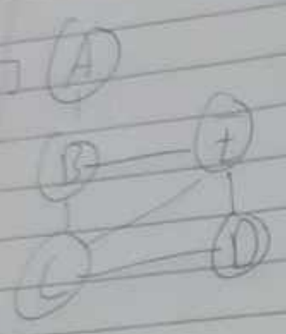
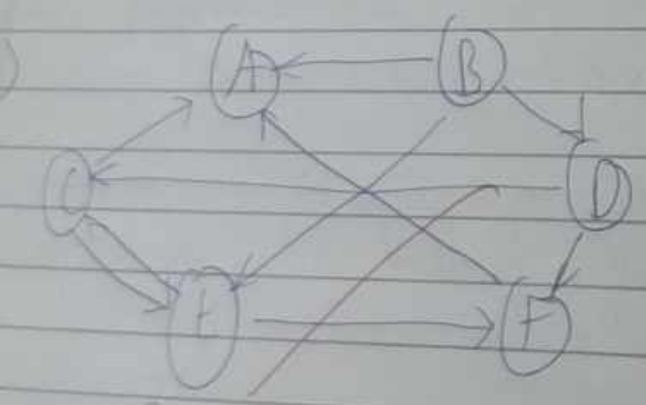
Q. Adjacency matrix and adjacency list repres



Adj list vertex	Adj vertices
A	B, C
B	A, C, E
C	A, B, D, E
D	C, E
E	B, C D

Adj Matrix

	A	B	C	D	E
A	0	1	1	0	0
B	1	1	1	0	1
C	1	0	0	1	1
D	0	1	1	0	1
E	0	0	1	1	0



PTO

Adj list

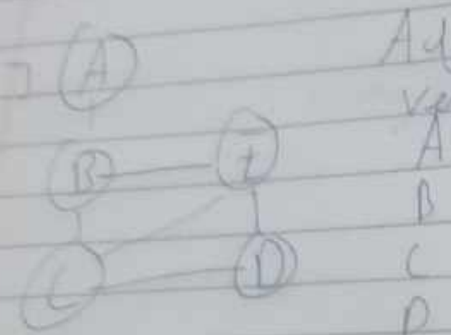
Vertex	Adj Vertices
A	B, C, F
B	A, D, E
C	A, D, E
D	B, C, F
E	B, C, F
F	A, E, D

Adj matrix →

	A	B	C	D	E	F
A	0	1	1	0	0	1
B	1	0	0	1	1	0
C	0	0	0	1	1	0
D	0	1	1	0	0	1
E	1	1	1	0	0	1
F	0	0	0	1	1	0

6/11

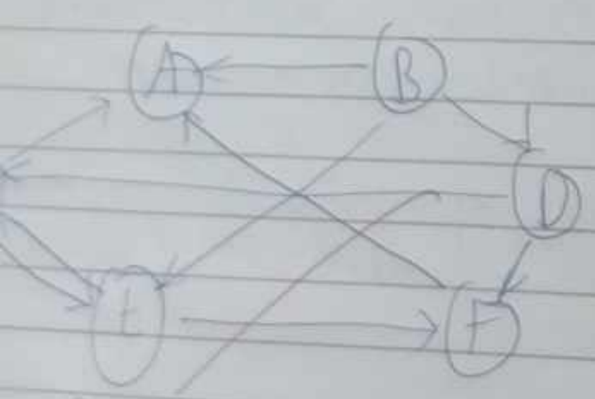
Q2 Adjacency matrix and adjacency list represen



Adj. list vertex	Adj. vertices
A	B, C
B	A, C, E
C	A, B, D, E
D	C, E
E	B, CD

Adj Matrix

	A	B	C	D	E
A	0	1	1	0	0
B	1	1	1	0	1
C	1	0	0	1	1
D	0	1	1	0	1
E	0	0	1	1	0



PTO

Adj list

Vertex	Adj Vertices
A	B, C, F
B	A, D, E
C	A, D, E
D	B, C, F
E	B, C, F
F	A, E, D

Adj matrix →

	A	B	C	D	E	F
A	0	1	1	0	0	1
B	1	0	0	1	1	0
C	0	0	0	1	1	0
D	0	1	1	0	0	1
E	1	1	1	0	0	1
F	0	0	0	1	1	0

6/11

```
new = Create Node(src);
new -> next = adj list [dest];
adj list [dest] = New;
}
void print Adj list (){
    for(int i=0; i<V; i++){
        struct node *temp = adj list [i]
        printf ("%d->", temp -> vertex);
        temp = temp -> next;
    }
    printf ("%null \n");
}
}

main () {
    init ();
    insertEdge (0,1);
    insertEdge (0, 2);
    insertEdge (2, 0);
    insertEdge (2, 3);
    insert (1, 3);
    insert (4, 3);
    insert (2, 4);

    printAdjlis();
    return 0;
}
```

O/P

0: → 2 → 2 → 1 → N
1: 3 → 0 → N
2: 4 → 3 → 0 → 0 → N
3: 4 → 1 → 2 → N
4: 2 → 3 → N

Theory question

1) Write all basic terminologies of graph & explain them

=> I] vertex => it is generally refferred to as node, it is the fundamental unit of graph and is used to represent an entity in a graph

II] Edges => these are the lines of connection btwn vertices in a graph

III] Degree of vertex => it is the no of edges incident on (connected to) that vertex

IV] Path => it is a sequence of vertices where each adjacent pair is connected by an edge

```
printf("\n");
}

int main() {
    int Adjmatrix[V][V];
    init(Adjmatrix);
    insertedge(Adjmatrix, 0, 1);
    insertedge(Adj Matrix, 0, 2);
    insertedge(Adj Matrix, 1, 0);
    insertedge(Adj Matrix, 2, 3)
    insertedge(Adj Matrix, 4, 3)
    insertedge(Adj matrix, 1, 3)
    insertedge(Adj matrix, 2, 4)
    printAdj Matrix(Adj matrix);
    return 0;
}
```

O/P

Adjacency Matrix

```
0 1 1 0 0
1 0 0 1 0
0 1 1 0 1
0 0 1 1 0
```

2) Adjacency list

```
#include <stdio.h>
#include <stdlib>
#define V5

struct node {
    int vertex;
    struct node * next;
};
struct node * adjlist [V];
struct node * create node (int V) {
    struct node * new = malloc (sizeof(struct node));
    new -> vertex = V;
    new -> next = Null;
    return new;
}

void init () {
    for (int i = 0; i<V; i++) {
        adj list [i] = Null;
    }
}

void insertedge (int src, int dest) {
    struct node * new = create node (dest);
    new -> next = adj list [src];
    adj list [src] = new;
```

```
postorder (root -> left);
postorder (root -> right);
printf ("%d", root -> dat);
}

void inorder (struct node * root){
    if (root == Null){
        return;
    }
    inorder (root -> left);
        printf ("%d", root -> data);
        inorder (root -> right);
}
```

O/P

Menu ---

1) preorder traversal
2) postorder traversal
3) inorder traversal

Choose opt : ~~1 2 3 4 5~~

1 2 3 4 8 5 3 6 7

Menu ---

Choose option : 2

8 4 5 2 6 7 3 1

Choose option 3

8 4 2 5 1 6 3 7

Expt 12

Graphs

1) Graph using adjacency matrix

```
#include <stdio.h>
#define V 5

void init (int arr [V][V]){
    int i, j;
    for (i = 0 ; i < V ; i++ ){
        for (j = 0 ; j < V ; j++ ){
            arr [i][j] = 0;
}}}

void insertedge (int arr[V][V], int i, int j)
{
    arr [i][j] = 1;
    arr [j][i] = 1;
}

void printadjmatrix (int arr [V][V])
    int i, j ;
    printf ("\n Adjacency Matrix
    for (i = 0 ; i < V ; i++ ){
        for (j = 0 ; j < V ; j++ ){
            printf ("%d", arr [i][j]
        }
    printf ("%d", arr [i][j])
```

Q2 Pre order, post Order, In Order Traversal

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *left;
    struct node *right;
};

// declare all funcs from last program

void preorder (struct node *root);
void postorder (struct node *root);
void inorder (struct node * root);

int main () {
    int choice;
    while (1) {
        printf ("--- Menu ---\n");
        printf ("1. Pre-order traversal\n");
        printf ("2. post order traversal\n");
        printf ("3 in-order traversal\n");
        printf ("4. Exit");
        printf ("\nchoose option :");
        scanf ("%d", &choice);
        switch (choice) {
            case 1:
                preorder (root);
                break;
            case 2:
                postorder (root);
                break;
            case 3:
                inorder (root);
                break;
            case 4:
                exit (0)
                break;
            default:
                printf ("invalid input ");
        }
    }
    return 0;
}

// same func definition as last program

void preorder ( struct node* root ) {
    if (root == Null) {
        return;
    }
    printf ("%d", root -> data);
    preorder (root -> left);
    preorder (root -> right);
}

void post order (struct node *root) {
    if (root == Null) {
        return;
    }
```

```
struct node* insert left (struct node* root, int ...
{
    if (root == Null) {
        printf ("cant insert in Null");
        return Null;
    }
    root -> left = create (...)
    return root -> left;
}

void delete subtree (struct node* node) {
    if (node == Null) {
        return;
    }
    delete subtree (node -> left)
    delete subtree (node -> right
    free (node);
}

void delete left (struct node* root) {
    if (root == Null || root -> left == Null)
    {printf ("Nothing to delete");
    return;
    }
    delete subtree (root -> left);
    root -> left = Null;
    printf ("In left subtree deleted");
```

```
void delete right (struct node* root) {
    if (root == Null || root -> right == Null) {
        printf ("Noth to delete")
        return 0;
        delete subtree (root -> right) {
        if (root == Null) {
            return 0;
        }
    }

Void display (struct node* root) {
    if (root == Null) {
        return;
    }
    printf ("%d", root -> data);
    display (root -> left);
    display (root -> right);
}
```

O/P

original tree: ~~1234~~ 1248 5 3 6 7

Left subtree deleted

Tree after deletion: 1 3 6 7

→ PTO

Expt 11

Q1 operations on binary tree

```
→ #include <stdio.h>
  #include<stdlib.h>
  struct node {
      int data;
      struct node* left
      struct node* right;
  }
struct node* create (int val)
struct node* insertleft (struct node* root, int val)
void delete leftinsertright (structnode* root);
void deleteleft (structnode* root);
void deleteright (struct node* root);
void display (struct node* root);

int main () {
    struct node* root = create(1);
    insert left (root, 2);
    insert right (root, 3);
    insert left (root→left, 4);
    insertright (root→right, 6)
    insertright (root→right, 7)
}   insert left (root→left→left, 8);

    printf ("Og tree");
    display (root);
    printf ("\n")
```

```
delete/eft (root)
printf ("Tree after deleting leftsubtree of root")
display(root),
return 0;
}

delete node* create (int val) {
    struct node* new =(struct node*) malloc
                        (sizeof(struct node))
    if (new = = Null) {
        print f ("Memory allocfailed
        exit(1);
    }
    new → data = val;
    new → left = Null,
    new → right = Null;
    return new;
}
struct node* insert right (struct node*
int val) { if
        if (root = = Null){
        print f ("cant insert in Null
        return Null;
    }
    root → right = create (val);
    return root → right;
}
```

Step3-
Remove the element ~~at~~ at the front and report deletion if the element at report deletion was the last element then front = rear if not advance for the next element front = front + 1 and set front = 1 and rear = 1

Step4 -
Exit

) Write and explain applications of linear queue

1) Printer:
When documents sent for printing, they will get placed in a printing queue and get printed one after other

) Call center:
The first person that calls gets entered in a queue and gets connected first to the next available operator.

Amusement park:
People wait in a queue

Traffic management
Traffic is a physical queue

5) Diff btwn stack and queue

Stack	Queue
1) It works on LIFO	1) It works in FIFO
2) Insertion happens at top	2) Insertion happens at rear
3) Deletion happens at top at front	3) Deletion happens
4) Only element is accessible	4) Both front and r elements are access
5) Traversal is not possible	5) Traversal possib
6) Can be implemented using array or linked list	6) Can be implem using array, link or circular arr

Nov 6/11

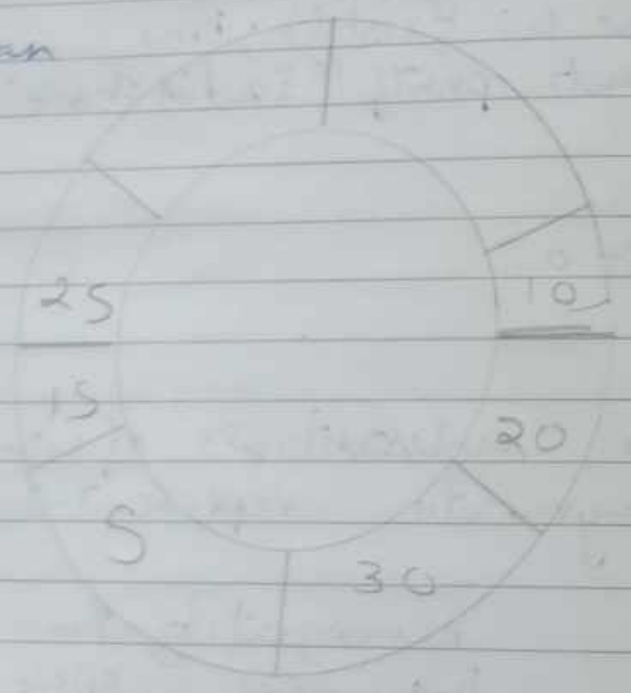
2) Prevents overflow -

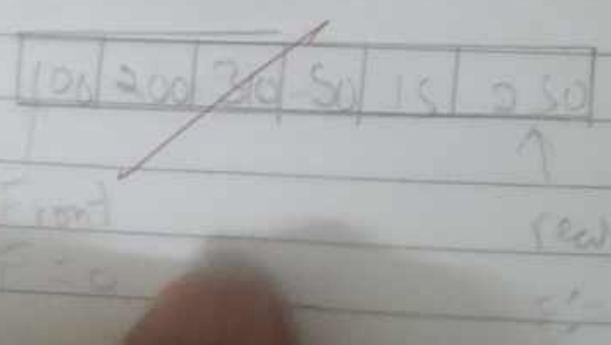
In linear queue, once rear reaches the end, no more insertion are possible even if there are empty slots at the front

3) Fixed size but optimized.
work well for fixed size memory buffers

4) faster operation
Insertion and deletion are O(1)

Diagram





3) Algorithm - insert, delete for circular queue

Insertion
Step1 -
check for overflow

Step2 -
check if the queue is currently empty if f==-1, in case the queue is empty set the values of front and rear to 0 so that the new elements can be stored at the location, if the rear already has some value increment it so that it points to the next pos

Step3 -
the value inserted is stored at the position pointed by the rear

Step4 - Exit

Deletion -
Step1 -
Check for underflow

Step2 -
If rear of front is less than the value of rear then it means that the queue has some value the front is incremented

```
{
  r = (r+1) % size;
}
Queue [r] = value;
printf ("insert %d \n", val);
}

void delete ()
{
  if (f == -1)
  {
    printf ("queue is empty \n");
  }
  else
  {
    printf ("Deleted %d \n", queue [f]);
    if (f == r)
    {
      f = -1
      r = -1
    }
    else
    {
      f = (f+1) % size
    }
}
```

```
int main ()
{
  int choice
  while (1)
  {
    printf ("\n1. insert \n 2. Delete \n 3 display \n 4 exit
    printf ("enter your choice:");
    switch (choice)
    {
      case 1: insert (); break;
      case 2: delete (); break
      case 3: display (); break
      case 4: Exist (); break
      default: printf ("Invalid choice");
    }
  }
  return 0
}
```

2) What are advantages of circular queue show diagramatic representation

→ 1) ~~Efficiency~~ memory utilization
unlike linear queue, space is reused after deletion

→ P.T.O

A) Algorithm for insertion and deletion on Unique queue

(i) Insert / enque

Step1: check for overflow condition.
Step2: check if que is empty, the front and rear set to 0, sto the new value can be stored at the 0th loc

Step3: Value stored in queue at the location pointed by rear
~~Step~~
Step4:- Exit

ii) Delete / Dequeue

1) check for underflow condition, and underflow occurs if front = -1 or front > rear however if queue has some value the front is increased so that it now points to the next val

Exit

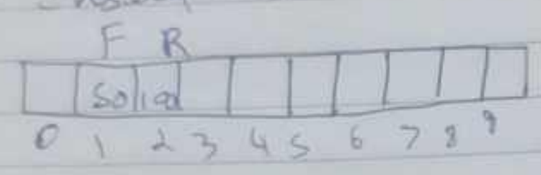
ADS Expt 1.

1) Primitive operation on circular queue
Insert, Delete, Display

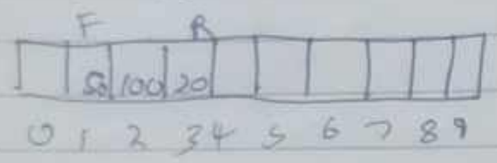
Code-

```
#include <stdio.h>
# include <stdlib>
# define size 5

int queue [size]
int f = -1, rear = -1

void ~~is~~ insert ()
{
  int value;
  if ((f == 0 && r == size-1
    size == -1)
  {
    printf ("queue is full \n");
  }
  else
  {
    printf ("enter val to insert:")
    scanf ("%d", &value);
    if (f == -1)
    {
      f = 0
      r = 0
    }
  }
  else
```

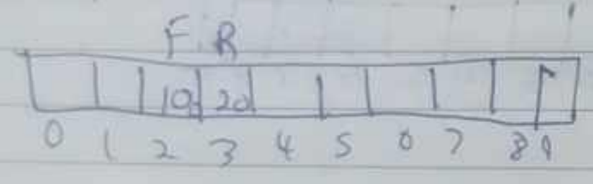
iv) Insert(100)



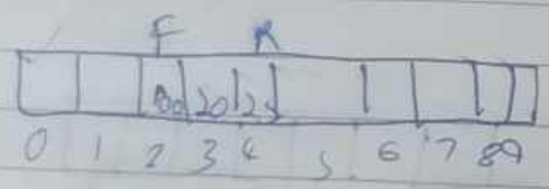
v) Insert (20)



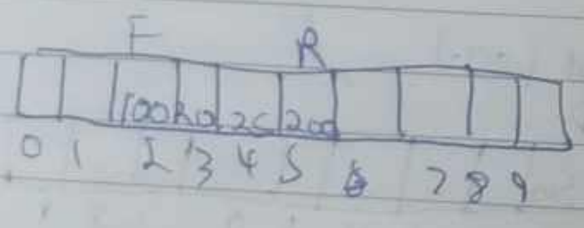
Delete



Insert 25



Insert 200



3) Exp ~~p~~ concepts of priority queue in detail

A3 Priority queue is a type of queue in which each element has a priority associated

- In a normal queue it is First in first out.
- In a priority queue it is the highest priority sort out

They were inserted in

~~In a normal queue~~ it

Types -

i) Ascending priority queue
- In APQ, smallest value gets highest priority
- eg if 30, 10, 20 inserted, delete order will be 10, 20, 30

ii) Descending priority queue
- In DPQ, largest value gets highest priority
- eg if 30, 10, 20, inserted, delete order be 30, 20, 10

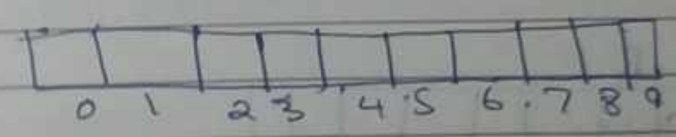
- operations - Insert (enqueue), Delete, Display

Eg: 1) Hospitals
2) OS

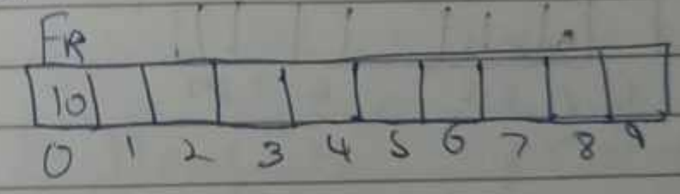
```
int main()
{ int choice;
do
{
printf("\n\nenta your choice:\n1.
insert\n2. Delete\n3 Display\n4.Ex
scanf("%d", &choice);
switch (choice)
{
case 1:
insert()
break;
case 2:
delete()
break;
case 3:
delete();
break;
case 3:
display();
break;
case 4:
printf("\n\tExiting..")
break;
default:
printf("\n Invalid");
}
while (choice != 4);
return;
```

2)

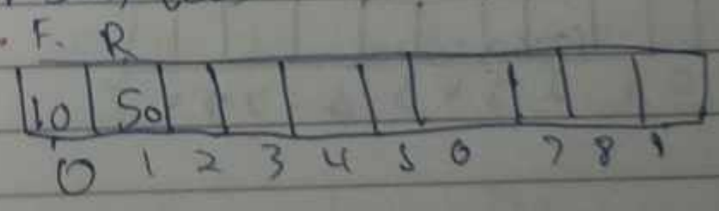
i) Insert (10)
ii) Insert (50)
iii) Delete
iv) Insert (100)
v) Insert (20)
vi) Delete
vii) Insert (25)
viii) Insert (200)



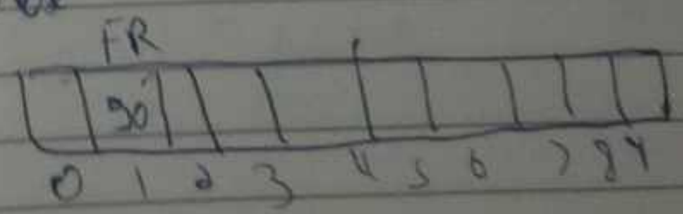
i) Insert 10



ii) Insert (50)



iii) Delete



Expt 9

Code

```
# include <stdio.h>
# include <stdlib.h>
# define MAXS
int queue [max];
int front = -1;
void insert ()
{
  int element;
  if (rear == Max-1]
  {
    int element;
    if (rear == Max-1)
    {
      printf ("\nQueue Overflow");
    }
    else
    {
      print ("\nEnter element to insert
      ~~ins~~ in queue:");
      scanf ("%d", &element;
      if (front == -1)
        front=0
      rear++;
      queue [rear]= element;
    }
  }
```

→ Pb

```
void delete ()
{
  if (front == -1 || front > rear)
  {
    printf "\nQueue underflow");
  }
  else
  {
    prints f ("\nDeleted element is %d,
     queue [fro ]);
     front ++;
  }
}
void display ()
{
  int i;
  if (front == -1 || front > rear)
  {
    printf ("\n Queue is empty");
  }
  else
  {
    printf ("\nQueue elements are
    for (i = front; i <= rear; i++)
    {
      prints f ("%d", queue[i]);
    }
  }
}
```

```
{
printf("%c", x);
}
else if (*e==')')
{
  push(*e);
}
else if (*e==')')
{
  while ((x=pop()) != '(')
  {
    printf("%c", x);
  }
}
else
{
while priority (stack[top]) >= prio
{
push(*e);
}
e++;
}
while (top != -1)
{
printf("%c", pop());
}
return;
}
```

O/P:

enter exp: 1-+* 765

1-765*+

3) Evaluate prefix expression using stack

+ - * + 1 2 / 4 2 1 $ 4.

Input	OP1	OP2	result	stack
2				2
4				2 4
8	4	2	16	16
/				16, 1
2				16, 1, 2
4				16, 1, 2,
/	4	2	2	16, 1,
2				16, 1, 2
1				16, 1, 2,
+	2	1	3	16, 1,
*	3	2	6	16, 1,
-	6	1	5	16, 5
+	5	16	21	21

= 21

6/11

-	*)*-	HGFE^D/
C		HCFE^D/
↑	*)*_*	
B		HCFE^D/
C	*)-*(	HCFE^D/
*	*+	HCFE^D/
+		HCFE^D/
A		HCFE^D/

+A*_*BC*/D^EFGH

4) Write a C program to evaluate in postfix or infix expression

Code -

```
# include <stdio.h>
# include <ctype.h>
char stack [100];
int top = -1;
void push (char x)
{
    stack [++top] = x;
}
char pop()
{
    if (top = -1)
    {
        return -1;
    }
    else
    {
        return stack [top--];
    }
}
int priority (char x)
{
    if (x == '(')
    {
        return 0;
    }
    if (x == '+' || x == '-')
    {
        return 1;
    }
    if (x == '*' || x == '/')
    {
        return 2;
    }
    return 0;
}
int main()
{
    char exp [100];
    char * e, x;
    printf ("enter expression:");
    scanf ("%s", exp);
    printf ("\n");
    e = exp;
    while (*e = '\0')
    {
        if (isalnum (*e))
    }
```

```
}
return0;
}
int main()
{
char exp[100]
char *e, x;
printf("enter expression :");
scanf("%s",exp);
printf("\n");
e = exp;
while(*e != '\0')
{
  if (isalnum(*e))
  {
    printf("%c", *e);
  }
  elseif (*e == '(')
  {
    push(*e);
  }
  elseif (*e == ')')
  {
    while((x=pop()) != '(')
    {
      printf("%c", x);
    }
  }
  else
  {
    while (priority(stack[top]) >= p
    (*e))
```

```
}
  printf("%c", pop());
}
push(*e);
}
e++;
}
while(top != -1)
{
  printf("%c", pop());
}
return0;
}
```

expt 8

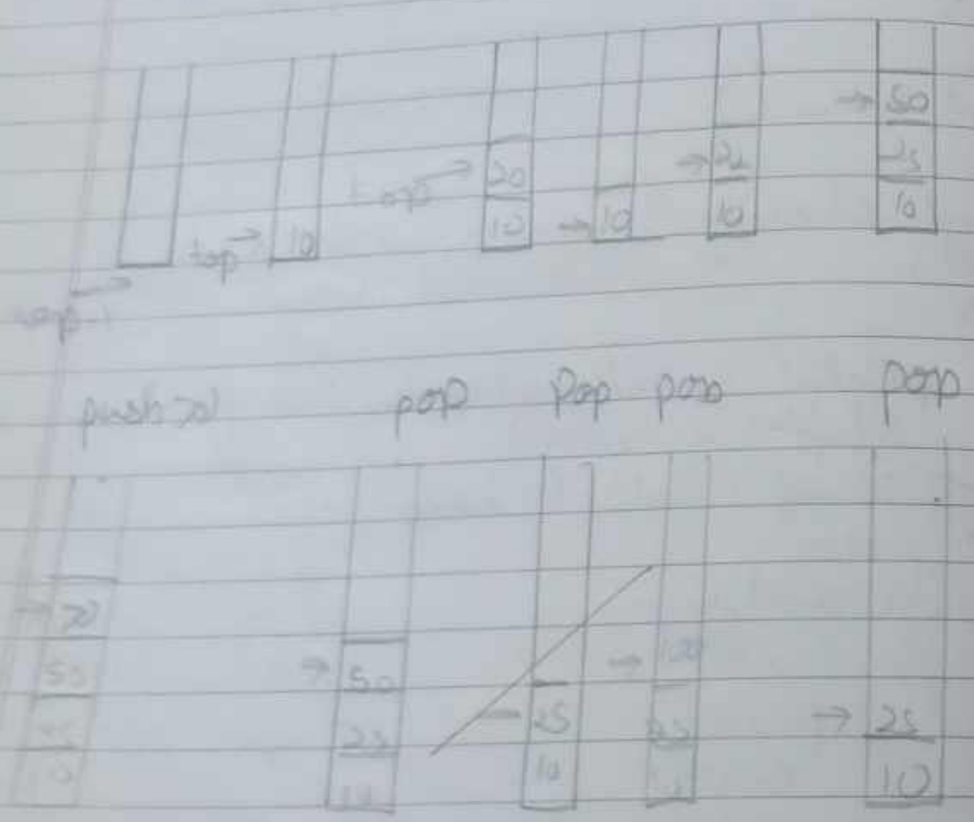
Q Convert infix to prefix using stack

A+(B*C-(D/E^F)*G)*H

Input	stack	Output
H		H
*	*	
)	*)	H
G	*)*	H G
-	*)-*	
)	*)-*)	
F		H G F
^	*)*)^	^
E		H G F E
/	*)*)*/	H C F E ^
D		H G F E D
(	*)*)/(	H C F E ^

```
case 4:
    printf("\n exists");
    break;
  }
} while (choice != 4);
return 0;
}
```

Q stack Size - 8 for push(10), push(20), pop, push(25) push(50), push(20), pop, push(100) pop and draw final output



NM 6/11

Expt 8

Post fix

```
#include <stdio.h>
#include <ctype.h>
char stack[100];
int top=-1;
void push(char x)
{
    stack[++top]=x;
}
char pop()
{
    if (top==-1)
    {
        return -1;
    }
    else
    {
        return stack[top--];
    }
}
int priority (char x)
{
    if (x=='(')
    {
        return 0;
    }
    if (x=='+'||x=='-')
    {
        return 1;
    }
    if (x=='*'||x=='/')
    {
        return 2;
    }
```

Expt 7 (stack)

Code

```
#include <stdio.h>
#include <stdlib.h>
#define Max 5
int top=-1
int s[Max];
void push()
{
int element;
if (top==Max-1)
{
  printf("\n overflow");
}
else
{
  printf("\n enter element to enter stack:")
  scanf("%d", &element);
  top= top+1
  s(top)=element;
}
}
void pop()
{
int item;
if (top==-1)
{
  printf("\n underflow");
}
else
{
  item = s[top]
  top--;
  printf("\n deleted element is:%d", item);
}
}
void display()
{
  int i;
  printf("\n array is:\n");
  for (i=0;i<=top; i++)
  {
    printf("%d", s[i]);
  }
}
int main()
{
  int choice
  do
  {
    printf("\n enter ur choice:\n1.push\n2.pop\n
    3.display\n 4.exit\n");
    scanf("%d", &choice);
    do
    { switch
      case 1:
      push()
      break;
      case 2:
        pop()
        break;
      case 3:
        disp()
        break;
```

```
void del_pos()
{
  struct node* temp, *ptr;
  int pos, i;
  if (start == NULL)
  {
    printf("\n List is empty!\n");
    return;
  }
  printf("Enter pos to delete:");
  scanf("%d", &pos);
  if (pos==1)
  {
    temp=start;
    start=start->next;
    if (start!=NULL)
    {
      start->prev=NULL
      free(temp);
      return;
    }
  }
  ptr=start;
  for(i=1;i<pos && ptr!=NULL;i++)
  {
    ptr=ptr->next;
  }
  if (ptr==NULL)
  {
    printf("\n position out of range")
    return;
  }
```

```
ptr->prev->next=ptr->next;
if (ptr->next!=NULL)
  ptr->next->prev=ptr->prev
```

Q Short note on circular Linked list

A circular linked list (CLL) is a variation of a linked list in which the last node is connected back to the first node, forming a circular list; unlike a linear linked list, it does not have a null pointer at the end.

Circular linked list can be;

Singly circular Linked list: Each node has one pointer to the next node, and the last node points back to the first node.

Doubly circular linked list: Each node has two pointers, one to the next node, with the first and last nodes linked to each other.

Advantages:

Efficient insertion and deletion at both ends

Traversal can start from any node

Suitable for round-robin scheduling and continuous processes

Applications:

Operating systems (CPU scheduling)

Circular queues and buffer management

Multiplayer game turn management

```
void insert_end()
{
  struct node* temp, *ptr;
  temp = (struct node*) malloc (sizeof (struct node));
  printf ("\nEnter data ");
  scanf ("%d", &temp->info);
  temp->next = NULL;
  temp->prev = NULL;
  if (start == NULL)
  {
    start = temp;
  }
  else
  {
    ptr = start;
    while (ptr->next) = NULL)
    {
      ptr = ptr->next;
    }
    ptr->next = temp;
    temp->prev = ptr;
  }
}

void delete_begin()
{
  struct node* temp;
  if (start) = NULL)
    start->prev = NULL;
    printf ("\n Deleted element %d", temp->info
  }
```

```
void delete_end()
{
  struct node* temp;
  if (start == NULL)
  {
    printf ("\n list is empty\n");
    return
  }
  if (start->next == NULL)
  {
    printf ("\n Deleted element %d\n", start->info
    free (start)
    start = NULL;
    return;
  }
  temp = start;
  while (temp->next) = NULL)
  {
    temp = temp->next;
  }
  printf ("\n Deleted element
  %d\n", temp->info);
  temp->prev->next = NULL;
  free (temp);
}

→ P3
```

```
}
}
void ()
{
struct node* ptr;
if (start == NULL)
{
printf("\Empty list\n");
return;
}
else
{
ptr=start;
printf("\n List elements are\n")
while (ptr != NULL)
{
printf("%d", ptr->info);
ptr= ptr->next;
}
}
}
void insert_begin()
{
struct node *temp
temp= (struct node*) malloc (sizeof(struct
printf("\n Enter data");
scanf("%d", &temp->info);
temp->next= start;
temp->prev= NULL;
if (start != NULL)
```

```
start->prev=temp;
start = temp
}
void insert_mid()
{
struct node* temp, *ptr;
int pos,i;
printf("\nEnter pos to insert:");
scanf("%d", &pos);
temp= (struct node*) malloc (sizeof (struct node));
printf("\Enter data:")
scanf("%d", &temp->info);
if (pos==1)
{
temp->next = start;
temp->prev = NULL;
if (start != NULL)
start->prev = temp
start = temp;
return;
}
ptr=start;
for(i=1; i<pos-1 && ptr != NULL; i++)
{
ptr= ptr->next;
}
if (ptr== NULL)
{
printf("\n Pos out of range\n")
return;
}
```

Adding a new node

newnode → next = head;

vi) Deletion -
Removing a node from the linked list
Types:
From the beginning
From the end

E.g. - struct node* temp = head;
head = head → next
free (temp);

3) Diff btwn linked list array

Linked list	Array
1) Dynamic size	1) Fixed size
2) Complex to implement	2) Easy to implement
3) Needs less memory	3) Needs more memory
4) Elements can be of diffrent type	4) All elements must be of same type

Nm
11/9

Exp 6
Doubly linked list

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

struct node
{
    int info;
    struct node* next;
    struct node* prev;
};
struct node * start = Null;
void create ()
{
    struct node* temp, *ptr;
    temp = (struct node*) malloc (sizeof(struct node));
    printf ("\n Enter data: ");
    scanf ("%d", &temp→info);
    temp → next = NULL;
    temp → prev = NULL;
    if (Start == NULL)
    {
        start = temp;
    }
    else
    {
        ptr = start;
        while( ptr → Next != NULL)
        {
            ptr = ptr → next;
        }
        ptr → next = temp;
        Temp → prev = ptr;
```

```
Case 7:
    delete end()
    break;
case 8:
    delete pos();
    break;
Case 9:
    exit(0)
default:
    printf("invalid choice");
  }
}
return 0;
}
```

OP

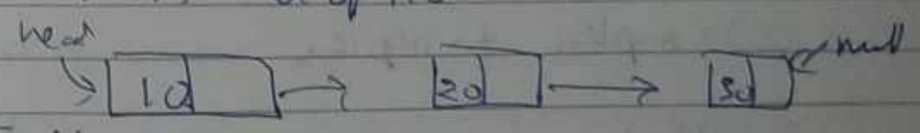
2)

- i) Node:
  - It is the basic building of a linked list
  - Node has two parts, data pointer

Eg:-

```
struct node {
    int data;
    struct Node* next;
};
```

ii) Head -
- Points that stores address of first node in the list
- Without head, we cannot access the list
- Eg:
  struct node in the list points to Null marking end of list



iii) Tail node
The last node in the list points to Null marking end of list

iv) Traversal
Process of visiting each node in the list

v) Insertion

```
void delete pos()
{
struct node* temp, *ptr;
int pos;
if (start == Null)
{
printf ("list is empty");
return;
}
printf ("enter pos to delete");
scanf ("%d", &pos);
if (pos==2)
{
temp = start
start = start -> next
printf ("\n deleted element %d\n
temp -> info );
free (temp)
return;
}
ptr=start;
for(i=1;i<pos-1 && ptr!=Null;
{
printf ("position out of range");
return;
}
temp= ptr -> next
ptr -> next = temp -> next
```

```
printf ("\n deleted element")
free (temp
int main ()
{
int choice
while (1){{
printf ("\n menu \n: create \n display \n
insert begin \n inserted \n insert mid \n
Delete begin \n 7. Delete end \n 8. Delete pos
\n 9. exit")
scanf ("%d", &choice);
switch (choice)
{
case 1:
create ()
break;
case 2:
display ()
break;
case 3
insert begin ()
break;
case 4
insert end ()
case 5 break
insert mid ()
case 6
delete_begin ();
break;
```

```
void insert_end()
{
    struct node * temp, * ptr;
    temp = (struct node *) malloc (sizeof (
    printf ("Enter a data: ");
    scanf ("%d", & temp->info);
    temp -> next = Null;
    if (start == Null)
    {
        start = temp;
    }
    else
    {
        ptr = start
        while (ptr->next != Null)
        {
            ptr=ptr->pnext;
        }
        ptr -> next = temp;
    }
}
void delete_begin()
{
    struct node * temp;
    if (start == Null)
    {
        printf ("In list is empty\n")
        return
    }
    temp = start;
```

```
    start = start -> next;
    printf ("\n deleted element: %d\n", temp->
    pre temp)
}
void delete_end()
{
    struct node, emp, ptr;
    if (start == Null)
    {
        print f ("List is empty");
        return;
    }
    if (start -> nex = Null)
    {
        print ("\n deleted element: \n", start->
        info)
        free start)
        start = Null;
        Return;
    }
    ptr = start;
    while (ptr -> next -> != Null)
    {
        ptr = ptr = ptr -> next;
    }
    temp = ptr -> next;
    printf ("Deleted element = \n", temp
    free (temp);
    ptr -> next = Null
```

Pas:

```
        ptr = ptr -> next;
        ptr -> next = temp;
    }
}
void display ()
{
    struct node* ptr;
    if (start == Null)
    {
        printf ("\n empty list \n");
        return;
    }
    else
    {
        ptr = start
        printf ("\n list elements are: \n");
        while (ptr != Null)
        {
            printf ("%d", ptr -> info);
            ptr = ptr -> next
        }
    }
}
void insert_begin ()
{
    struct node* temp;
    temp = (struct node*) malloc (sizeof(st
    printf ("\n Enta data: ")
    scanf ("%d", & temp -> info);
    temp -> next = st
```

```
        ptr = ptr -> next;
        ptr -> next = temp;
    }
}
void display()
{
    struct node* ptr;
    if (start == Null)
    {
        printf ("\n empty list \n");
        return;
    }
    else
    {
        ptr = start;
        printf ("\n list elements are: \n");
        while (ptr != Null)
        {
            printf ("%d", ptr -> info);
            ptr = ptr -> next;
        }
    }
}
void insert_begin ()
{
    struct node* temp;
    temp = (struct node*) malloc (size of (str
    printf ("\n Enta data: ");
    scanf ("%d", & temp -> info);
    temp -> next = start
    start = temp;
}
```

			3	10	15	12
5	2	6	3	3 10	15	12
5	2	6	3	10	15	12
2	5	6		10	15	12
2	5	6	3	10	15	12
2	5	5	6	10	15	12
2	3	5	6	10	15	12
	3	5	6	10	15	12
	3	5	6	10	15	12
	3	5	6	10	15	15

Exp 5

1) Write a menu driven program that implements singly linked list for the following operations

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int info;
  struct node * next;
};
struct node * start = Null;
void create ()
{
 struct node* start = Null;
 temp = (struct node*) malloc (sizeof (structnode));
 printf ("\n enter data \t");
 scanf ("%d", &temp -> info);
 temp -> next = Null;
 if (start == Null)
 {
   start = temp;
 }
 else
 {
   ptr = start;
   while (ptr -> next != Null
```

4 — 4, 1, 2, 5, 6, 9

1 2 — 2, 1, 4, 5, 6, 9

2 — 2, 1, 4, 5, 6, 9

0 — 1, 2, 4, 5, 6, 9

Hence it is sorted

Q5 Sort the following using shell sort

Pass 1

22 18 29 7 4 55 21 8

22 18 29 7 4 55 21 8

Pass 2

4 18 21 7 22 55 29 8

4 21

18 7

Pass 3

4 7 21 8 22 18 19 25

4 7 21 8 22 29 55

7 4 21 8 22 18 29 55

7 8 21 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 18 21 22 29 55

Sorted are = 7, 8, 9, 18, 21, 22, 29, 55

Implement opposite

3) are = [3, 10, 15, 12, 5, 2, 6]

3 10 15 12 1 5 2 6

Pass 2

5 2 6 3 10 15 12

1 5

11) 256, 99, 145, 239, 20, 18

0 1 2 3 4 5 6 7 8 9

256
099
145
239
018

256
020
145
256
018
099
239

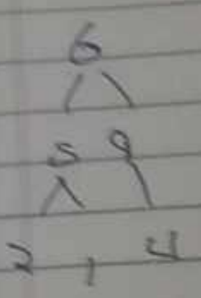
0 1 2 3 4 5 6 7 8 9

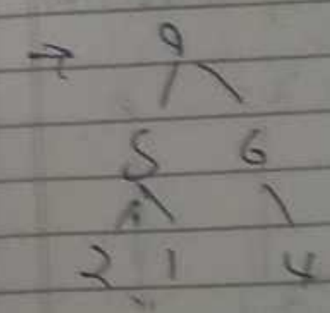
018 018
020 020
145 145
256 256
239 239
099 099

=> 18, 20, 99, 145, 239, 256

5) Sort the following elements using heap sort

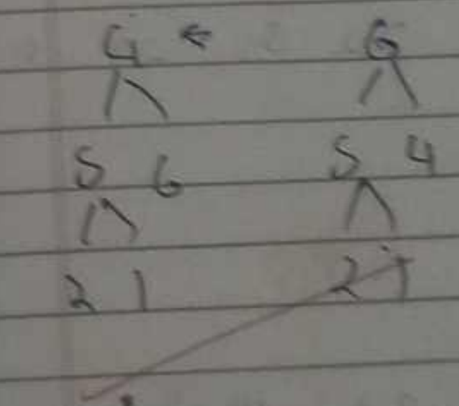
1) 6, 5, 9, 2, 1, 4





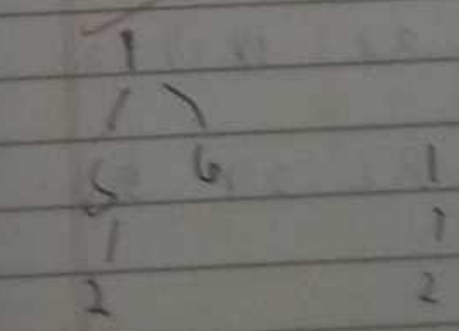
9, 5, 6, 2, 1, 4

4, 5, 6, 2, 1, 9



6, 5, 4, 2, 1, 9

1, 5, 4, 2, 6, 9



5, 1, 4, 2, 6, 9

2, 1, 4, 5, 6, 9

4) Solve using radix sort

1) 100, 225, 390, 4130, 956, 99, 5431

A) 0 1 2 3 4 5 6 7 8 9

0100 100
0225 225
0390 390
4130 4130
0956 0956
0099 99
5431 5431

B) 0 1 2 3 4 5 6 7 8 9

0100 100 390
0390
4130 4130
5431 5431
0225 225
0956 0956
0099 99

6) 0 1 2 3 4 5 6 7 8 9

0100 100
0225 0225
4130 4130
5431 5431
0390 390
0956 956
0099 0099

d) 0 1 2 3 4 5 6 7 8 9

100 ~~0~~99
0225 225
4130 ~~4130~~ 4130 5
5431 5431
390 ~~390~~
~~0099~~ 99

=> 99, 100, 225, 390, 956, 4130, 5431

∴ Hence sorted

~~3) What is the~~

Q2

```
#include <stdlio.h>
int main()
{
int a[50], output [50], count[10];
int n,i,j,max=0,pla=1;
printf ("enta number of elements
while (max/pla>0){
    for (i=0; i<10;i++)
        count[i]=0;
    for (i=0;i<n;i++)
    count (a[i]/pla)%10]++;
    for (i=1 ;i<10;i++)
        count[i] += count[i-1]
    int digit = a[a[i]/pla)%10
    output[--count[digit]]=a[i];
}
for (i=0 ;i<n;i++)
    printf ("%d" a[i])
return 0;
}
```

output :-

Enta the num of array
elements: 5
Enta array elements: 9
8
7
6
5

after pass1: 8 9 7 6 5
pass 2: 7 8 9 6 5
pass 3: 6 7 8 9 5
pass 4: 5 6 7 8 9

Sorted array
5
6
7
8
9

3) What is the output of the ~~is~~ insertion sort

7, 3, 1, 9, 4, 8, 6

→ 1st statement:-

7 3 1 9 4 8 6

3 7 1 9 4 8 6

2nd ~~I~~ iteration

3 7 1 9 4 8 6

1 3 7 9 4 8 6

For loop

```
for (i=0; i<n; i++){
    for (j=i; j<n; j++){
        if (arr[i] < arr[j]){
            temp = arr[j];
            arr[i] = arr[j];
            arr[j] = temp;
        }
}}
```

Experiment 4

Q) Sort element in ascending order using insertion sort

```
→ #include <stdio.h>
int main()
{
    int a[50], n, i, j, temi
    printf ("\n Enter the no. of elements");
    scanf ("%d" &n);
    printf ("\n enter %d elements:", n);
    for (i=0; i<n; i++)
    {
        scanf ("%d", &a[i]);
    }
    for (i=1; i<n; i++)
    {
        for (j=0; j<i; j++)
        {
            if (a[i] < a[j]))
        }
        temp = a[i];
        for (int k=i-1; k>=j; k--)
        {
            a[j+1] = a[k];
        }
        a[j] = temi;
        break
    }
}
printf ("after pass %d ", i);
for (p=0; p<n; p++) {
    printf ("%d", arr[n]);
```

The num of comparison is 10

4)

500, -20, 30, 14, 50

Pass 1

500 -20 30 14 50

-20 500 30 14 50

-20 500 30 14 50

-20 500 30 14 50

Pass 2

-20 500 30 14 50

-20 30 500 14 50

-20 14 500 30 50

~~Pass 3~~

-20 14 500 30 50

-20 14 30 500 50

Pass 4

-20 14 30 50 500

3) Find the number of comparison in the list 100, 200, 300, 400, 500

Pass 1

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

Pass 2

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

Pass 3

100 200 300 400 500

100 200 300 400 500

Pass 4

200 300 400 500

The num

4)

500, -

Pass 1

500

-20

-20

-20

Pass 2

-20

-20

-20

-20

Pass 3

-20

-20

Pass 4

-20

2) WAP to to write elements in descending order as selection sort

Code:-

```
#include <stdio.h>

int main () {
    int arr[5];
    int i,j, temp;
    for (i=0; i<5; i++){
        printf ("enter element");
        scanf ("%d", &arr[i]);
    }
    for (i=0; i<5; i++){
        for (j=i; j<n; j++){
            if (arr[i]<arr[j]){
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }}
    for (i=0; i<n; i++){
        printf ("%d", arr[i]);
    }
    return 0;
}
```

Output

12345

54321

Enter the num of elts

Enter 5 elements

1
2
3
4
5

The smallest is 2

Output :-
Enter 5 elements

1
2
3
4
5
2 is the second smallest element

```
#include <stdio.h>

int main() {
    char str [1000];
    int i=0, word_count=0;
    printf ("Enter a string: ");
    fgets str, sizeof (str), stdin;
    while (str [i] != '\0') {
        if ((str[i] == ' '
        str (i-1) ))
        word_count++;
    }
    i++;
    }
    printf ("Num of words : %d\n",
    }
```

Enter the num of el =

Enter 5 elements

1
2
3
4
5
The ~~second~~ smallest el is

5) #include <stdio.h>

```
int main() {
  int n, i;
    printf("enter the num of elements: ");
    scanf("%d", &n);
    if (n<2) {
      printf("need at least two elements to find the
       second!\n");
      return 0;
    }
    int arr[n];
    printf("enter %d elements: \n", n);
    for (i=0; i<n; i++) {
      scanf("%d", &arr[i]);
    }
    int sm, ss
    if (arr[0] < arr[1]) {
      smallest = arr[0];
        ss = arr[1]; }
    else {
        s = arr[1];
        ss = arr[0]
    for (i=2; i<n; i++) {
      if (arr[i] < smallest) {
        ss = s;
        smallest = arr[i];
      } else if (arr[i] < ss && arr[i] != s) {
        ss = arr[i]; }
      return 0;
}
```

Output

Enter array

2

1

3

2

Sum of duplicate elements:

Q) WAP in C to print all unique elements from

```
#include<stdio.h>
void main(){
    int n;
    printf("enter length");
    scanf("%d", &n);
    int arr1[n];
    int i,j;
    int k=n-1;
    for (i=0;i<n;i++){
    printf("enter elements for first array")
    scanf("%d", &arr1[i]);
    }
    for (i=0; i<n ;i++){
        for (j=i+1; j<n;j++){
            if (arr1[i] == arr1[j]){
```

enter the length of arr 4
enter the element 1
2
3
4

4
3
2
1

Q WAP to count the duplicate elements
arr array

Code :-

```
# include <stdio.h>
int main () {
  int n;
  printf ("enter the num of elements
  scanf ("%d", &n);
  int arr1 [n];
  int i,j,k;
  for (i=0; i<n; i++) {
      printf ("enter elements for first array")
      scanf ("%d", &arr1[i]);
  }
  for (i=0; i<n; i++) {
      for (j=i+1; j<n; j++) {
          if (arr1[i] == arr1[j]) {
              k++;
          }
      }
  }
```

printf ("the num of du

Output

enter element 1
2
1
3
2

The num of duplicate

Q3 WAB in C in p

```
# include <stdio.h>
int main () {
  int n;
  printf enter
  scanf ("%d
  int arr i
  int i,j;
  int k=n
  for (i=0;
  printf (
  scanf ("
  }
  for (i=0
    for (j
```

Extra

Q: Copy elements from 1 array to another in reverse

Code:

```
# include <stdio.h>

int main () {
    int n;
    printf ("enter the length of array");
    scanf ("%d", &n);
    int array1 [n];
    int array2 [n];
    int i,j;
    for (i =0; i<n; i++){
        printf ("enter elements for first array");
        scanf ("%d", &array1 [i]);
    }

    for (i =0; i<n; i++){
        array2 [i]= array1 [n-1-i];
    }

    for (i=0; i<n; i++){
        printf ("%d \n", array2 [i]);
    }
    return 0;
}
```

→ PTO

Q. State the limitations of linear search in terms of complexity

1) Inefficient for large data sets:
→ Linear search has a time complex of O(n), where n is the num of elements in the list

2) No improvement for sorted data
→ Linear search does not take advantage of sorting

3) Average and worst cases are the same
→ The average case and worst case are both requires scanning about half or all the elements

4) Not suitable for time critical applications
→ Due to its linear time, it is not ideal for applications where fast is necessary

17/7/25

Q. Copy el
reverse

Code:

```
#include <

int main()
    int n;
    printf(
    scanf
    int a
    int a
    int i
    for (i
    prin
    scan
    }
    for(
    }
    }
    }
    }
```

3) Compare Linear search vs Binary search

Linear	Binary
1) Unsorted or sorted data	Must be sorted
2) Sequential access	Divide and conquer
3) O(n)	O(log n)
4) less efficient	more efficient
5) Simple code	More complex

→ PTO

Q2 Search data using binary search

```
#include <stdio.h>
int main()
{
  int a[100], n, m, i, h, j, found=0, k;
  printf("enter number of array element: ");
   scanf("%d", &n);
   printf("enter key %d be searched")
   scanf("%d", &k);
   printf("\n enter sorted array element\n");
    for (i=0; i<n; i++)
    {
      scanf("%d", &a[i]);
    }
    l=0; b<n-1);
    while (b<=l)
    {
      m (int)(l+b)/2);
      if (k==a[m])
      {
        printf("\n element found at index %d", m);
        break;
      }
      else
      {
        if (k>a(m))
        {
          l=m+1;
          if (k<a(m)) {
```

Experiment 2

A

```
#include <stdio.h>
int
intmain () {
    int arr[] = {56, 36, 89, 57, 1, 0, 67, 513
    int n = size of (arr)
    int k, f;
    printf ("enter the element to search :"),
    scanf ("%d", & k);
    for (int i=0; i<n; i++) {
      if (arr [i] == k) {
          f = 1;
          printf ("element found at index %d\n", i);
          break;
      }
    }
    if (f == 0) {
        printf ("Element not found in array")
    }
    return 0;
}
```

O/P :

enter the element to search : 1
Element found at index is

OP 2 : Enter element to search : 55
element not found in array

Pattern 3:

```
* * * * * *
  * * * *
    * * *
      * *
        *
```

Pattern 4:

```
1
1 2
3 3 3
1 2 4 3 6
5 5 5 5 5
```

Pattern 5

```
*   *
  *   *
    *   *
      *   *
```

```
}
printf("pattern 3:\n");
for(int i=n; i>=1; i--) {
    for(s=0; s<(n-i); s++)
        printf(" ");
    for (j=1; j<=(2*i-1); j++ {printf("
    printf("\n");
}
printf("patterns:\n");
for(int i=1; i<=n; i++) {
    for(int j=1; j<=n-i; j++ {printf(" ");}
    for (int k=1; k<=2*i-1; k++) {printf("
    printf("\n");
}
return 0
}
```

O/P:

```
enter num of rows: 5
pattern 1:
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Pattern 2

```
*
* *
* * *
* * * *
* * * * *
```

Pattern 3:

```
* * * * *
  * * * *
    * * *
      * *
        *
```

Pattern 4:

```
1
1 2
3 3 3
1 2 4 3 4
5 5 5 5 5
```

Pattern 5

```
* *
  * *
    * *
      * *
        * *
```

Extra

Q Pattern

1) 1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

2) *
# #
$ $ $
? ? ? ?

3) * * * *
* * *
* *
*

4) 1
1 2
3 3 3
1 2 3 4
5 5 5 5 5

5) *
* * *
* * * *

A) #include <stdio.h>

```
int main () {
    printf ("enter num of rows:" ); int n;
    scanf ("%d", &n);
    printf ("Pattern 1: \n");
    printf
    for (int i=1; i<=n; i++) {
        for (int j=1; j<=i; i++) {
            printf ("%d", i);
        }
        printf ("\n");
    }
    printf ("pattern 2: \n")
    char arr[] {*, #, $, ?}
    for (int i=1; i<5; i++) {
        for (int j=1; j<=i; j++) {
            arr[i-1]
```

5) Square the odd elements of an array

Code :-

```
# include <stdio.h>
int main () {
int arr[10];
int i;
int
for (i=0; i<10; i++) {
    printf ("enter element");
    scanf ("%d", & arr [i]);
  }
for (i=0; i<10; i++){
  if (arr [i] % 2 == 0){
      printf ("%d\n", arr [i]);
  }
  else {
      printf ("%d \n", arr [i] * arr[
  }
  return 0;
}
```

Output

enter element 1 2 3 4 5 6 7 8 9 0

1 2 9 4 25 6 49 8 81 0

(2) Pattern

1) 1
   2 2
   3 3 3
   4 4 4 4
   5 5 5 5

5) *
   * * *
   * * * *

A) # include

int main (
pri
sc
pr

2) find greatest and smallest num

Code:-

```
#include <stdio.h>
int main() {
    int arr[10];
    int i, j;
    for (i=0; i<10; i++){
        printf ("enter element,");
        scanf ("%d", &arr[i]);
    }
    int greatest = arr[0];
    int s = arr[0];

    for (i=0; i<10; i++){
        for (j=i+1; j<10; j++){
            if (arr[j] > greatest) {
                greatest = arr[j];
            }
            elseif (arr[j]<s){
                s = arr[j];
            }
        }
    }
    printf ("The greatest is %d\n", greatest);
    printf ("The smaller element is %d\n", s);
    return0;
}
```

Output

element : 1 2 3 4 5 6 7 8 9 0

greatest = 9

smallest = 0

3) Find reptetive elements of an array

Code:

```
# include <stdio.h>
int main () {
int int arr[10];
  int i,j;
  for (i=0; i<10; i++){
    printf ("enter elements");
    scanf ("%d", &arr[i]);
  }
  for (i=0; i<10; i++){
    for (j=i+1; j<10; j++){
      if (arr[i] == arr[j]){
        printf ("repetere elements are %d", arr[i]);
      }
    }
  }
  return 0;
}
```

output

enterelement 1 2 3 3 4 5 6 7 5 4
repetetive elements are 3 4 5

2) Display the pattern

Code:

```
#include <stdio.h>
int main () {

    int i,j;
    for (i=0; i<5; i++){
        for (j=0; j<=i; j++){
            if (i%2==0){
                printf("*");
            }
            else {
                printf("#");
            }}
            printf("\n");
        }
        return 0;
    }
```

Output

```
*
# #
* * *
# # # #
* * * * *
```

Assignment 1 * Write program to implement array operations *

1) Write a program to get avrg of 10 elements of a array

Code :-

```
#include<stdio.h>
int main ()
{
    int arr [10];
    int i;
    float sum;
    for (i=0; i<10; i++){
     printf ("enter the elements");
     scanf ("%d", & arr[i]);
     sum = sum + arr [i];
    }
     printf("the avrg of 10 num is %f", sum);
     return 0;
}
```

Output

enter elements 1 2 3 4 5 6 7 8 9 10

avrg is 4.5