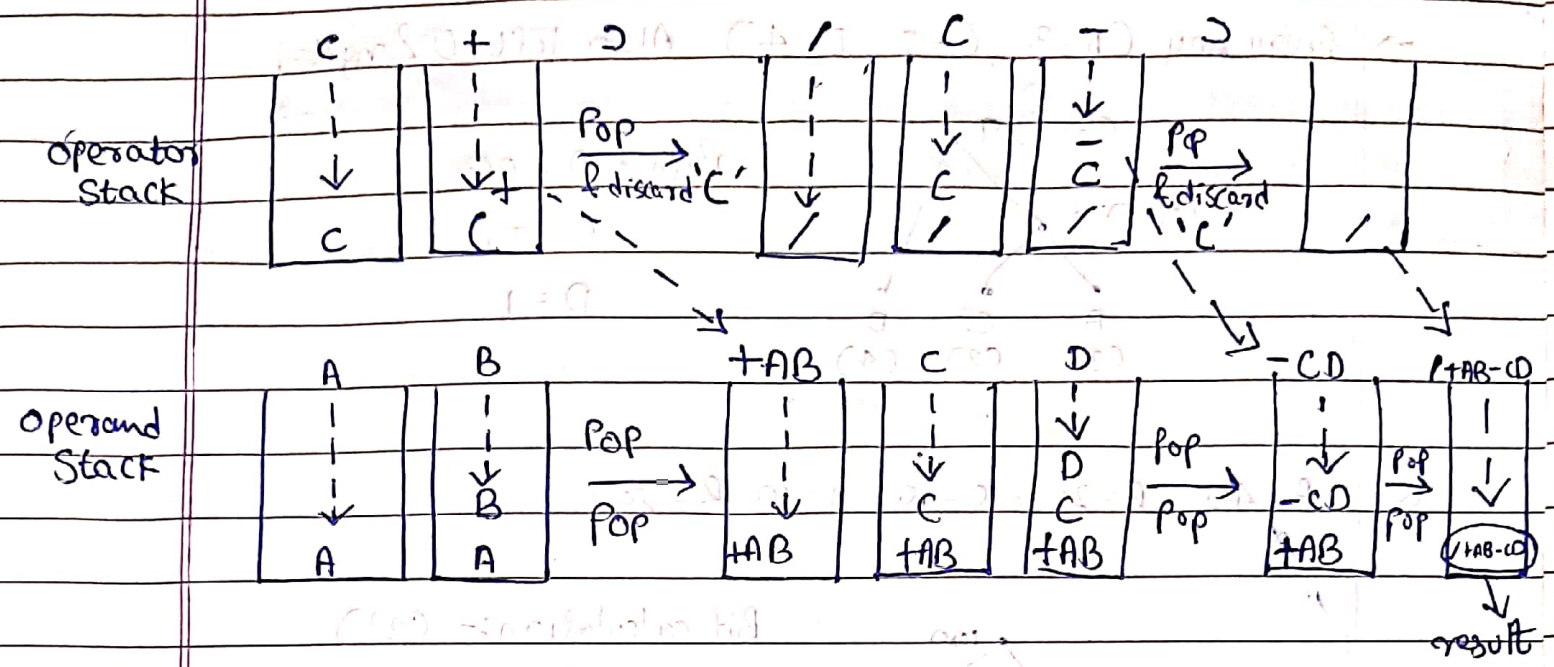
Data Structures

Infix → Prefix (A+B) / (C-D)



Trees

→ Recursive Functions

Preorder	PostOrder	Inorder

Input: x is root
if x ≠ null
then o/p key(x);
preorder(left(x));
preorder(right(x));
	I/p: x is root
	if x ≠ null
	then postorder(left(x));
	postorder(right(x));
	o/p key(x);
		I/p: x is root
		if x ≠ null
		then Inorder(left(x));
		o/p key(x);
		Inorder(right(x));

→ Non-Recursive Functions

Inorder

```
do{
 while(P!=null)
 { push(stack,P);
   P=P->left;
 }
 if(!empty(stack))
 { p=pop(stack);
   print(P->info);
   P=P->right;
 }
}while(!empty(stack)
   || P!=null)
```

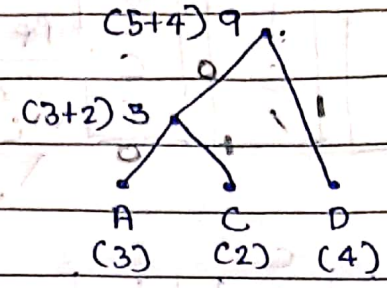
Preorder

```
do{
 while(P!=null)
 { print(P->info);
   push(stack,P);
   P=P->left;
 }
 if(!empty(stack))
 { P=pop(stack);
   P=P->right;
 }
}while(!empty(stack)
   || P!=null)
```

(*) Hoffman's Compression Algo

Q AACCDDADD

→ frequency (A=3, C=2, D=4) Also left=0 & right=1

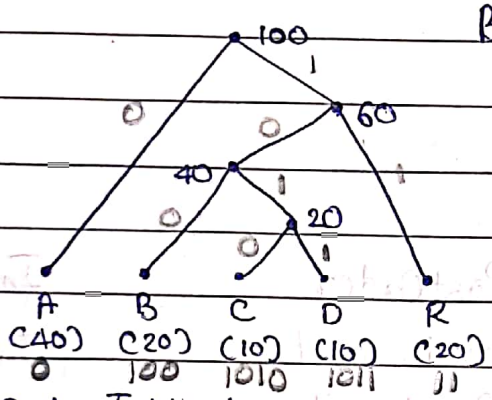


Path : A = 00

C = 01

D = 1

Q A=40 B=20 C=10 D=10 R=20



Bit calculation :- (2^n)

number of data = 5
(A, B, C, D, R)

$\therefore 2^1 = 2 >= 5$ ✗

$\therefore 2^3 = 8 >= 5$ ✓

Bits = n = 3

Data Table 1

letter	freq	bits	fre*bits
A	40	3	120
B	20	3	60
C	10	3	30
D	10	3	30
R	20	3	60
			300

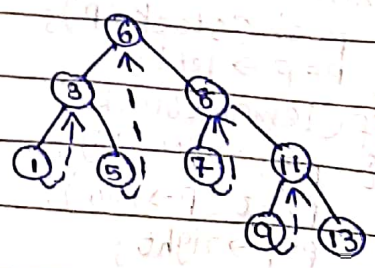
Data Table 2

letter	freq	path bits	freq*path
A	40	1	40
B	20	3	60
C	10	4	40
D	10	4	40
R	20	2	40
			220

Compression % = $\frac{\text{change}}{\text{original}} \times 100 = \frac{300-220}{300} \times 100 = 26.67\%$

(*) Threaded Tree

```
//Initialize ptr to the root
while(ptr!=null)
{ while(ptr->left!=null)
  { ptr = ptr->left; }
  print(ptr->info);
  while(ptr->right == thread)
  { ptr=ptr->right;
    print(ptr->info);
  }
  ptr = ptr->right;
}
```



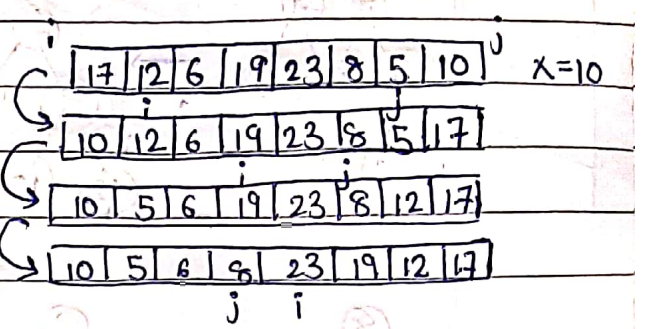
O/P :- 1 3 5 6 7 8 9 11 13

Page No.
Date

Linear Time Partitioning

→ Partition (A, p, r) // where r = last value in A[]

```
X = A[r]
i = p-1
j = r+1
while TRUE
    repeat j=j-1
    until A[j] ≤ X
    repeat i = i+1
    until A[i] > X
    if i<j   // i is left of j
     then exchange A[i] ↔ A[j]
    else return j
```

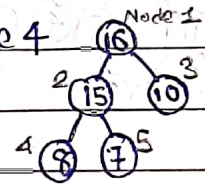


Heap Sort & Heapify Algo (Parent Node ≥ Child Node)

→ Left of node i = 2i Eg: Left of Node 2 = Node 4

→ Right of node i = 2i+1

→ Parent of node i = i/2



→ Heap Sort Algo :-

```
Parent(i)
    return (i/2)
left(i)
    return 2i
right(i)
    return 2i+1
```

Heap Property

A[parent(i)] ≥ A(i)

→ Heapify Algo :- (n = no of elements)

1) Heapify (A, i)
2) Left & Right sub-tree of i are heaps
3) Make sub-tree rooted at i a heap
4) l ← left l = 2i
5) r ← right r = 2i+1
6) if l ≤ n && A[l] > A[i]
   then largest ← l
7) else largest ← i
8) if r ≤ n && A[r] > A[largest]
   then largest ← r
9) if largest ≠ i then exchange A[i] & A[largest]
10) Heapify (A, largest)

Heap Sort (A)

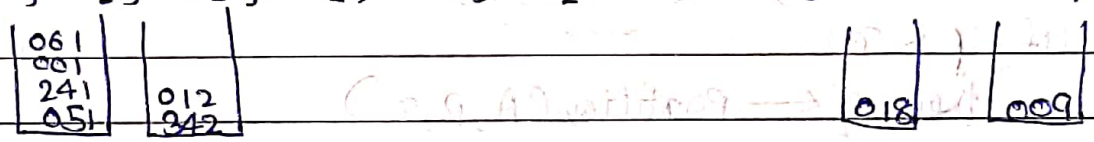
1) Build-heap (A)
2) for i ← n down to 2
3) exchange A[1] ↔ A[i]
4) n ← n-1
5) Heapify (A, 1)

⊛ Radix Sort & Radix Sort Reverse

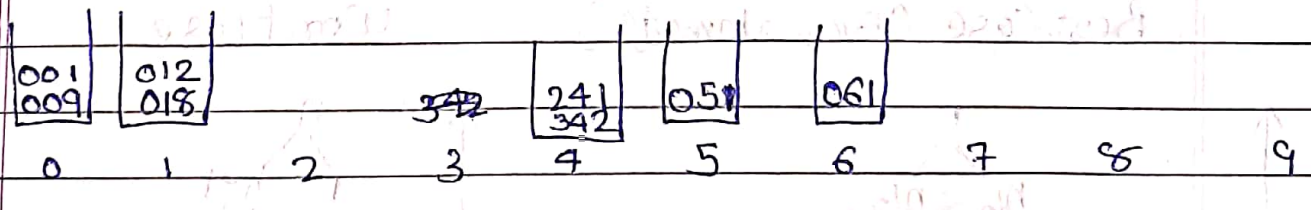
♡ 61, 12, 1, 342, 241, 51, 18, 9

Radix sort from left to right i.e 61 to 9 &
Radix sort reverse from right to left // gives ascending order

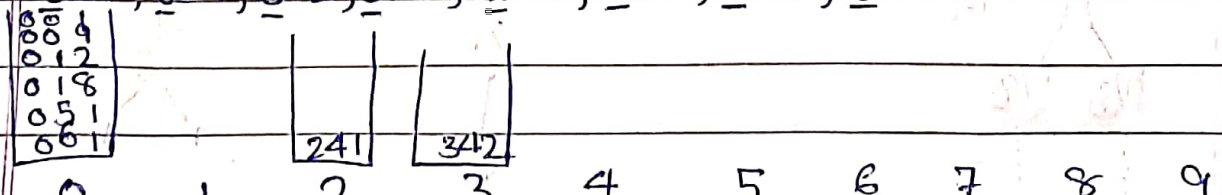
061, 012, 001, 342, 241, 051, 018, 009 // Unit's place



pop → 061, 001, 241, 051, 012, 342, 018, 009 // ten's place



pop → 001, 009, 012, 018, 241, 342, 051, 061 // Hundred's place

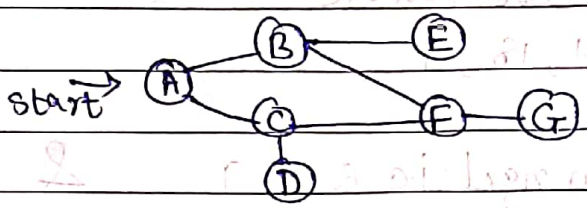


pop → 001, 009, 012, 018, 051, 061, 241, 342

Page No.
Date

BFS Algo

→ while queue Q not empty
  dequeue first vertex u from Q
  for each vertex directly reachable from u
  if v is unvisited
    enqueue v to Q
    mark v as visited



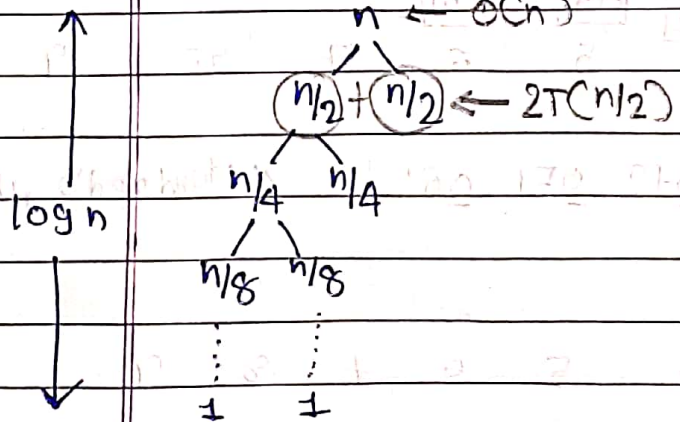
O/P :-

A B C E F D G

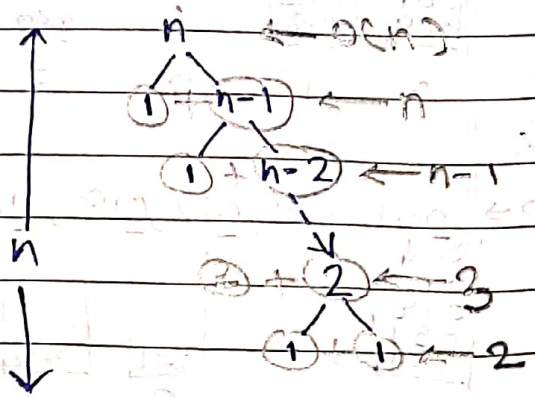
Quick Sort

→ Initial call QuickSort (A, p, r)
if p < r
  then q ← partition(A, p, r)
    QuickSort(A, p, q)
    QuickSort(A, q+1, r)

Best Case (Even elements)



Worst Case



Complexity = $O(n^2)$

$T(n) = 2T(n/2) + O(n)$

Page No.
Date

DFS Algo

// ~~Initially all vertices are unvisited~~

```
DFS (vertex v)
{   mark v as visited
    for each vertex v directly reachable from v
    if v is unvisited
        DFS (v)
}
```

Bubble Sort Algo

// Two pointers

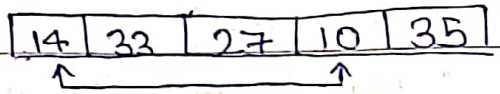
→	60	42	42	42	42	
→	42	→ 60	60	60	60	First pass complete
	75	→ 75	→ 75	75	75	
	83	83	→ 83	→ 83	27	
	27	27	27	→ 27	83	

Result after Second pass

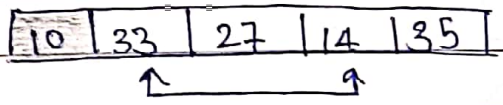
// Third pass	// Fourth Pass	
42	27	
27	42	
60	60	
75	75	Done
83	83	

Selection Sort

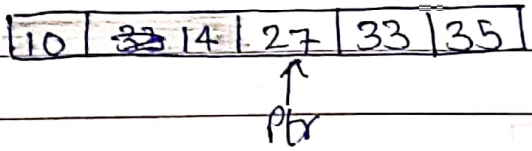
Q



// firstly scan all elements and exchange min val with first ele.

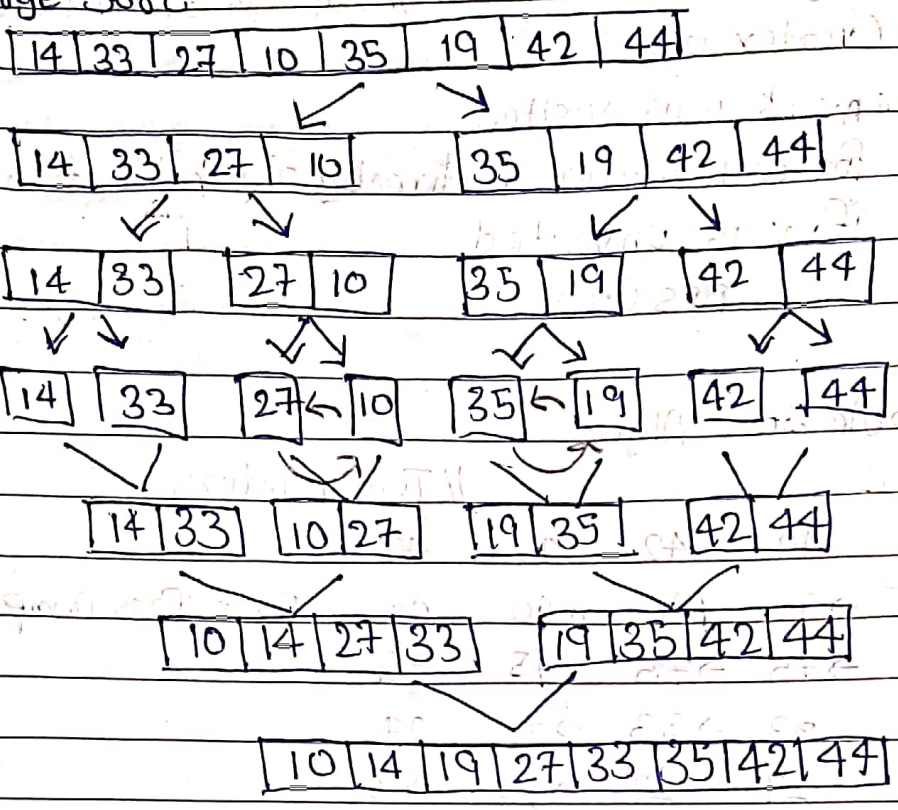


// ptr moves to second ele scan min value & exchange



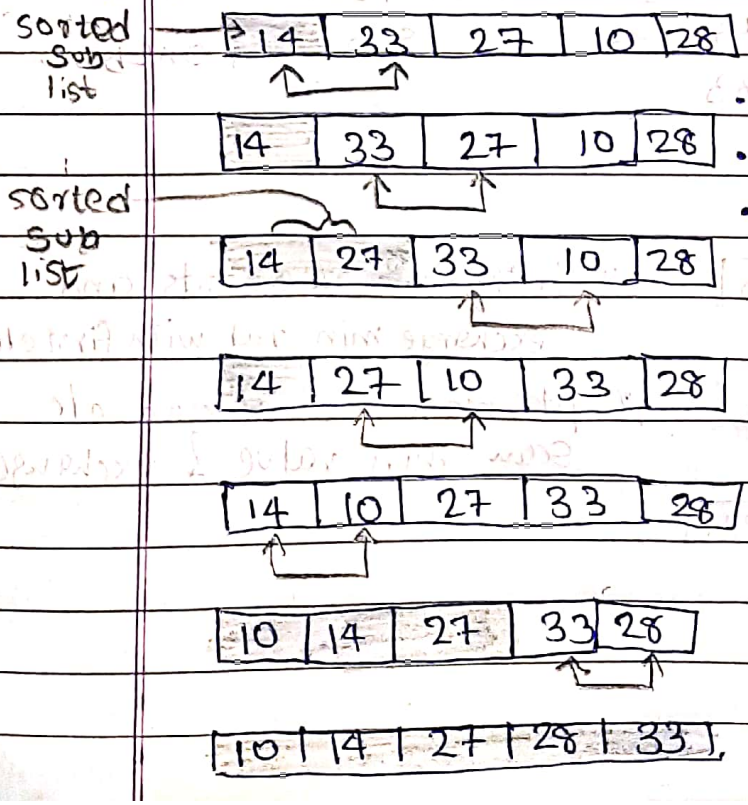
Page No.
Date

⊛ Merge Sort



Algo :- Divide list into 2 parts till it can no more be divided
Merge smaller lists into sorted order.

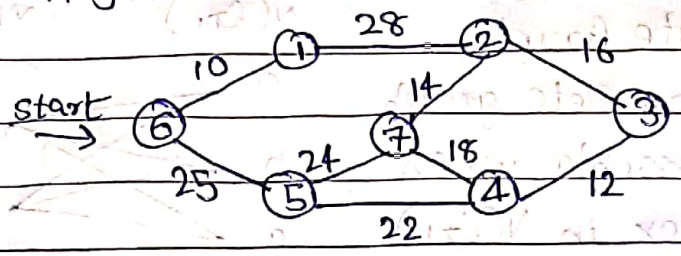
⊛ Insertion Sort :-



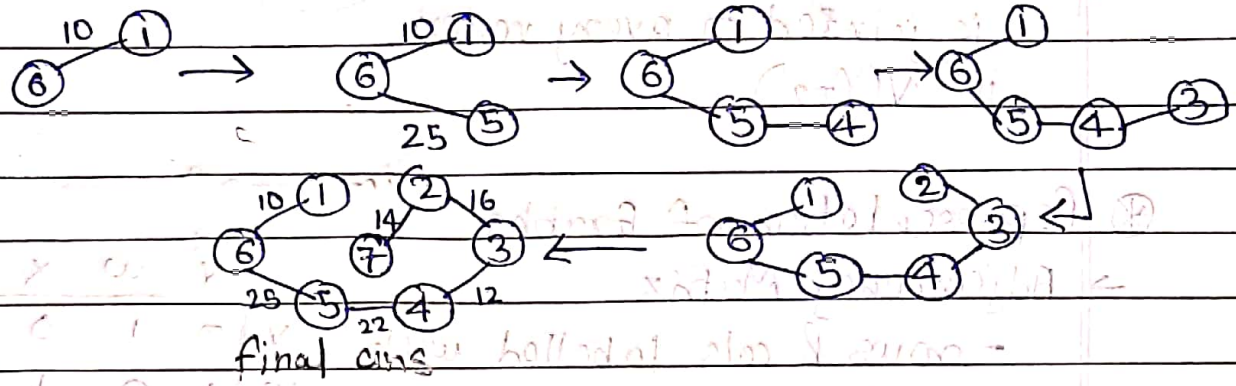
Algo :-
- Sort first 2 elements
- Go to next 2 elements
- Compare with all elements in sorted-sub list
- Repeat until sorted.

✱ Prim's Algorithm

Q



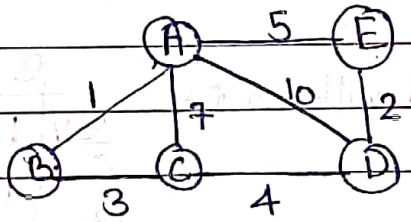
Ans.



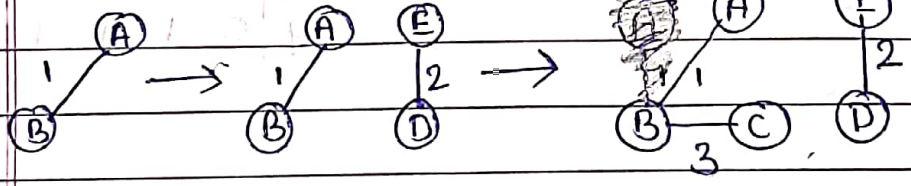
Min cost = 10+25+22+12+16+14 = 99

✱ Krushkal's Algo

Q



Ans.



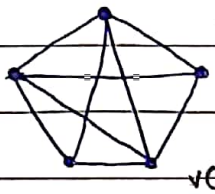
// Since all nodes are covered we stop

Min cost = 1+2+3 = 6

✱ Graphs

→ Complete graph (K_n)
- every pair of vertices is joined by an edge

Eg :-



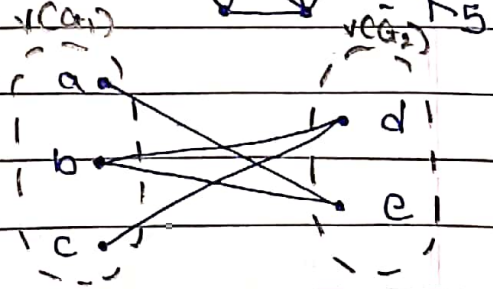
→ Bipartite Graphs

$V(G) = V(G_1) \cup V(G_2)$

$|V(G_1)| = m$, $|V(G_2)| = n$

$V(G_1) \cap V(G_2) = \phi$

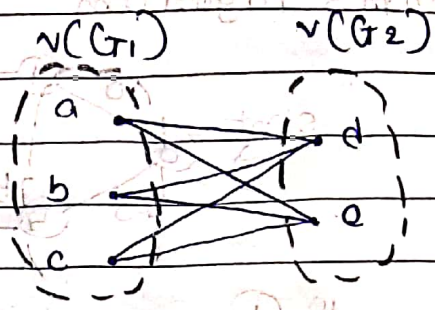
- No edges exists b/w any 2 vertices in same subset.
  Eg :- Edge b/w a & b should not exist.



Page No.
Date

→ Complete Bipartite Graph

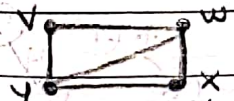
- Should be complete graph $K_{m,n}$, possible when every vertex in $V(G_1)$ is jointed to every vertex in $V(G_2)$.



⊛ Representation of Graphs

→ Adjancency Matrix

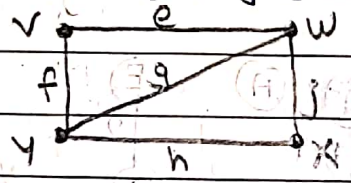
- rows & cols labelled with ordered vertices.



	v	w	x	y
v	0	1	0	1
w	1	0	1	1
x	0	1	0	1
y	1	1	1	0

→ Incidence Matrix

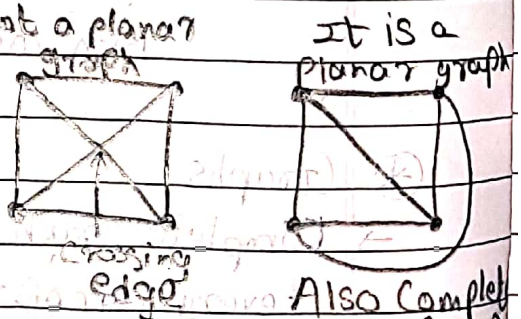
- rows are labelled with vertices
- cols by edges



	e	f	g	h	j
v	1	1	0	0	0
w	1	0	1	0	1
x	0	0	0	1	1
y	0	1	1	1	0

⊛ Planar Graphs :-

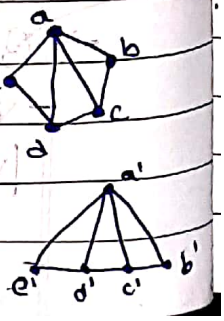
- A graph is planar if it can be drawn in the plane without crossing-edges.



⊛ Isomorphic Graphs :-

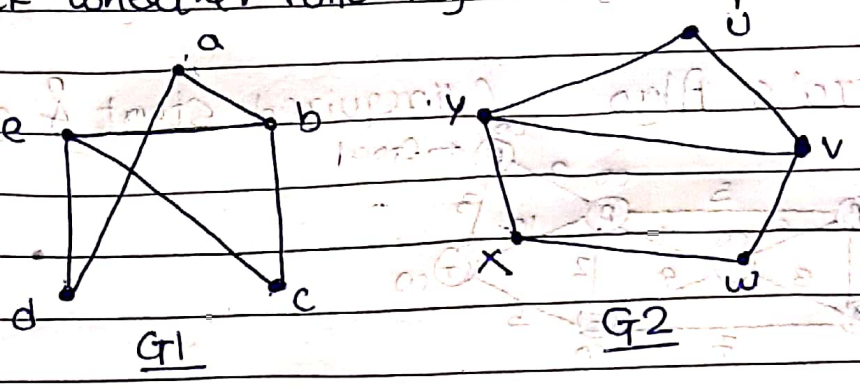
- if there exists one-to-one onto functions $f: V(G_1) \to V(G_2)$ & $g: E(G_1) \to E(G_2)$ such that

- edge e is adjacent to vertices v, w in G_1 if & only if $g(e)$ is adjacent to $f(v)$ & $f(w)$ in G_2



⊛ Isomorphic Graphs with adjacency matrix

Q Check wheather following are isomorphic graphs :-

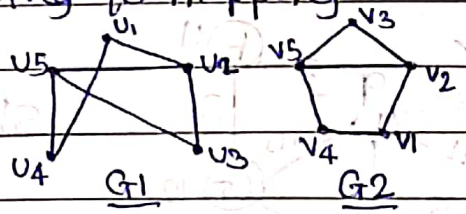


Ans. No of vertices in G1 = 5 in G2 = 5 ✓ satisfied
No of edges in G1 = 6 in G2 = 6 ✓ satisfied
degree sequence G1 : 2,3,2,2,3 G2 : 2,3,2,2,3
Mapping :- • we have two degree 3 vertices i.e e,b & y,v
lets map e ↔ y & b ↔ v

- Now check neighbours degree w has 2 neighbour with degree 2 & 3 same for x. Similarly in graph G1 we have d & a.
- But u & c has two neighbours with degree 3 Hence map u ↔ c. So remaining w ↔ a & x ↔ d
- Sort according to mapping :-

U_1	U_2	U_3	U_4	U_5
a	b	c	d	e
w	v	u	x	y
V_1	V_2	V_3	V_4	V_5

- New graphs :-



- Get adjacency Matrix :-

G1 :

	U_1	U_2	U_3	U_4	U_5
U_1	0	1	0	1	0
U_2	1	0	1	0	1
U_3	0	1	0	0	1
U_4	1	0	0	0	1
U_5	0	1	1	1	0

G2 :

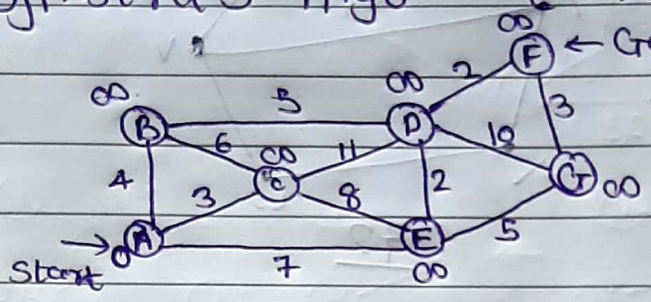
	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	1	0
V_2	1	0	1	0	1
V_3	0	1	0	0	1
V_4	1	0	0	0	1
V_5	0	1	1	1	0

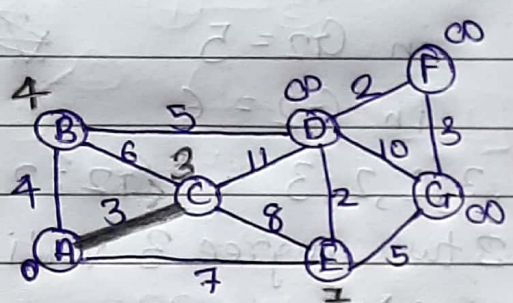
Since adj matrix for G1 & G2 are same
∴ G1 & G2 are isomorphic Graphs.

Page No.
Date

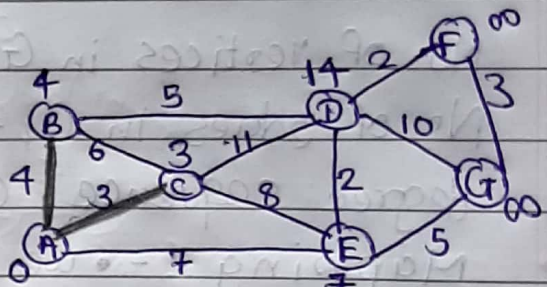
(*) Shortest path Algos :-

→ Djikstra's Algo (If required start & goal node)

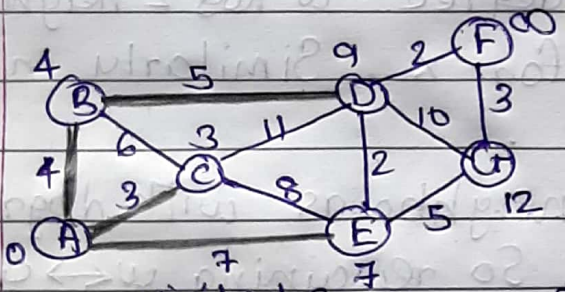




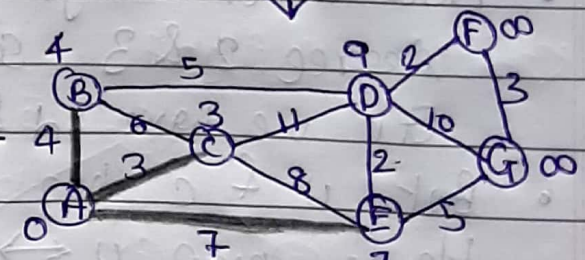
visited {A, C}



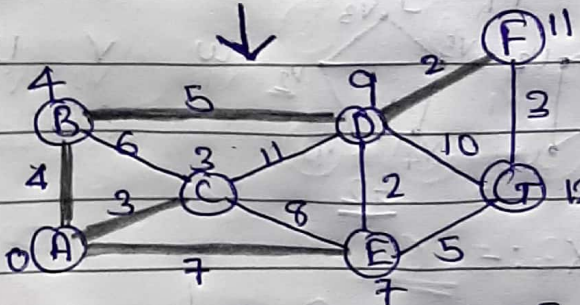
visited {A, B, C}



visited {A, B, C, D, E}



visited {A, B, C, E}



visited {A, B, C, D, E, F}

Shortest path = A - B - D - F
= 11 units