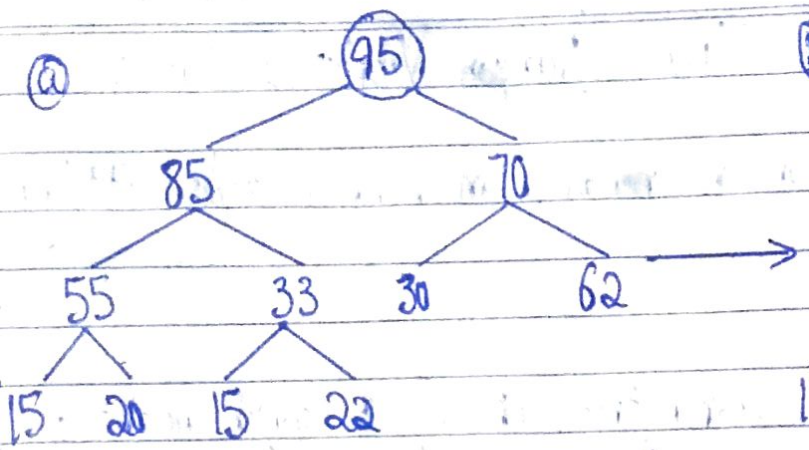
Deleting Root Node of Heap

Suppose H is a heap with N elements & suppose we want to delete root R of H.

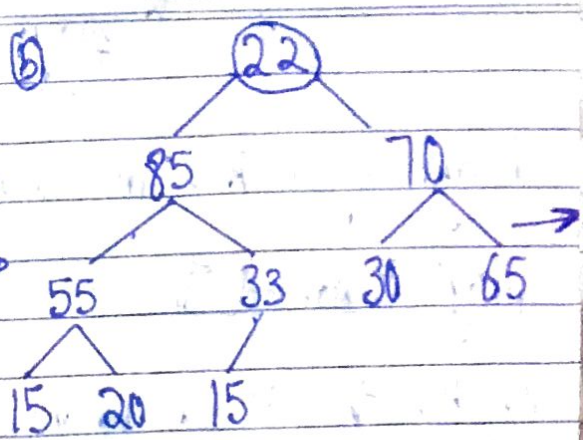
① Assign the root R to some variable Item.
② Replace the deleted node R by last node N. So that, it is a complete tree but not necessarily a heap.
③ Let N sink to its appropriate place. So that H is finally a Heap. This step 3 is Reheap.

Date .....

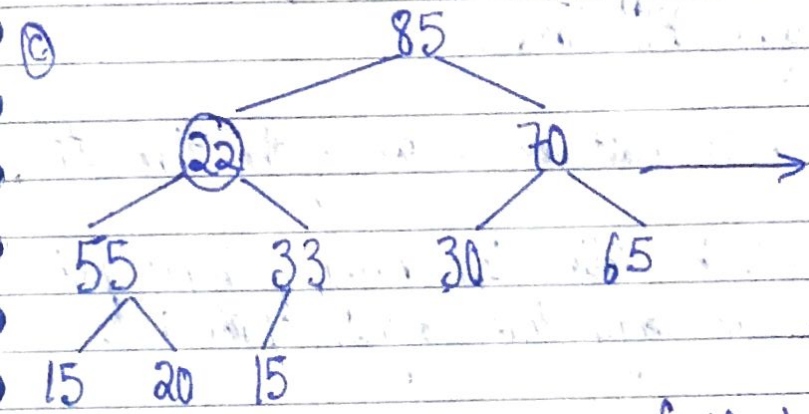
(a)



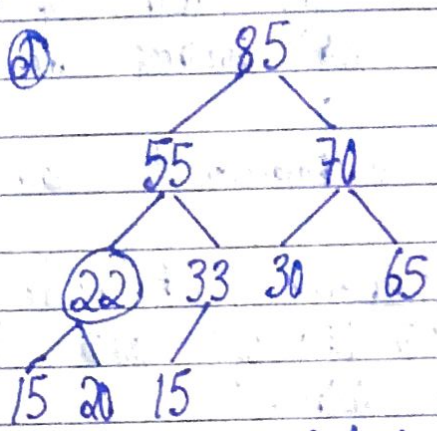
(b)



(c)



(d)



Complexity of heap $f(n) = O(n \log_2 n)$

Date 25/10.....

* AVL Tree [Adelson Velskii & Landis]

Height of an AVL Tree with n nodes, can never exceeds 1.44 $\log n$.

Height Balanced Tree -

An AVL tree is a binary search tree (BST) in which height of left & right subtree of root differ by atmost 1 & in which left & right subtrees are given AVL Tree.

→ In a Balanced tree each node must be in one of the three states.

① Left Heavy → A node will be called left Heavy, if longest path is in its left subtree. is one larger than longest path in its right subtree.

② Balanced → A node is called Balanced, if longest path in both of the subtrees are equal.

③ Right Heavy → A node is called Right Heavy if longest path in its right ~~heavy~~ subtree is one larger than longest path in its left subtree.

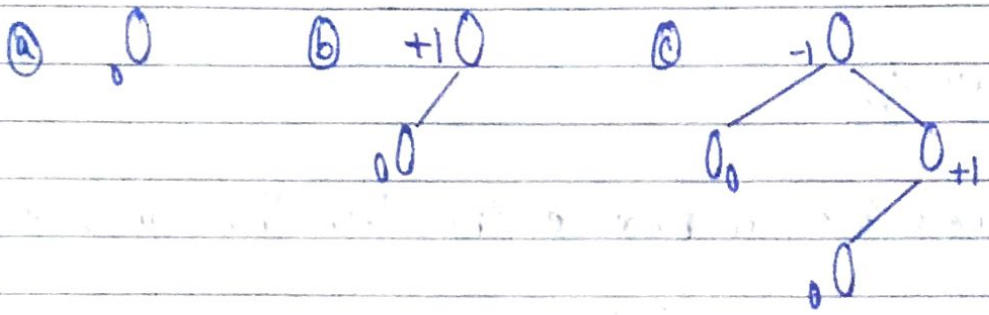
→ If there exists a node in tree which doesn't satisfied any of the above mention properties then this type of node is called Unbalanced.

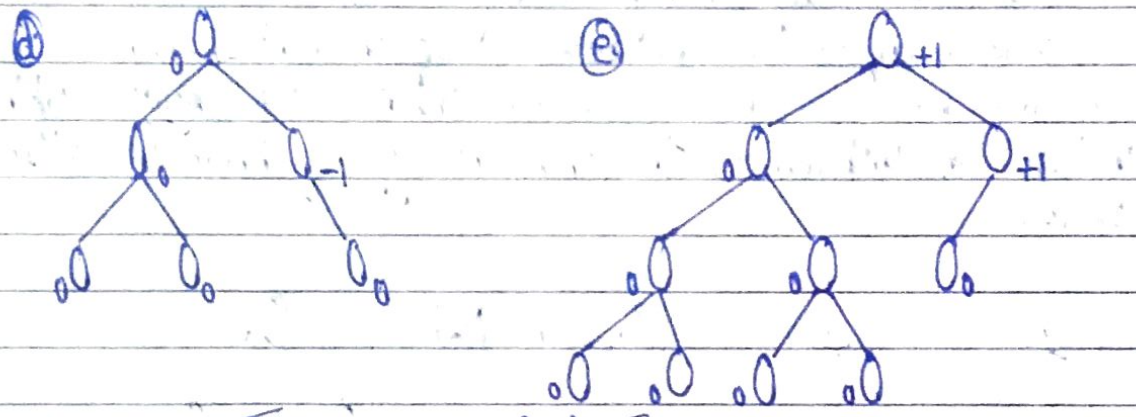
Spiral

Date .....

Balance Factor

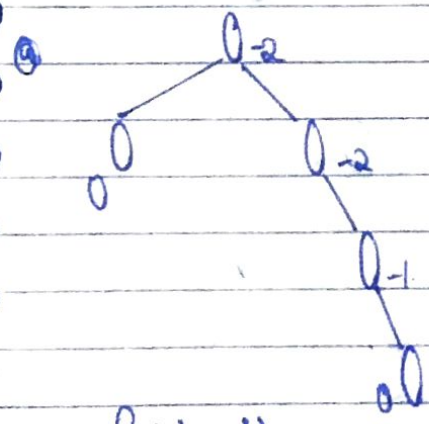
Balance Factor of a node in a Binary tree is defined as →
Height of left subtree − Height of right subtree



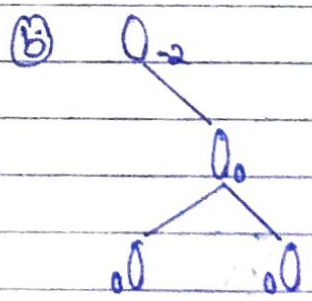


These are all AVL Trees

* In An AVL tree, Balanced Factor should be 0, 1, -1



Right Heavy [Not AVL]

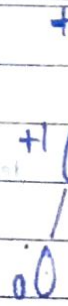


Right Heavy [Not AVL]

Scroll

Date .....

(1.)



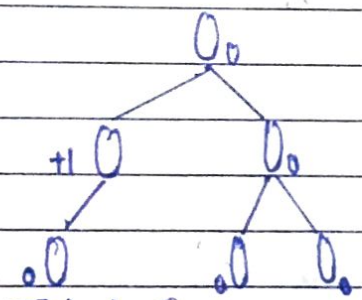
→ Left Heavy

→ This is not Balanced Unbalanced

Insertion in AVL Tree

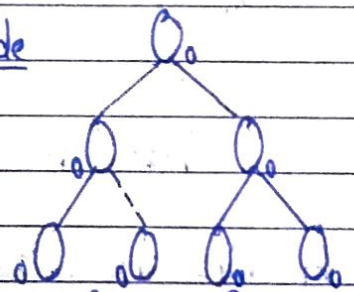
If we insert a node into a Balanced tree then there are 3 cases

1. The node was either left or right heavy & now become balanced.
2. The node was balanced & now become left or right heavy.
3. The node was heavy & now node has inserted in heavy subtree which creates an unbalanced tree, then this type of Node is called Critical node.



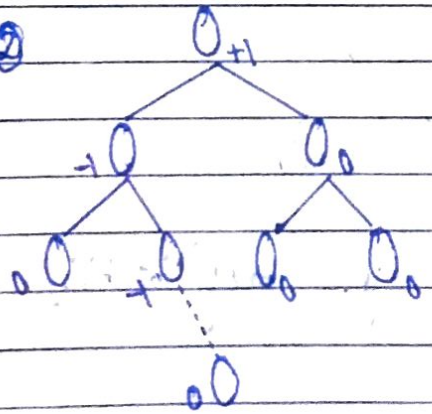
Initial Tree

① Insert Node

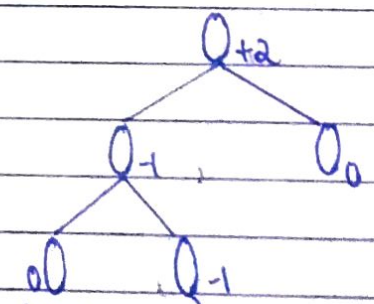


It was left heavy & now Balanced

(2)



(3)



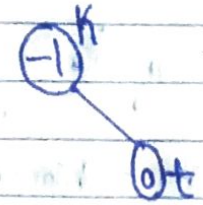
Case 3:
The node was left heavy
& now become Critical Node

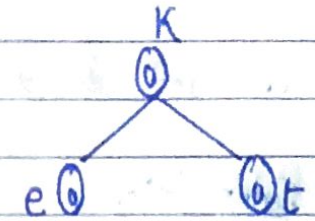
Spiral

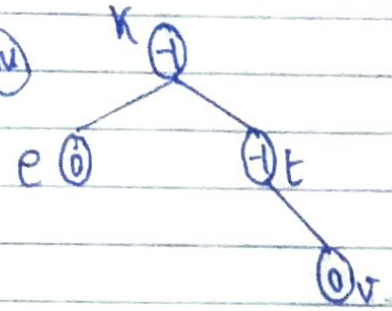
Date .....

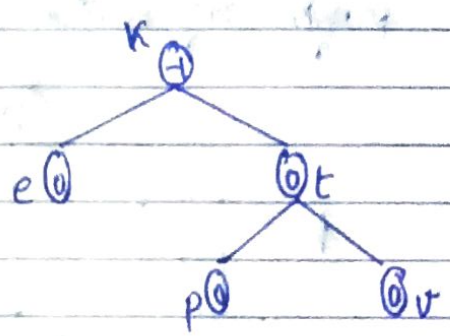
K, T, e, v, p, a, m, u

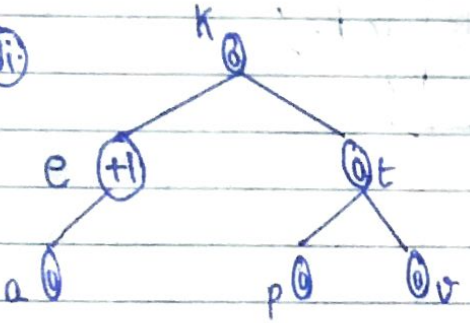
(i) K(0)

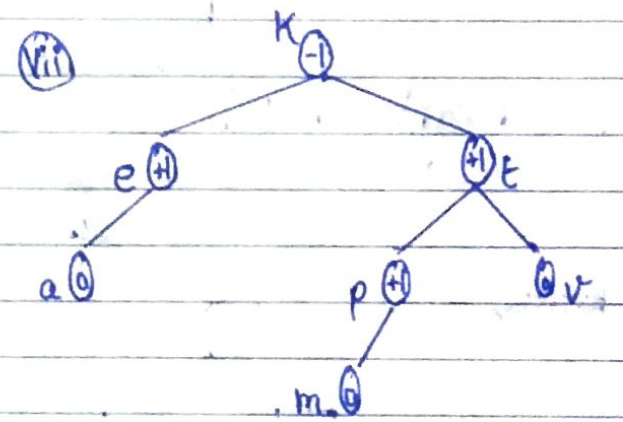
(ii)


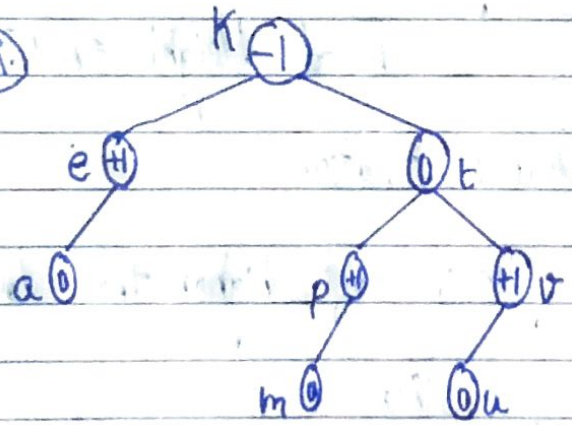
(iii)


(iv)


(v)


(vi)


(vii)


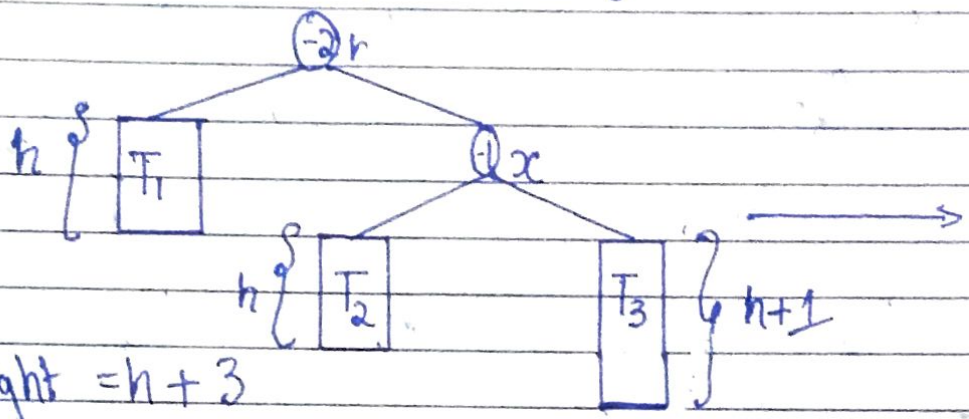
(viii)


Date .....

AVL Tree Rotation

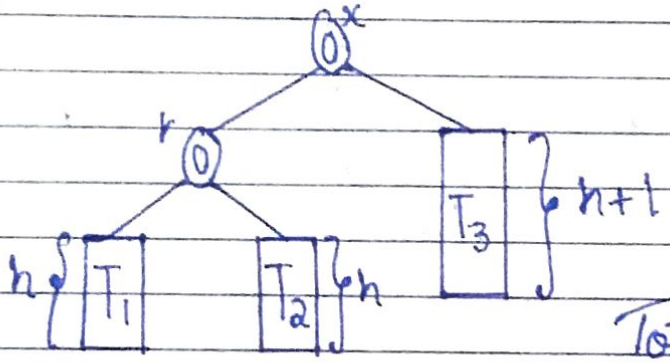
Let r be the root & x be the root of its right subtree.

Case 1 → Right Higher - When x is right higher then we need to perform left rotation for rebalancing.



Total Height = h + 3

Left Rotation →



Total Height = h + 2

Case 2 : → Left Higher Double Rotation

* In this case, when balanced factor of x is left higher then double rotation needs to be done.

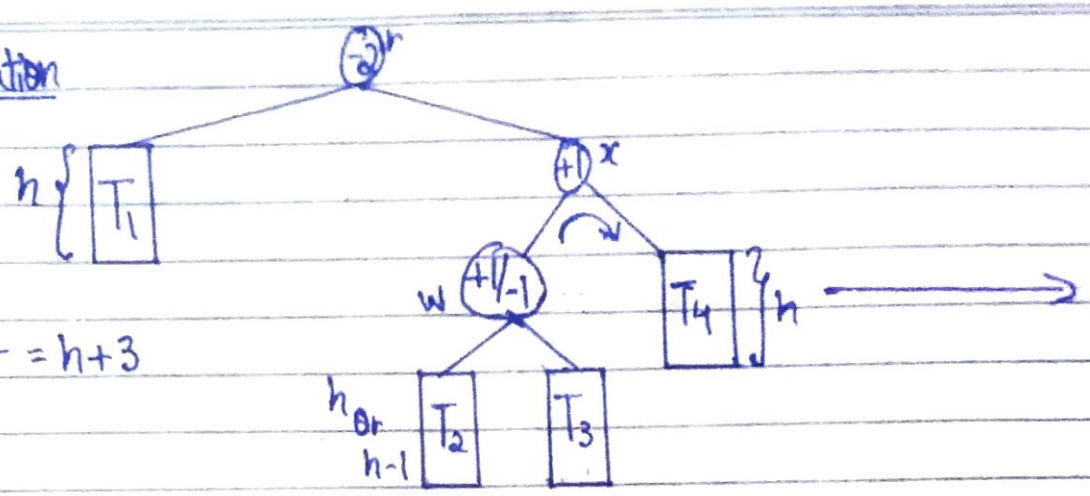
→ Let w be the root of left subtree of x, then

1. First rotate the subtree with root x to the right. So that w becomes the root node.
2. Then rotate the tree with root r to the left then w will become the root node.

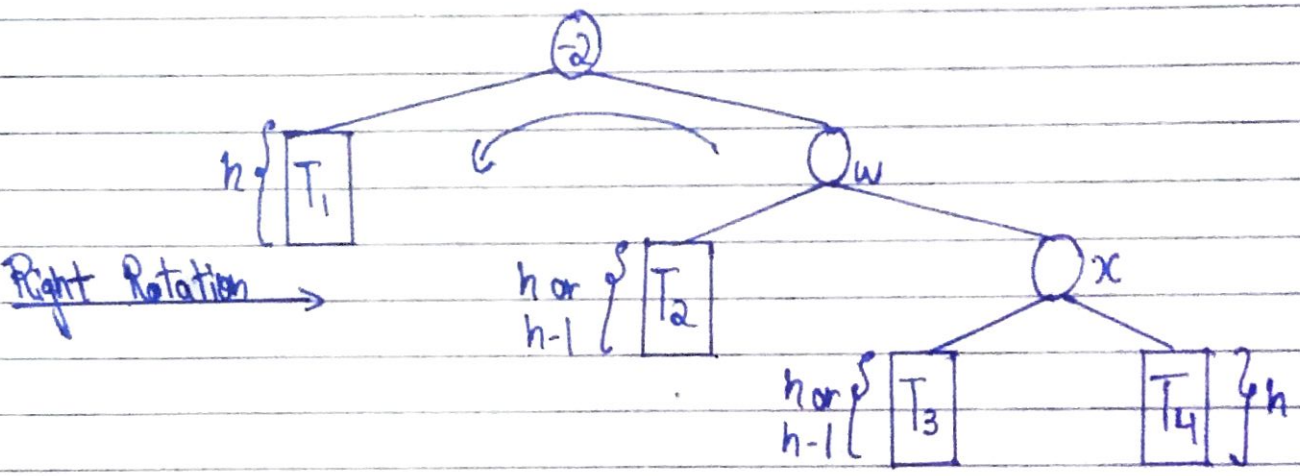
Spiral

Date .....

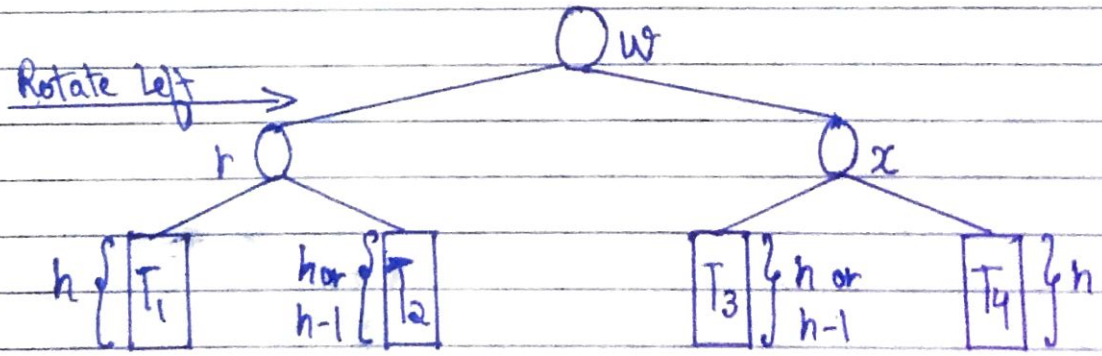
Double Rotation



Total height = h+3



Right Rotation →



Total Height = h+2 → AVL

Date .....

* K, m, u, t, v, p.

(a) (0) K

(b) (-1) K
(0) m

(c) K (-2)
(-1) m
(0) u

Right heavy
So, single
rotation

(d) (0) m
K (0)
(0) u

(e) (-1) m
K (0)
(+1) u
t (0)

(f) (-1) m
K (0)
(0) u
t (0)
(0) v

(g) (-2) m
K (0)
(+1) u
t (+1)
(0) v
p (0)

Case-2
Double Rotation

(h) (-2) m
K (0)
(-1) t
p (0)
(-1) u
(0) v

left Rotation →

(i) (0) t
m (0)
(-1) u
(0) K
(0) p
(0) v

Spiral

Date .....

B-tree definition

A B-tree of order m is an m-way search tree in which

1. All leaves are on the same level.
2. All internal nodes except the root have at most m non-empty children and at least $\lceil m/2 \rceil$ non-empty children.
3. The no. of keys in each internal node is one less than the no. of its non-empty children, and these keys partition the keys in the children in the fashion of a search tree.
4. The root has at most m children but may have as few as 2 if it is not a leaf, or none if the tree consists of the root alone.

→ A B-tree of order n is also called n--n (n-1) tree or (n-1)--n tree which indicated that each node in a B-tree has a maximum of n-1 keys and n children. Thus, a 4-5 or 5-4 tree is a B-tree of order 5.

Order : How many non-empty children a node can have.
Key : [Maximum or minimum no. of non-empty children] - 1

Insertion into B-tree

1. Locate the leaf into which the key should be inserted.
2. If the located leaf is not full, add the key to the leaf.
3. If the located leaf is full, split the full leaf in two nodes, a left & right leaf at the median key, which is then sent up into the parent node.

Date .....

Ex: Insert the following letters into an empty B-tree of order 5:

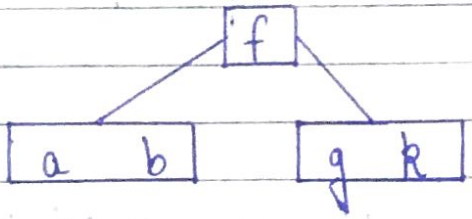
• a g f b k d h m j e s i r x c l n t u p

• Max. children = 5
• Max. Key = 5 - 1 = 4

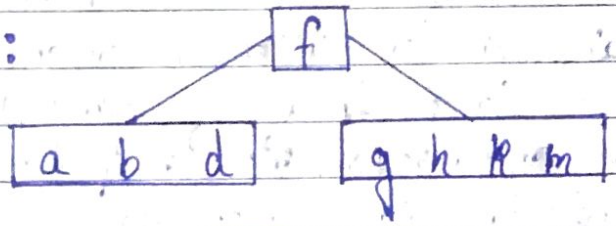
Min. Children = $\lceil m/2 \rceil = \lceil 5/2 \rceil = 3$
Min. Key = $\lceil m/2 \rceil - 1 = 3 - 1 = 2$

① a, g, f, b : [a b f g]

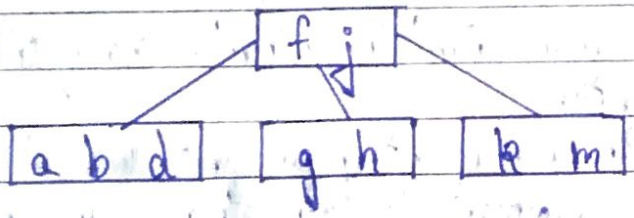
② k :



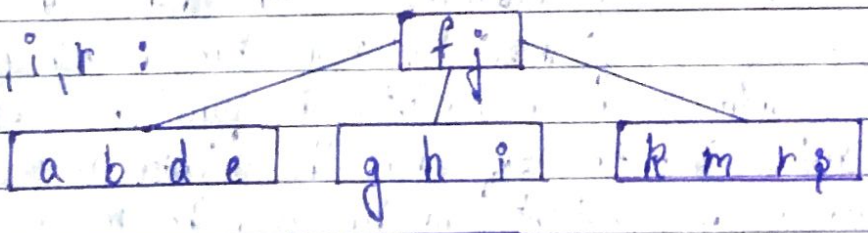
③ d, h, m :



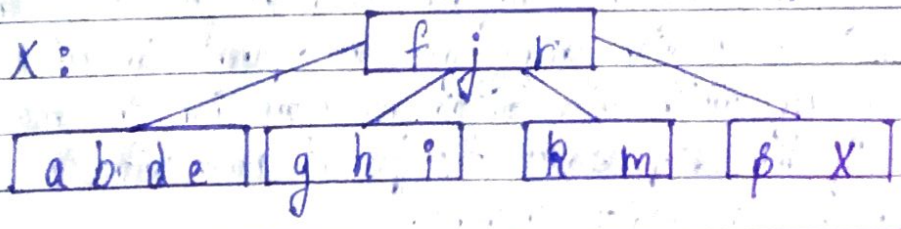
④ j :



⑤ e, s, i, r :



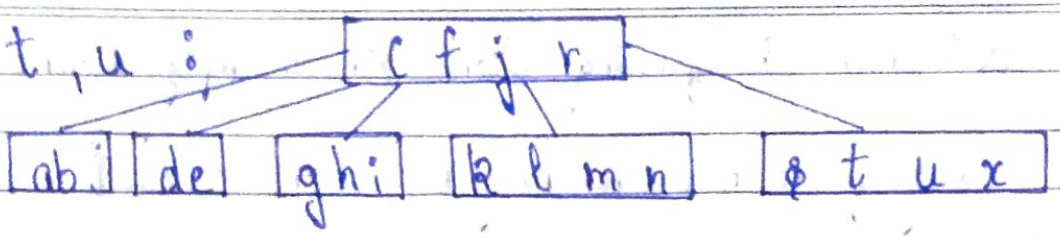
⑥ x :



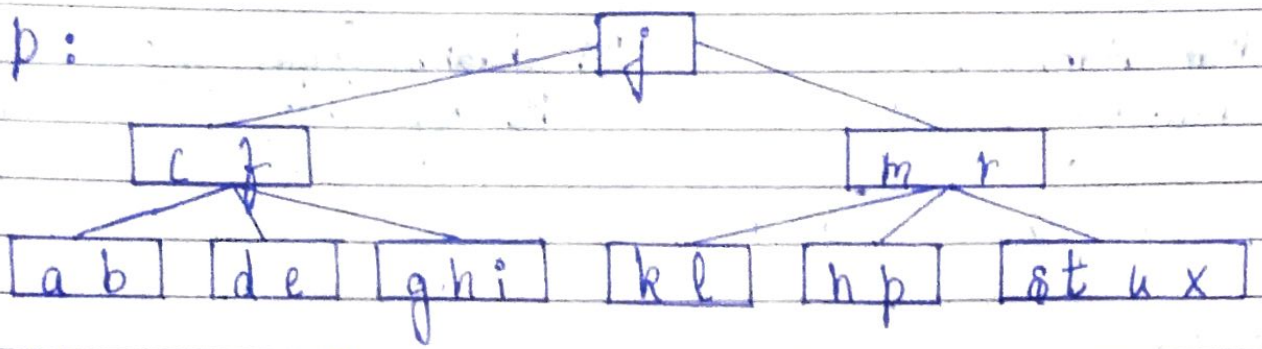
Spiral

Date .....

(7.) c, l, n, t, u :



(8.) p :

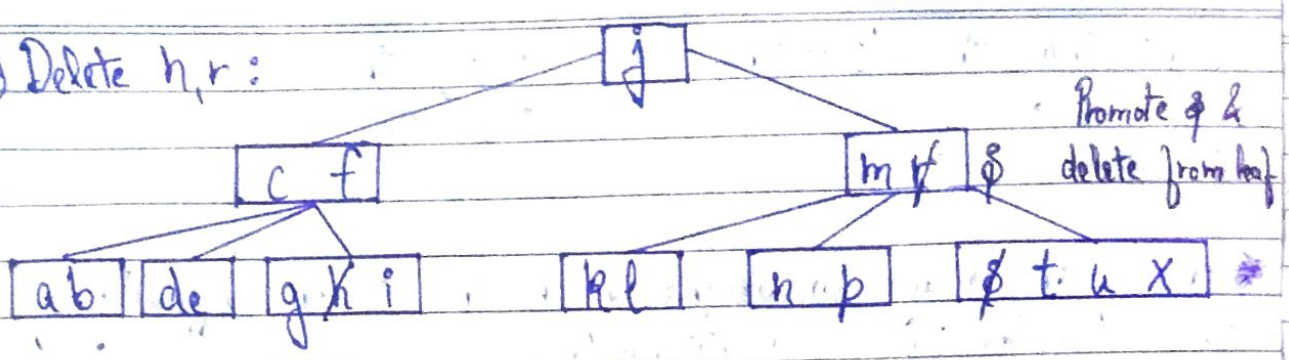


Deletion from B-tree

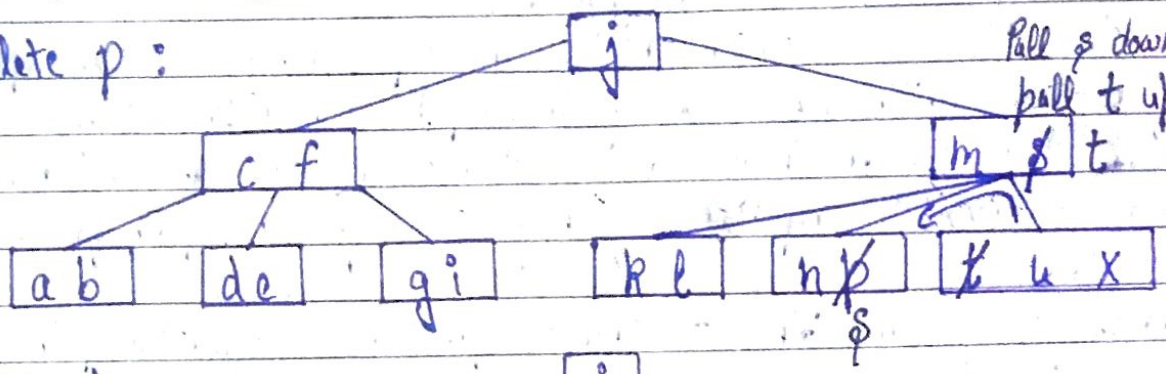
- Case 1 : If the entry that is to be deleted is not in a leaf, then its immediate predecessor (or successor) under the natural order of keys is guaranteed to be in a leaf. Hence, we can promote the immediate predecessor (or successor) into the position occupied by the deleted entry, & delete the entry from the leaf.
- Case 2 : If the leaf contains more than the minimum no. of entries, then there is no problem in deletion.
- Case 3 : If the leaf contains the min. no., then we first look at the two leaves that are immediately adjacent to each other & are children of the same node. If one of these has more than the min. no. of entries, then one of them can be moved into the parent node, & the entry from the parent moved into the leaf where the deletion is occurring.
- Case 4 : If finally, the adjacent leaf has only the min no of entries, then the two leaves & the median entry from the parent can all be combined as one new leaf, which will contain no more than the max. no. of entries allowed. If this step leaves the parent node with too few entries, then the process propagates upward.

Date .....

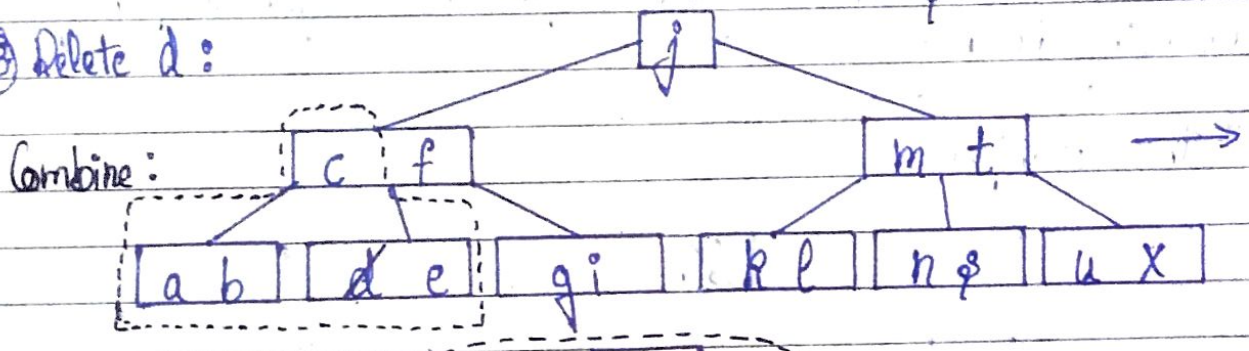
① Delete h, r:

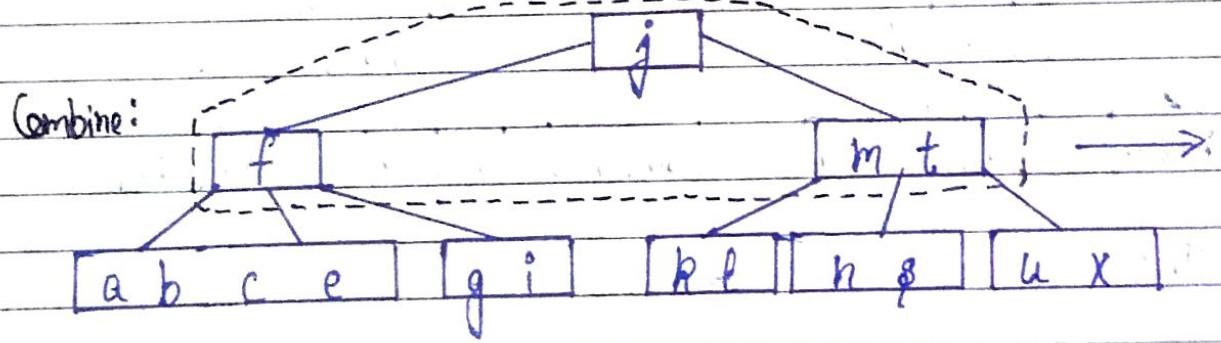


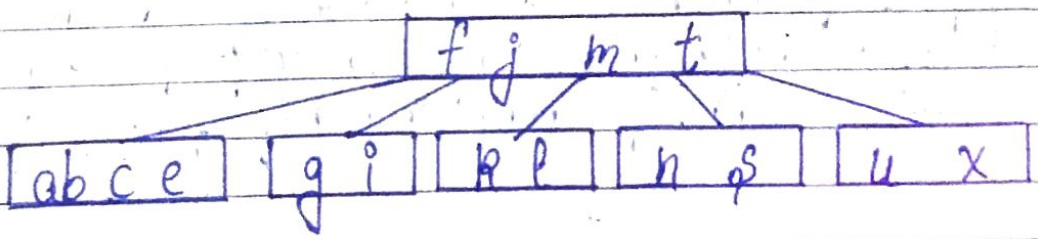
② Delete p:



③ Delete d:







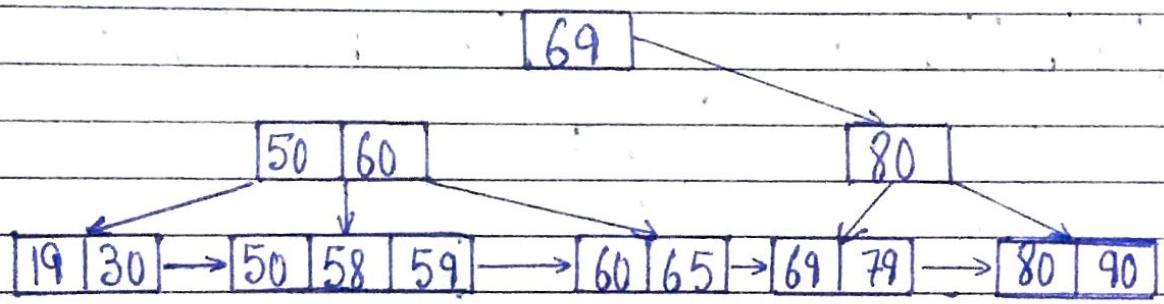
Date .....

Improving the B-tree
- B+ tree
- B* tree

B+ tree

- One of the major drawbacks of B tree is the difficulty of traversing the tree sequentially.
- B+ tree is used which retains the rapid random-access property of B-tree while also allowing the rapid sequential access.
- In B+ tree, all the Keys are maintained in leaves and all non-leaf nodes contains the replication of Keys to define the path for locating records.
- These leaves are linked together to provide a sequential path for traversing the Key.

B+ Tree Example



B* Tree

- It is used for improving the storage utilization of a B Tree
- A B* Tree is a B-tree in which every node, except possibly the root is at least two-third full rather than half full.

Date .20/11.....

UNIT - III

Sorting has 2 types namely Internal & External

Internal Sorting	External Sorting
(1) All data i.e to be sorted can be adjusted at a time in main memory.	If data i.e to be sorting is too large to be sorted in main memory, then its some part has to be stored in auxiliary memory like harddisk, magnetic disk, etc
(2) Data access time is too less i.e in nano seconds. (10^{-9} sec)	Data access time is too large i.e. in milliseconds. (10^{-3} sec)
(3) It may access 1 record at a time.	It doesn't not one record at a time but it reads a complete page or block at a time.
(4) Simple techniques are used for sorting	The techniques are quite complex.
(5) It's various techniques are - quick sort heap sort, bubble sort, selection sort, Insertion sort, merge sort, radix sort, shell sort, etc.	Various Techniques are - Polyphase merge sort, balanced sort, K-way merge sort, etc

NOTE -

Complexity of Bubble Sort :

Best Case, Worst Case, Average Case : $O(n^2)$

Date .....

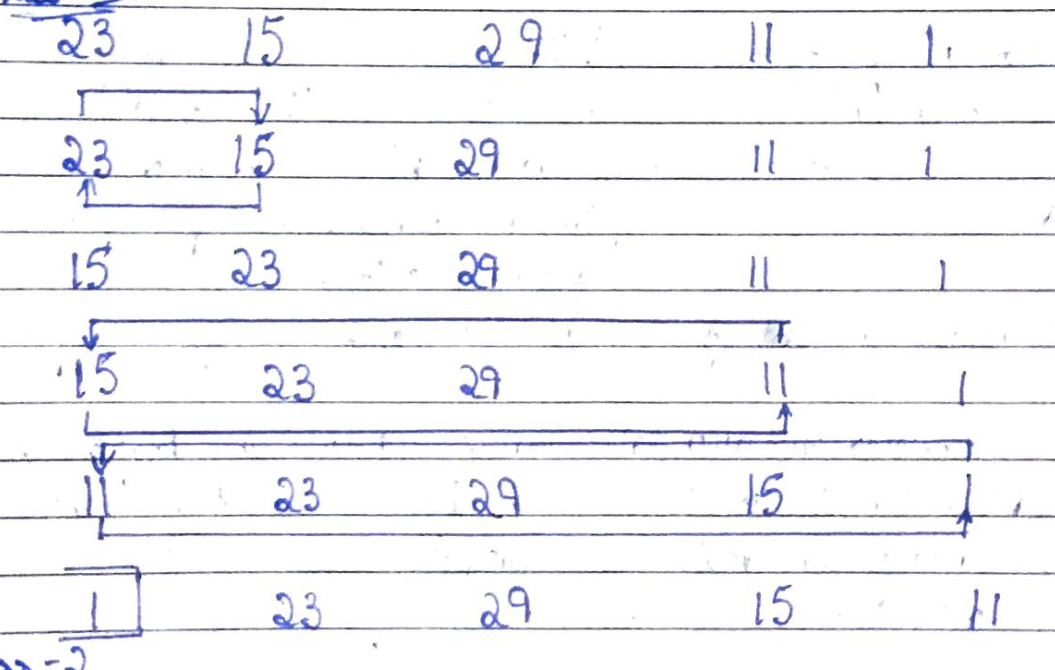
Selection Sort

(1) To sort data in ascending order, the 0^{th} element is compared with all other elements if 0^{th} element is found to be greater than compared elements, then they are interchanged.

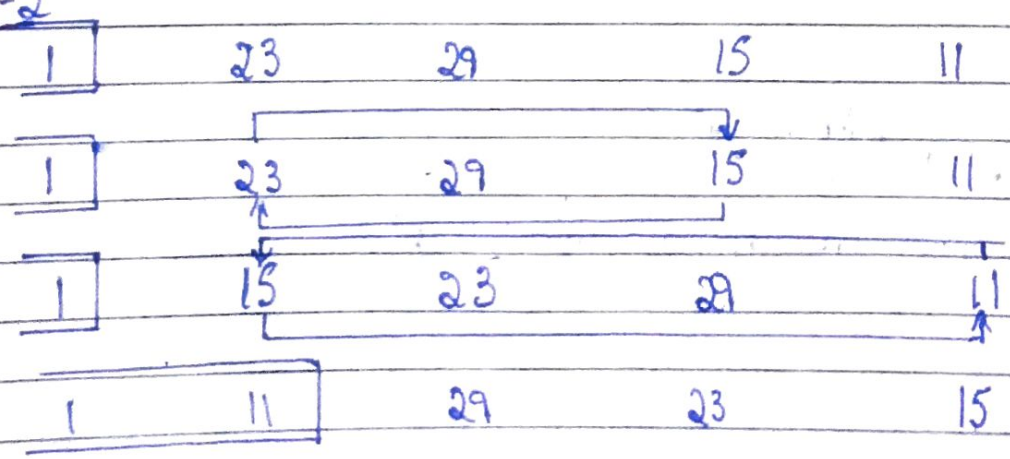
(2) We find the smallest element first.

(3) If there are n elements then after n-1 passes the array is sorted.

Pass - 1



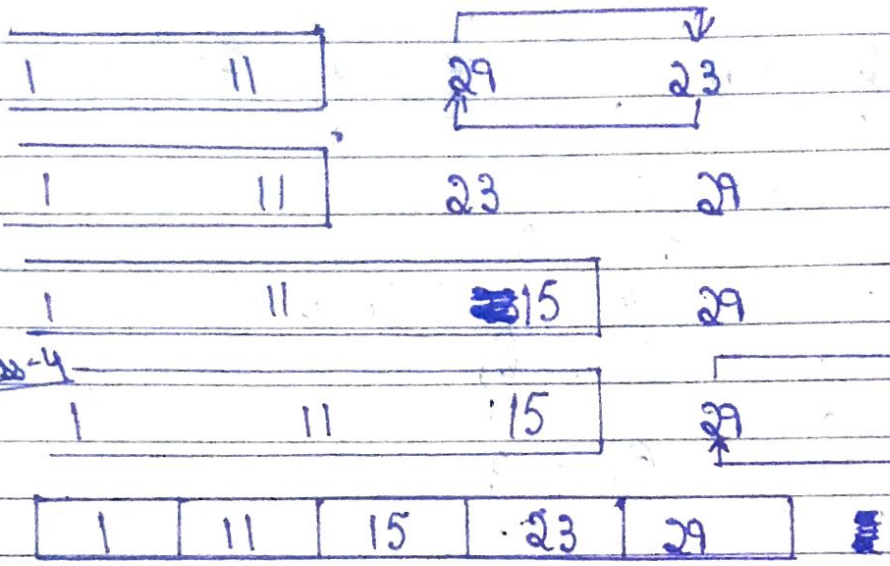
Pass - 2



Spiral

Date .....

Pass 3



Best Case, Worst Case, Average Case = $O(n^2)$

* Quick Sort / Partition Exchange Sort

1. It used as an application of 'Stack'
2. It follows the strategy of 'divide & conquer'.
3. In this approach, numbers are divided & then sub-divided until it's not possible to divide them further.
4. It follows recursion procedure.
5. Here, first element is generally fixed & is called 'Pivot Element'.

Ex-

10 1 9 11 46 20 15 0 72 2

Pass 1 scan list from last →

(10) 1 9 11 46 20 15 0 72 (2)

← R to L

(2) 1 9 11 46 20 15 0 72 (10)

→ L to R

2 1 9 (10) 46 20 15 0 72 (11)

← R to L

Spiral

Date .....

2 1 9 (0) 46 20 15 (10) 72 11

L to R →

2 1 9 0 (10) 20 15 46 72 11

Pivot

Pass-2

(2) 1 9 0

← R to L

(0) 1 9 (2)

→ L to R

0 1 (2) 9

Pivot

Pass-3

(20) 15 46 72 11

← R to L

(11) 15 46 72 (20)

→ L to R

11 15 (20) 72 46

Pass-4

(0) 1

→ L to R ← R to L

Pass-5

9 | 0 1 2 9 |

Pass-6 (11) 15 | 11 15 |

Pass-7 (72) 46

← R to L

Pass-8 (46) 72 | 46 72 |

0	1	2	9	10	11	15	20	46	72

Spiral

Date .....

Complexity

Best case - $O(n \log n)$

Avg case - $O(n \log n)$

Worst case -

$$n + (n-1) + \cdots + 2 + 1$$
$$= \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$
$$= O(n^2)$$

Insertion Sort

In Insertion Sort first element of list should be smallest possible element.

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
$-\infty$	13	15	7	6	18	3

Pass-1

A[0]	A[1]
$-\infty$	13

Pass-2

A[0]	A[1]	A[2]
$-\infty$	13	15

Pass-3

$-\infty$	13	15 ←	7	Store 7 in Temp.
$-\infty$	13	7	15	
$-\infty$	7	13	15	

Date .....

Pass -4

$-\infty$	7	13	15	← 6	Store 6 in temp variable
$-\infty$	7	13	6	15	
$-\infty$	7	6	13	15	
$-\infty$	6	7	13	15	

Pass -5

$-\infty$	6	7	13	15	18	
$-\infty$	6	7	13	15	18	← 3 Store 3 in temp variable
$-\infty$	6	7	13	15	3	18
$-\infty$	6	7	13	3	15	18
$-\infty$	6	7	3	13	15	18
$-\infty$	6	3	7	13	15	18
$-\infty$	3	6	7	13	15	18

$-\infty$	3	6	7	13	15	18

Complexity → Worst case – $O(n^2)$

Average case – $O(n^2)$

Spiral

Date .....

Merge Sort

Merging is the process of combining 2 sorted list into single sorted list.

To perform merge sort, both the sorted lists are compared & smaller of both the elements is stored in 3rd Array. Sorting completes when all the elements from both the list are placed in 3rd list.

In Merging, 2 list must be sorted using any algorithm.

Ex- a (i) List 1 : 3 5 7 12 20

b (i) List 2 : 2 6 9 10 11 25

(ii) List 3 : 2 3 5 6 7 10 11 12 20 25

if b[j] < a[i], add it to list 3 & j++

else b[j] > a[i], add a[i] to list 3 & i++

Merge Sort Ex- O(n log n)

Pass-1 : 10 15 9 7 20 25 18

Sorting

Pass-2 : 10 15 7 9 20 25 18

Merging Merging

Pass-3 : 7 9 10 15 18 20 25

Merging

Pass-4 :

7	9	10	15	18	20	25

Date .....

Radix Sort / Bucket Sort O(n log n)

In Pass-1, cards are sorted according to 1's digit
In Pass-2, cards are sorted according to 10's digit &
In Pass-3, cards are sorted according to 100's digit

Ex

750 356 301 428 650 250 215

Pass-1

I/P	0	1	2	3	4	5	6	7	8	9
750	750	301				215	356		428	
356	650									
301	250									
428										
650										
250										
215										

Pass-3

I/P	0	1	2	3	4	5	6	7	8	9
750	301	215	428			750				
650						650				
250						250				
301						356				
215										
356										
428										

Date .....

Pass-3

IP	0	1	2	3	4	5	6	7	8	9
301			215	301	428		650	750		
215			250	356						
428										
750										
650										
250										
356										

After Sorting :-

215
250
301
356
428
650
750

Spiral

Date .....

Shell Sort / Dimmishing-Increment Sort / Comb Sort - $O(n \log n^2)$

17 3 2 5 10 15 6 9

Pass-1 Gap = 3

Sublist 1: 17 5 6
Sublist 2: 3 10 9
Sublist 3: 2 15

- We create sublists of elements that have gap of '3'. Here, we are not creating separate list physically but logically.
- In this, gap power of 2 must be avoided and prime numbers should be taken.

We sort the sublist seperately using Insertion Sort:

Sublist 1: 5 6 17
Sublist 2: 3 9 10
Sublist 3: 2 15

(17) 3 2 (5) 10 15 (6) 9 (given)
We swap
5 (3) 2 6 (10) 15 17 (9)
we swap
5 3 (2) 6 9 (15) 17 10

The list is ordered (2 & 15)

Date .....

5 3 2 6 9 15 17 10

Pass 2 Gap = 3 - 1 = 2

Sublist 1: 5 2 9 17
Sublist 2: 3 6 15 10

We sort the sublists separately using Insertion Sort

Sublist 1: 2 5 9 17
Sublist 2: 3 6 10 15

(5) 3 (2) 6 (9) 15 (17) 10

we swap

2 (3) 5 (6) 9 (15) 17 (10)

we swap

2	3	5	6	9	10	17	15

Pass 3 Gap = 2 - 1 = 1

Sublist : 2 3 5 6 9 10 17 15

We sort the sublist using Insertion Sort:

2 3 5 6 9 10 15 17

2	3	5	6	9	10	15	17

↓

Sorted List

Spiral

Date .....

External Sorting

- Polyphase Merge
- Balanced Merge
- K-way Merge

① K-way Merge Sort :-

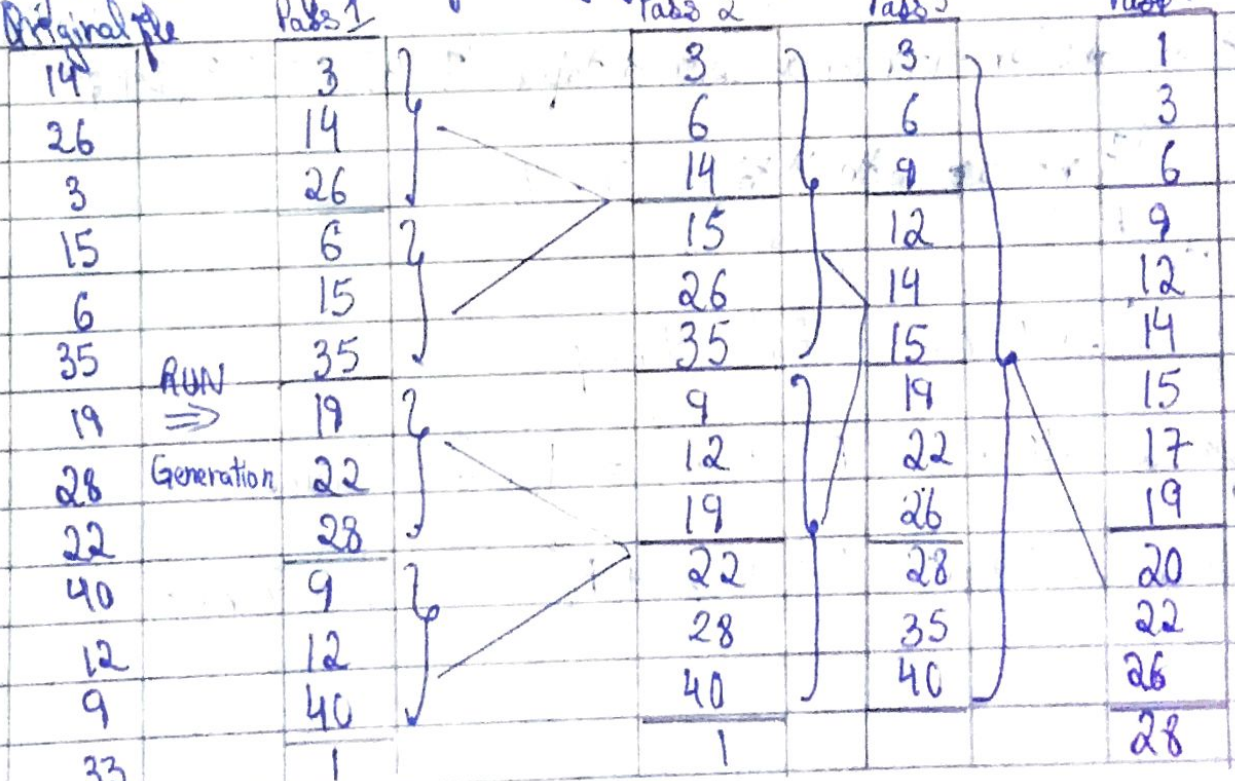
→ Let's conside a buffer size of 3. The first pass creates runs of length 3. For K-way Merge (K+1) buffers are required.

→ In this, 2 I/P & 1 O/P Buffers is required.

→ Strategy for K-way Merge Sort :-

No of runs = r

Merge r runs taking K at a time using merging algorithm which is generalization of 2-way merging technique.



Spiral

33		20		17			33
20		33		20			35
1		17		33			37
17		37		37			40
37		X		X			X

② Polyphase Merge - Sort :

→ This is one of the best known method for external sorting.

→ In this, tapes are required, It's basic strategy of this is to distribute initial runs in multiple phases.

→ Fibonacci Series is used to determine no. of runs on each tape.
Series is -

$$F^{p}_{s-1} + F^{p}_{s-2} + F^{p}_{s-3} - - - - - F^{p}_{s-k}$$

Let us consider 17 records with 4 tapes ($T=4$), $P=T-1=3$, Let $s=5$

17 records with 4 tapes ($T=4$)
$P = T-1 = 3$, $S = 5$

Tape 1 will receive $F_4^3 = 4$ runs

Tape 2 will receive $F_4^3 + F_3^3 = 4+2 = 6$ runs

Tape 3 will receive $F_4^3 + F_3^3 + F_2^3 = 6+1 = 7$ runs

Step 1

1	2	3	4
14			
26			
3			
15			
6			
25			
9			
28			
22			
40			
12			
9			
33			
20			
1			
17			
37			

Step 2

1	2	3	4
14	6	12	
26	35	9	
3	19	33	
15	28	20	
	22	1	
	40	17	
		37	

Step 3

1	2	3	4
26	35	9	6
3	19	33	12
15	28	20	14
	22	1	
	40	17	
		37	

Step 4

1	2	3	4
3	14	33	6
15	28	20	12
	22	1	14
	40	17	9
		37	26
			35

Spiral

Step 5

1	2	3	4
15	28	20	6
	22	1	12
	40	17	14
		37	9
			26
			35
			3
			19
			33

Step 6 Date .....

1	2	3	4
	22	1	6
	40	17	12
		37	14
			9
			26
			35
			3
			19
			33
			15
			20
			28

1	2	3	4
1	40	17	9
6		37	26
12			35
14			3
22			19
			33
			15
			20
			28

1	2	3	4
1		37	3
6			19
12			33
14			15
22			20
9			28
17			
26			
35			
40			

Step 7

Step 8

Spiral

Step 9

1	2	3	4
9	1		15
17	3		20
26	6		28
35	12		
40	14		
	A		
	22		
	33		
	37		

Step 10 Date .....

1	2	3	4
		1	
		3	
		6	
		9	
		12	
		14	
		15	
		17	
		19	
		20	
		22	
		26	
		28	
		35	
		35	
		37	
		40	

Polyphase merge sort is good for 6 or less no of tapes, but its performance is ↓ for large no. of tapes.
Not good

③ Balanced Merge Sort :-

It is the simpliest & slowest ~~merging~~ tape merging Algorithm.

→

Date .....

Step-1

Tape	Containing runs				
T_1	R_1	R_3	R_5	R_7	R_9
T_2	R_2	R_4	R_6	R_8	R_{10}
T_3					
T_4					

Step-2

Tape	Containing runs		
T_1			
T_2			
T_3	$R_1 - R_2$	$R_5 - R_6$	$R_9 - R_{10}$
T_4	$R_3 - R_4$	$R_7 - R_8$	

Step-3

Tape	Containing runs	
T_1	$R_1 - R_4$	$R_9 - R_{10}$
T_2	$R_5 - R_8$	
T_3		
T_4		

Step-4

Tape	Containing runs
T_1	
T_2	
T_3	$R_1 - R_8$
T_4	$R_9 - R_{10}$

Spiral

Date .....

Step-5

Tape	Containing runs
T_1	R_1 - R_{10}
T_2	
T_3	
T_4	

UNIT-4 Spanning Tree [Skeleton of Graph]

→ It is an undirected tree consists of only those edges which are necessary to connect all the nodes.

→ Spanning tree has the property that to any pair of nodes, there exists only path b/w them & insertion of any edge to spanning tree forms a unique cycle.

→ A Tree T is said to be spanning tree of connected graph if T is a subgraph of G and T contains all the vertices of G".

→ It's also called 'Skeleton of Graph'.

→ In real life application, weights are present on edges, then aim is to generate a spanning tree having minimum cost.

→ Cost of spanning tree is the sum of the cost of all the edges in the tree.

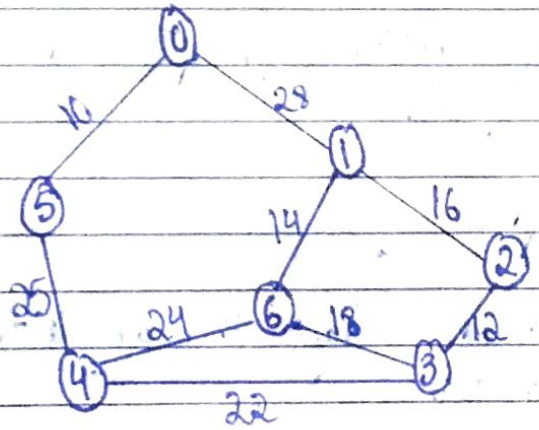
Minimum Cost
- Prim's Algorithm
- Kruskal's Algorithm

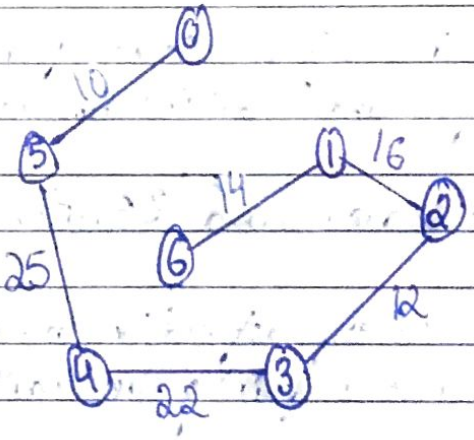
Spiral

Date .....

Kruskal's Algorithm

```
$-1 = T { }
$-2 = While (T contains less than (n-1) edges
                && E is not empty)
$-3: Choose a least cost edge (v,w) from E
$-4: Delete (v,w) from E
$-5: If [(v,w) doesn't create a cycle in T] then
$-6: Add (v,w) to T
$-7: Else
        discard (v,w)
$-8: [end of if]
     [end of while]
$-9: If (T contains lower than (n-1) edges)
        print ("No spanning Tree")
```





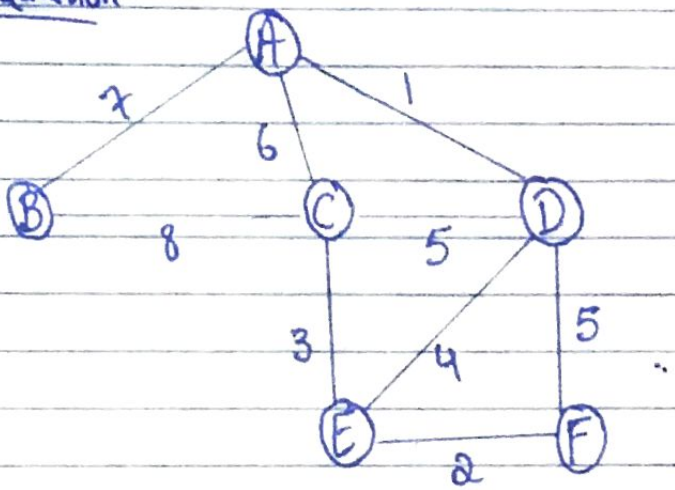
Edge	Weights	Result
(0,5)	10	Added to tree
(2,3)	12	Added to tree
(1,6)	14	Added to tree
(1,2)	16	Added to tree
(6,3)	18	Discarded from tree

Spiral

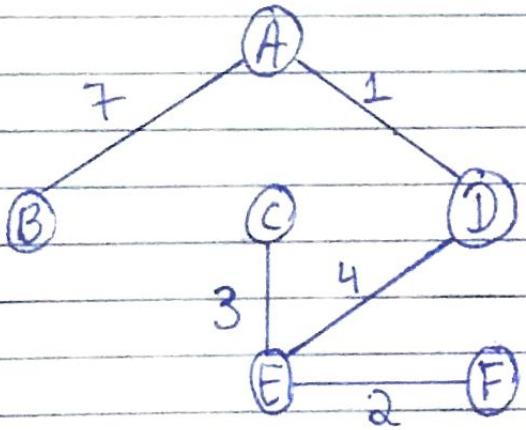
Date .....

(3,4)	22	Added to tree
(4,6)	24	discarded from tree
(5,4)	25	added to tree
(0,1)	28	discarded from tree

Imp. Question



Ans → 1, 2, 3, 4, 7



Final Spanning Tree

Date 21/11

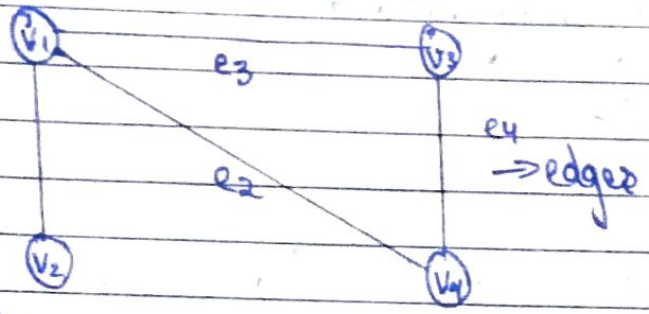
UNIT - 4 Graphs

Vertex

$G = (V, E)$ ← Edges

$V = \{ v_1, v_2, \ldots v_n \}$

$E = \{ e_1, e_2, \ldots e_n \}$



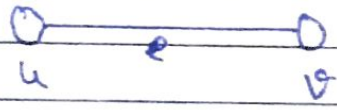
→ edges

Example -

$e_1 [v_1, v_2]$ [Vertex | Nodes | Points], $e_3 [v_1, v_3]$, $e_2 [v_1, v_4]$, $e_4 [v_3, v_4]$

End

If there is an edge, where u & v are called points of e / Adjacent Points / neighbors.



Degree of Node - deg(u) - No. of edges connecting u

Example - deg $(v_1) = 3$
deg $(v_2) = 1$
deg $(v_3) = 2$
deg $(v_4) = 4$

⇒ If deg (any node) = 0, then it is called an Isolated Node.
Ex- deg $(v_5) = 0$

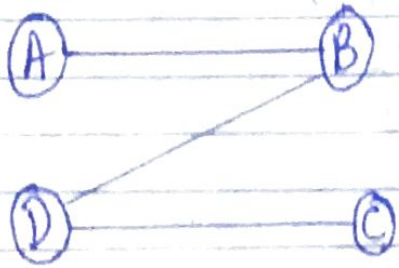
Spiral

Date .....

Path -

It is a sequence of distinct vertices each adjacent to next.

Example -



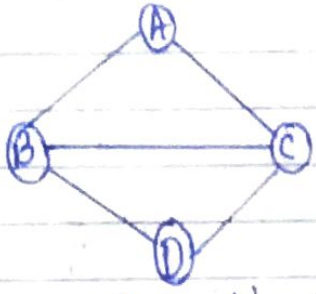
{ Acyclic Graph }

Length of path = No. of edges
Here, length of path = 3

Cycle -

A path from node to itself is called cycle.

Example -



{ Cyclic Graph }

Cycle - ① ABCA
② ABDCA

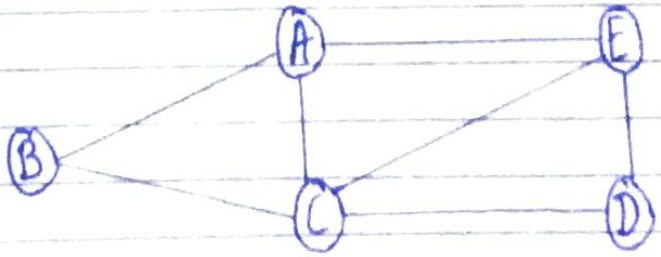
Cycle, then it's called.

If a graph contains a cyclic graph or otherwise Acyclic graph.

Connected Graph -

A graph is called Connected if there is a path from any vertex to any other vertex.

Example -

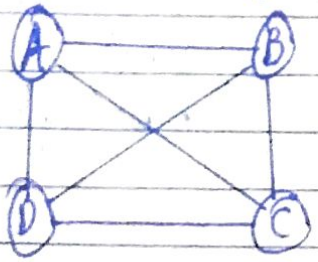


Date .....

Completed Graph -

A graph is called complete if every node in G is adjacent to every other node in graph

Example -

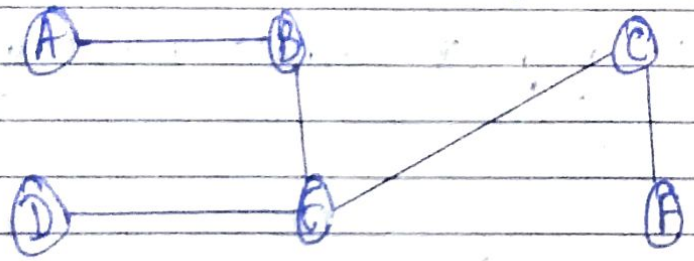


A complete graph with n node has $\frac{n(n-1)}{2}$ edges

Ex - $\frac{4(4-1)}{2} = \frac{4(3)}{2} = 6$, n nodes → $\frac{n(n-1)}{2}$ edges

Tree Graph / Free Graph / Tree

A Connected Graph with any cycle is called Tree Graph.



A Tree graph with m nodes has m-1 edges.

Date .....

Labelled Graph -

If every edge is assigned a label, then it is called a labelled graph.

Weighted Graph -

If every edge is assigned a weight, then is called a Weighted graph.

Multi-Graph -

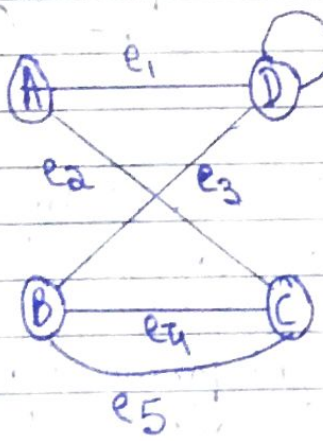
It contains multiple edges & loops. It is also called Parallel edges. They are called Distinct edges if they connect the same end points.

Multiple edges / Parallel edges -
Distinct edges e and e' are multiple edges if they connect the same end points. If, $e[u, v]$ then $e' = [u, v]$

Loop: An edge e is called Loop if it has identical end points.

Ex - $e[u, u]$

Example -



Multiple Edges - $e_4[B, C]$, $e_5[B, C]$

Loops - $e_6[D, D]$

Date .....

Directed Graph / Diagraph / Graph

A Directed graph is similar to multigraph except that each edge is assigned a direction.

→ Outdegree → Out deg (u) : It is No of edges beginning at u

→ Indegree → In deg (u) - No of edges ending at u.

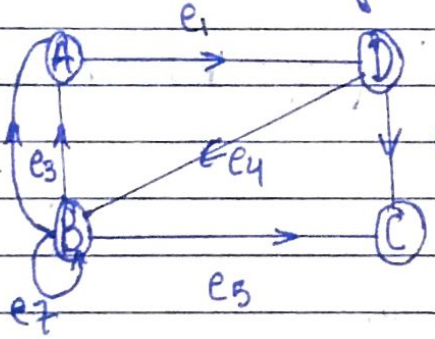
Source -

Source is a node having +ve Out-degree but Zero In-degree

Sink -

Sink is a node with Zero Outdegree but +ve Indegree

Example -



Out(A) - 4, In(A) - 2
Out(B) - , In(B) - 2
Out(C) - 0, In(C) - 2
Out(D) - 2, In(D) - 1

Multiple edges = e2, e3
Nodes = 4
Edges = 7
Loop = e7
Sink → C

- C is sink node as indegree of it is 2 & out degree is zero, as there is no path from C to any other node. So, C is not strongly connected.
- A Diagraph is called Simple if it has no parallel edges.
- A Simple Graph may have loops but it can't have more than one loop at a given node.

Spiral

Date 22/11....

Graph Representation -

- Sequential Rep.
- Linked list Rep.

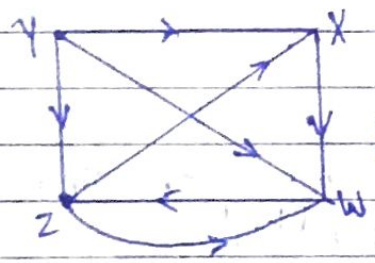
Sequential Graph Representation → Adjacency Matrix
→ Path Matrix

Adjacency Matrix → n nodes.

A = a(ij) is n x n matrix

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \text{ or if that is an edge } (v_i, v_j) \\ 0 & \text{otherwise} \end{cases}$$

Such a matrix A which has only 0 & 1 is called Bit/Boolean Matrix.



$$A = \begin{array}{c|cccc} & x & y & z & w \\ x & 0 & 0 & 0 & 1 \\ y & 1 & 0 & 1 & 1 \\ z & 1 & 0 & 0 & 1 \\ w & 0 & 0 & 1 & 0 \end{array}$$

$B_r = A + A^2 + A^3 + \dots + A^r$

$B_4 = A + A^2 + A^3 + A^4$

$$A^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad A^3 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad A^4 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Date .....

$$B_4 = \begin{bmatrix} 1 & 0 & 2 & 3 \\ 5 & 0 & 6 & 8 \\ 3 & 0 & 3 & 5 \\ 2 & 0 & 3 & 3 \end{bmatrix}$$

Path Matrix ⟶ m nodes / Reachability matrix

P = p[i,j] is m × m matrix

$$p_{ij} = \begin{cases} 1 & \text{if there is a path from } v_i \text{ & } v_j \\ 0 & \text{Otherwise} \end{cases}$$

Replace numericals with 1 and 0 as it is

$$P = \begin{array}{c} \\ x \\ y \\ z \\ w \end{array} \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

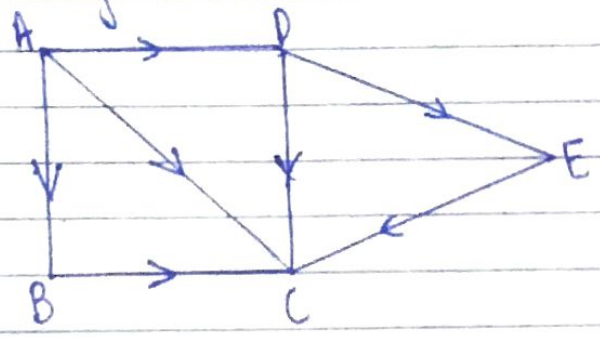
→ Since y = 0 then y is not reachable from any of the other nodes ⟹ The Graph is not strongly connected.

→ If a graph is strongly connected then matrix P of graph has no 0 entries.

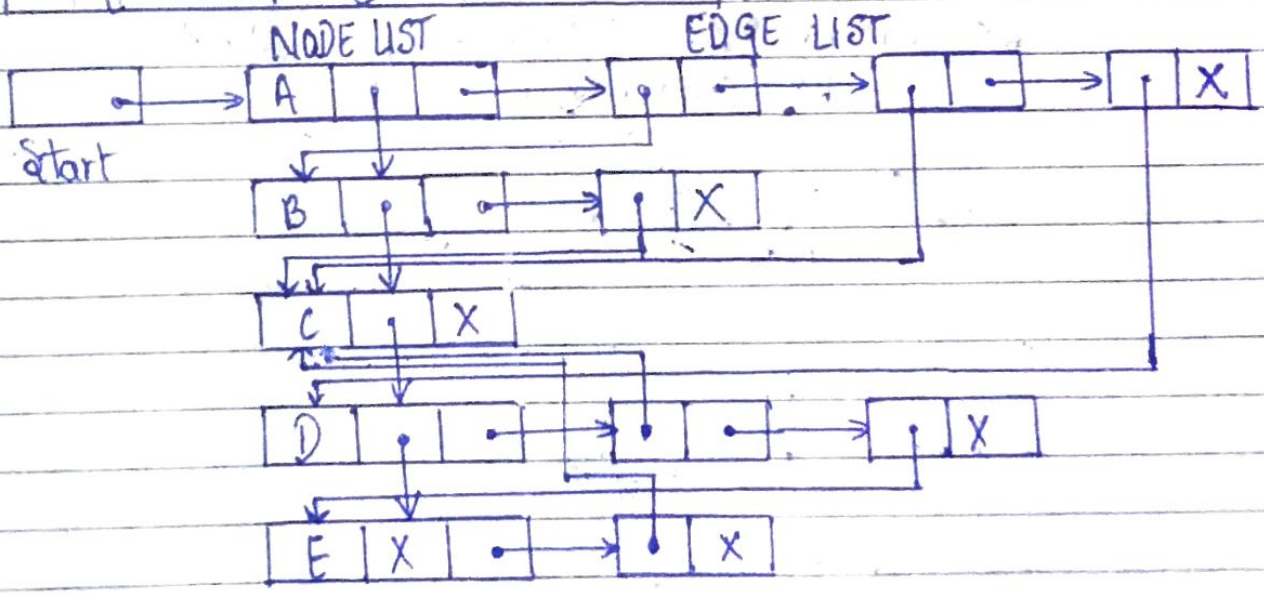
Date .....

LinKed list Representation

It maybe be difficult insert or delete nodes in Sequential Graph as the size of A Matrix changes & there maybe many changes in matrix.



Node	Adjacency List
A	B, C, D
B	C
C	—
D	C, E
E	C



Spiral

Date .....

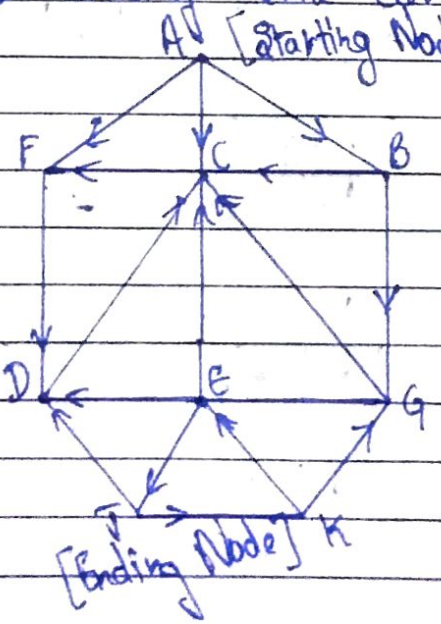
Graph Traversal

- BFS [Breadth First Search]
- DFS [Depth First Search]

Breadth First Search

Status - 1 - Ready State
Status - 2 - Waiting State
Status - - Processed State

Algorithm - ① Initialize all nodes to ready state. (Status - 1)
② Put the starting node in queue and change its status to waiting state. (Status - 2)
③ Repeat step 4 & 5 until Queue is Empty.
④ Remove the front node N of queue. Process N and change status of N to processed state. (Status - 3)
⑤ Add to the rear of queue all neighbors of N that are in ready state (Status - 1) & change their status to waiting state (Status - 2)
⑥ Exit



Spiral

Date .....

Adjacency List

A: F, C, B
B: G, C
C: F
D: C
E: D, C, J
F: D
G: C, F
J: D, K
K: E, G

① Initially add A to queue and add NULL to Orig
FRONT - 1 Queue: A
REAR - 1 Orig: ϕ

② Remove A from queue and add to queue the neighbors of A.
FRONT - 2 Queue: A [F C B]
REAR - 4 Orig: ϕ ~~A A A~~

③ Remove F from Queue and add to queue the neighbors of F
FRONT - 3 Queue: A F C B D
REAR - 5 Orig: ϕ A A A F

④ Remove C from queue and add to queue the neighbors of C
FRONT - 4 Queue: A F C B D
REAR - 5 Orig: ϕ A A A F

⑤ Remove B from queue and add to queue all the neighbors of B
FRONT - 5 Queue: A F C B D G
REAR - 6 Orig: ϕ A A A F B

Spiral

Date .....

⑥ Remove D from queue and add to queue the neighbors of D.

FRONT - 6 Queue : A F C B D G
REAR - 6 Orig : ϕ A A A F B

⑦ Remove G from queue and add to queue the neighbors of G.

FRONT 7 Queue: A F C B D G E
REAR 7 Orig: ϕ A A A F B G

⑧ Remove E from queue and add to queue the neighbors of E.

FRONT-8 Queue : A F C B D G E J
REAR-8 Orig: ϕ A A A F B G E

J ← E ← G ← B ← ~~A~~ ← A

J ← E ← G ← B ← A

Depth First Search

Status - 1 Ready State
Status - 2 Waiting State
Status - 3. Processed State

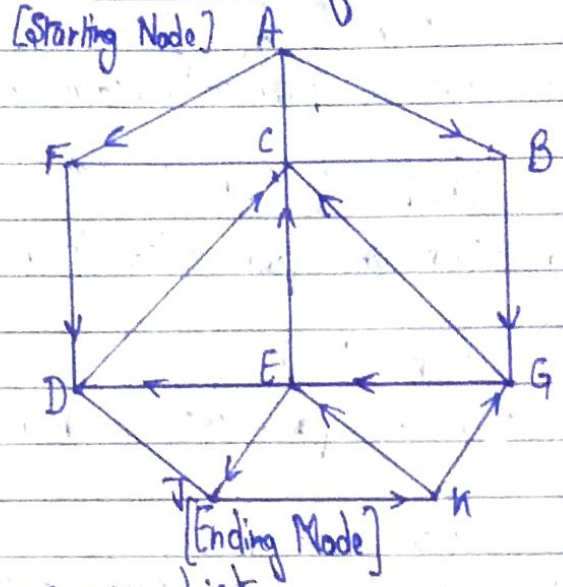
① Initialize all nodes to ready state (Status - 1)
② Push the starting node onto stack and change its status to waiting state (Status - 2)
③ Repeat steps 4 & 5 until Stack is empty.
④ Pop and Print top node of stack, process N and change the status of N to processed state (Status - 3)
⑤ Push onto stack all neighbors of N that are in ready state (Status - 1) and change their status to waiting state (Status - 2).
⑥ Exit

Spiral

Date .....

Find all nodes reachable from J

J, K, G, C, F, E, D



Adjacency List

A : F, C, B
B : G, C
C : F
D : C
E : D, C, J
F : D
G : C, F
J : D, K
K : E, G

① Push J, starting node of stack

Stack: J

② Pop and Print top node(J) and push onto stack all neighbors of J.

Print : J Stack : D, K

③ Pop and print top node(K) onto stack and push onto stack all neighbors of K.

Print : K Stack : D E G

Spiral

Date .....

④ Pop and print Top node G onto stack and push onto stack all neighbors of G.

Print : G	Stack : D, E, C

⑤ Pop and print Top node C onto stack and push onto stack all neighbors of C

Print : C	Stack : D, E, F

⑥ Pop and print Top node C onto stack and push onto stack all neighbors of F.

Print : F	Stack : D, E

⑦ Pop and print Top node E onto stack and push onto stack all neighbors of E.

Print : E	Stack : D

⑧ Pop and print Top node D onto stack and push onto stack all neighbors of D.

Print : D	Stack : Empty

J, K, G, C, F, E, D

Important Difference b/w questions

1. Diff b/w array and linked list
2. Stack and Queue
3. BFS and DFS
4. B^+ tree and B^* tree

Spiral

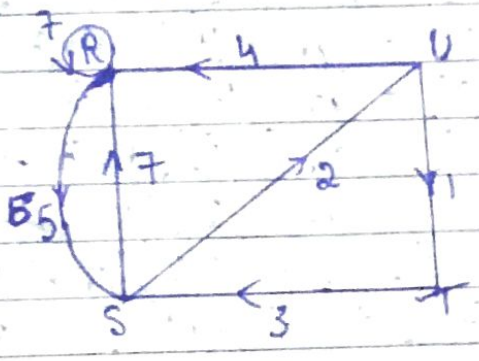
Date .....

Shortest Path b/w all pair of Vertices

Floyd — Warshall's Algorithm

$$W_{ij} = \begin{cases} w(e) & \text{if there is an edge from } v_i \text{ to } v_j \\ 0 & \text{Otherwise} \end{cases}$$

$$Q_k[i,j] = Min\left(Q_{k-1}[i,j],\ Q_{k-1}[i,k],\ Q_{k-1}[k,j]\right)$$



$$W = \begin{array}{c} \\ R \\ S \\ T \\ U \end{array} \begin{array}{c} \begin{array}{cccc} R & S & T & U \end{array} \\ \begin{bmatrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Algorithm — m is no. of nodes

① Repeat for $I = 1, J = 1, 2 \dots M$
  If $[W[I,J] = 0$ then set
    $Q[I,J] =$ Infinity
  else set $Q[I,J] = W^0[I,J]$

② Repeat step 3 & 4 For $K = 1, 2 \dots M$
③ Repeat step 4 For $I = 1, 2, \dots M$
④ Repeat For $J = 1\ 2 \dots M$
  Set $Q[I,J] = Min\left(Q[I,J], Q[I,K] + Q[K,J]\right)$
End Loop
End of step 3 Loop
End of step 2 loop

Date .....

Q_0 =

	R	S	T	U
R	7	5	∞	∞
S	7	∞	∞	2
T	∞	3	∞	∞
U	4	∞	1	∞

$K = 1$ to 4, $I = 1$ to 4, $J = 1$ to 4

$K = 1$ $\quad I = 1$ to 4, $\quad J = 1$ to 4

$Q_1(1,1) = \min(Q_0[1,1], Q_0[1,1] + Q_0[1,1]) = \min(7, 14) = 7$

$Q_1(1,2) = \min(Q_0[1,2], Q_0[1,1] + Q_0[1,2]) = \min(5, 7+5) = 5$

$Q_1(1,3) = \min(Q_0[1,3], Q_0[1,1] + Q_0[1,3]) = \min(\infty, 7+\infty) = \infty$

$Q_1(1,4) = \min(Q_0[1,4], Q_0[1,1] + Q_0[1,4]) = \min(\infty, 7+\infty) = \infty$

$Q_1(2,1) = \min(Q_0[2,1], Q_0[2,1] + Q_0[1,1]) = \min(7, 7+7) = 7$

$Q_1(2,2) = \min(Q_0[2,2], Q_0[2,1] + Q_0[1,2]) = \min(\infty, 7+5) = 12$

$Q_1(2,3) = \min(Q_0[2,3], Q_0[2,1] + Q_0[1,3]) = \min(\infty, 7+\infty) = \infty$

$Q_1(2,4) = \min(Q_0[2,4], Q_0[2,1] + Q_0[1,4]) = \min(2, 7+\infty) = 2$

$$Q_1 = \begin{pmatrix} 7 & 5 & \infty & \infty \\ 7 & (12) & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & (9) & 1 & \infty \end{pmatrix}$$

$K = 2$ $\quad I = 1$ to 4, $\quad I = 1$ to 4

$$Q_2 = \begin{pmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 9 & 1 & 11 \end{pmatrix}$$

Spiral

Date .....

$k=3$ $\quad Q_3 = \begin{pmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 4 & 1 & 6 \end{pmatrix}$

$k=4$ $\quad Q_4 = \begin{pmatrix} 7 & 5 & 8 & 7 \\ 7 & 11 & 3 & 2 \\ 9 & 3 & 6 & 5 \\ 4 & 4 & 1 & 6 \end{pmatrix}$

shortest path b/w all vertices $\Leftarrow$

Hashing | Hash Addressing

It is a technique which is independent of no. of records or no. of element.

$H : K \rightarrow L$

- Hash function
- Key
- Address at which Key is Placed

Hash Table

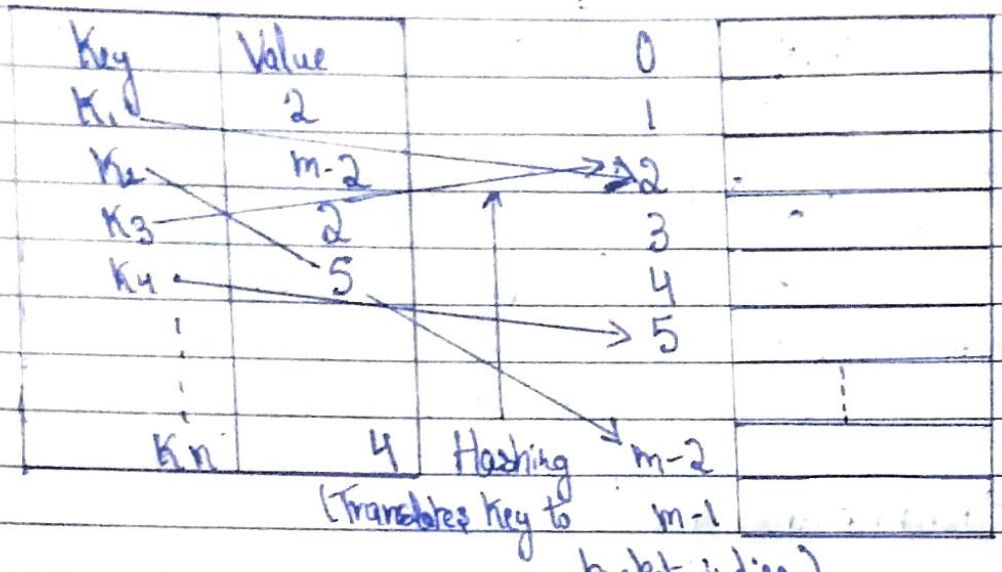
It is an array of m buckets together with a Hash function $H(K)$ which translates each Key K to bucket indices from 0 to m-1
A bucket has no. of slots & each slot can hold one record.

Spiral

Date .....

Hash Function

Hash Table



① Division Method -

$H(K) = K \pmod m$ or $H(K) = K \pmod m + 1$

② Mid-Square Method - $H(K) = l$

where l can be obtained by deleting digits from both ends of K^2.

③ Folding Method -

$H(K) = K_1 + K_2 + \dots + K_r$ where

$K_1, K_2 \dots K_r$ are the partitions of key K.

Q. Consider a company of 68 employees & each employee is assigned a unique four digit employee no. Suppose l consists of 100 two digit addresses starting from 00, 01 --- 99. Find the hash function for employees 3205, 7148 and 2345

Spiral

Date .....

Solution → Division Method - H(3205) = 3205 (mod 97) = 4
H(7148) = 7148 (mod 97) = 67
H(2345) = 2345 (mod 97) = 17
H(K) = K (mod m)

H(3205) = 4+1 = 5
H(7148) = 67+1 = 68
H(2345) = 17+1 = 18

Mid-Square Method -

K:	3205	7148	2345
K^2:	10272025	51093904	5499025
H(K) =	72	93	99

Folding Method -

H(3205) = 32+05 = 37
H(7148) = 71+48 = 119 ≃ 19
H(2345) = 23+45 = 68

Reversal -

H(3205) = 32+50 = 82
H(7148) = 71+84 = 155 ≃ 55
H(2345) = 23+54 = 77

Date .....

Overflow | Collision Resolution - Load factor (λ) = n/m

Load Factor (λ) - It is the ratio of no. n of keys in K (which is no. of records in F) to the ~~total~~ number m of hash addresses in L.

Types of Collision Resolution

① Linear Probing (open addressing) - $(h, h+1, h+2, h+3 \ldots)$

Main disadvantage of Linear Probing is that the records tends to cluster. Clustering increases the average search time for record. To resolve this clustering issue, different methods are used.

Quadratic Probing -

$\Rightarrow h, h+1^2, h+2^2 + h+3^2$

$\Rightarrow h, h+1, h+4, h+9$

② Double hashing $\{H(k) = h, H'(k) = h'\}$

$\Rightarrow h, h+h', h+2h', h+3h'$

Spiral