* Exp. 1

* write a program to implement array operation:

1) Find avg of 10 no. using array.

→
```
#include<stdio.h>
int main()
int i, num[10], sum=0;
printf("Enter the numbers:\n");
for(i=0; i<10; i++)
{
  scanf("%d", &num[i]);
}
for (i=0; i<10; i++)
{
  sum+ = num[i];
}
float avg = sum/10.0;
printf("the average is:%.2f",avg);
return0;
}
```

Output :-

```
1        8
2        9
3        10 9
4        the average is : 5.40
5
6
7
```

2) Display the following pattern.

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

→
```
#include<stdio.h>
int main(){
 int i, j;
 for (i=1; i<=4; i++)
 {
   for (j=1; j<=i; j++)
   {
     if (i%2 ==0)
     {
       printf("#");
     }
     else
     {
       printf("*");
     }
   }
   printf("\n");
 }
 return0;
}
```

Output :-

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

array

3) Find first repeating element of array.

```
# include<stdio>
int main ()
{
  int arr[] = {4, 2, 1, 3, 2, 5, 1};
  int n = sizeof(arr) / size of (arr[0]);
  int found = 0;
  for(i=0; i<n-1; i++)
{
    for (j=i+1; j<n; j++)
{
      if (arr[i] == arr[j])
{
        printf("First repeating element is:
                %d\n", arr[i]);
        found = 1;
        break;
}
}
  if (found) {
      break;
  }
  else {
    printf("No repeating element found");
  }
  return0;
}
```

OUTPUT :-

First repeating element is : 2.

4) Find greatest and smallest element in array

```
→ #include<stdio.h>
int main ()
int arr[] = {15, 10, 17, 15, 34, 2};
int n = sizeof(arr)/sizeof(arr[0]);
int i;
int min = arr[0];
int max = arr[0];
for (i=1; i<n; i++)
{
  if(arr[i] < min)
  {
    min = arr[i];
  }
  if(arr[i] > max)
  {
    max = arr[i];
  }
}
printf("The smallest no is = %d", min);
printf("The largest no is = %d", max);
return 0;
}
```

Output :-
The smallest no is = 2
The largest no is = 34

5) WAP searching the one position element

```
→ #include <stdio.h>
int main ()
{
  int n;
  printf("Enter the size of the array");
  scanf("%d", &n);
  int arr[n];
  printf("Enter %d elements", n);
  for (int i = 0; i<n; i++)
  {
    scanf("%d", &arr[i]);
  }
  for (int i=0; i<n; i++)
  {
    if (i%2 != 0)
    {
      arr[i] = arr[i] * arr[i];
    }
  }
  printf("updated array \n");
  for (int i=0; i<n; i++)
  {
    printf("%d", arr[i]);
  }
  return 0;
}
```

Output :-
Enter the size of array : 5
Enter 5 elements

updated array:

~~1 4 3 16 5~~

1 2 9 4 25

* Practise

1)

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

```
#include<stdio.h>
int main () {
    int i,j ;

    for (i=1 ; i<=4 ; i++)
    {
        for (j=1 ; j<=4-i ; j++)
        {
            printf ("__");
        }
        for (j=1 ; j<=i ; j++)
        {
            printf ("*__");
        }
        printf (" \n");
    }
    return 0;
}
```

2)
```
* * * *
  * * *
    * *
      *
```

→
```
#include <stdio.h>
int main(){

int i,j;

for(i=4;i>=1;i++)
{
  for(j=1;j<=4-i;j++)
{
    printf("  ");
}
  for(j=1;j<=2*i-1;j++)
{
  printf("*");
}
  printf("\n");
}
return 0;
}
```

3)
```
*
# #
$ $ $
? ? ? ?
```

→
```
#include<stdio.h>
int main(){
int i,j;
```

```
for(i=1;j<=4;i++){
  for(j=1;j<=i;j++)
{
    if(i==1)
      printf("*  ");
    else if(i==2)
      printf("#  ");
    else if(i==3)
      printf("$  ");
    else if(i==4)
      printf("?  ");
}
  printf("\n");
}
return0;
}
```

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

→
```
#include <stdio.h>
int main(){
int i,j;
for(i=1;j<=5;i++){
    for(j=1;j<=i;j++)
{
      printf("%d  ",j);
}
    printf("\n");
}
```

```
    return 0;
}
```

Q.5
```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

→
```
#include <stdio.h>
int main() {
  int i,j;
  for (i = 1 ; i<=5 ;i++)
  {
    for (j = 1 ;j<= i ;j++)
    {
      printf ("%d ",i);
    }
      printf ("\n");
  }
  return 0;
}
```

Exp 2 :-

① Search data using linear search. Consider following list to perform linear search.
56, 36, 89, 57, 1, 0, 67, 59

→
```
#include<stdio.h>
int main()
{
  int key,i, a[8];
  printf("Enter the elements of array:");
  for (i = 0; i<8 ;i++)
  {
    scanf ("%d", &a[i]);
  }
  printf ("Enter the search key: ");
  scanf ("%d", &key);
  for (i=0 ;i<8 ;i++)
  {
    if (a[i] = = key)
    {
      printf (" the element is found
                at %d pos.", i+1);
      break;
    }
  }
  if (i = = 8)
  {
    printf ("element not found");
  }
  return 0;
}
```

i. search element : 1 & found item found or found not

→ output :-

Enter the elements of array:
56
36
89
57
1
0
67
59
enter search key: 51
element found is at 5th position.

ii. Search element : 55

→ enter the elements of array:
56
36
89
57
1
0
67
59
enter search key: 55
element not found.

Q-2 Search same data using binary search

→
```
# include <stdio.h>
int main () {
    int k, l, h, m, i, a[8];
    printf ("enter the elements of
    array in sorted order:");
    for (i=0; i<8; i++)
{
        scanf ("%d", &a[i]);
}
    printf ("Enter the search key:");
    scanf ("%d", &k);
l = 0
h = 7;
while (l<=h)
{
    m = (l+h)/2;
    if (a[m] == k);
{
        printf (" element found");
        return 0;
}
else
{
    if (k < a[m])
{
        h = m-1;
}
else
{
        l = m + 1;
```

```
}
}
printf("element not found");
return 0;
}
```

output :-
enter the elements of array in sorted order:
0
1
2
3
4
5
6
7
enter the search key: 5
element found.

Q.3 Compare linear and binary search.

Linear	Binary
1. It works on both sorted & unsorted arrays.	· It works only on sorted arrays
2. Time complexity in (n).	· Time complexity in log(n)
3. It is used for small & unsorted data set.	· It is used for large data sets.
4. simple to implement	· complex to implement.
5. compares with each first element	· compared with middle element
6. Algorithm type is sequential.	· Algorithm type is divide & conquer

Q.4 State limitations of linear search in terms of time complexity.

→ Linear search has limitations in terms of time complexity especially for large datas' every element in array to determine if it is present or not, which makes it slower. The binary search is on sorted data

Extra questions :-

1. write a program to copy the elements of one array into another array in a reverse order.

→ [arr2[i] = arr[4 - i - 1]
↓ no. of elements.

CODE :-

```
# include <stdio.h>
int main ()
{
  int arr[4], arr2[4], i;
printf ("enter the elements of arr1: \n");
for (i = 0; i< 4; i++)
scanf ("%d", &arr [i]);
for (i=0 ; i< 4; i++)
{
    arr 2[i] = arr [4-i-1];
}
  printf(" reversed array is: ");
  for (i=0 ; i< 4; i++)
{
    printf("%d", arr 2[i]);
}
  return 0;
}
```

Output :-
enter the elements of array1:
3 4 5 6
reversed array is:
6543.

2) WAP in C to count total number of duplicate elements in an array.

→
```
#include <stdio.h>
int main() {
int arr [5], i, j, count = 0;
for (i=0; i< 5; i++)
{
  printf ("enter %dth elements of
          array :", i+1);
  scanf (i = 0 ; i < 5 ; j < 5 ; j++)
  & scanf ("%d", &arr [i]); }
  for (i = 0;  i < 5 ; i++)
{
    for (j = i+1 ; j< 5 ; j++)
{
        if (arr [i] == arr [j])
{
            count++ ;
            break;
    }
  }
}
  printf (" the total count of repeated
          cat elements is : %d", count);
  return 0;
}
```

Output :-
enter 1 element of array : 6
enter 2 element of array : 7
enter 3 element of array : 3
enter 4 element of array : 2

enter 5 element of array : 3
the total count of repeated elements is : 1 .

3. WAP in C to print all unique elements in an array.

→
```
# include <stdio.h>
int main ()
{
    int arr[100], n, i, j, count=0;
    printf ("enter no. of elements:");
    Scanf ("%d", &n);
    for (i=0; i<n; i++)
    {
    printf ("element %d", i+1);
    scanf ("%d", arr[i]);
    }
    for (i=0; i<n; i++)
    {
      for (j=0; j<n; j++)
    {
      if (arr[i] == arr[j])
    {
        count ++;
    }
    }
printf ("unique elements in array
        are : ");
  if (count == 1)
{
  printf ("%d", arr[i]);
}
```

```
return 0;
}
```

output :-
enter no. of elements : 6
element 1 = 3
element 2 = 5
element 3 = 3
element 4 = 2
element 5 = 7
element 6 = 5

unique elements in array are :-
2, 7 .

Q.4 WAP to seperate odd and even integers into seperate arrays

→
```
# include <stdio.h>
int main ()
{
   arr[10]
  int , arr1[] , arr[]
  int i
   for (i = 0 ; i<10 ; i++)
{
     printf (" element %d =", i+1);
     scanf ("%d", &arr[10]);
   }
   for (i = 0 ; i<n ; i++)
{
     if ( arr[i] %2 = = 0)
```

```
{
    arr[i] = arr1[int evencount++];
}
else
{
    arr[i] = arr2[int oddcount++];
}
}
printf("even numbers:");
for (i=0; i < evencount; i++)
{
    printf("%d", arr1[i]);
}
printf("odd numbers:");
for (i=0; i < oddcount; i++)
{
    printf("%d", arr2[i]);
}
return 0;
}
```

output :-
arr = [10, 21, 4, 45, 66, 93, 1, 9, 81, 12]

even numbers = 10, 4, 66, 12
odd numbers = 21, 45, 93, 1, 81, 9.

5. WAP to find second smallest element in an array.

→
```
#include <stdio.h>
int main()
{
    int arr[] = {10, 5, 8, 20, 2};
    int n = 5;
    int i, j, temp;
    for (i=0; i<n; i++)
    {
        for (j = i+1; j<n; j++)
        {
            if (arr[i] > arr[j])
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    for (i=1; i<n; i++)
    {
        if (arr[i] != arr[0])
        {
            printf("second smallest
                element is: %d", arr[i]);
            break;
        }
        else
        {
            printf("No second element:");
        }
```

```
return 0;
}
```

output :-
second smallest number : 5

6) WAP to c to count the total no. of words in a string

```
→ # include <stdio.h>
int main()
{
    char str [100];
    int i, words = 1;
    printf("enter a string:");
    Scanf ("%d [^\n]", str);
    for (i=0; str [i] != '\0'; i++)
    {
        if (str[i] == ' ')
        {
            words ++ i;
        }
    }
printf ("Total no. of words
    : %d\n", words);
    return 0;
}
```

output :-
enter a string : I Love coding
Total no. of words : 3

Exp. 3

① write 'c' program to sort data from given array using Bubble sort & selection sort.

(i) sort elements in ascending order using Bubble sort.

```
# include <stdio.h>
int main()
{
    int a[50], n, i, j, temp;

    printf("Enter the number of elements:");
    Scanf ("%d", &n);
    printf("enter %d elements:", n);
    for (i=0; i<n; i++)
    {
        scanf("%d", & a[i]);
    }
    for (i=0; i<n-1; i++)
    {
        for (j=0; j<n-i-1; j++)
        {
            if (a[j] > a[j+1];
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
printf ("sorted arrays in ascending
        order:");
```

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

Output :-
Enter number of elements : 5
enter 5 elements : 8 6 69 45 1
sorted array : 1 6 8 45 69

(ii) Sort elements in decending order using selection sort

```
#include<stdio.h>
int main()
{
    int a[50], n, i, j, max, temp;
    printf("enter number of elements:");
    scanf("%d", &n)
    printf("\n enter %d element:");
    for (i=0; i<n; i++)
{
    scanf("%d", &a[i]);
    for (i=0; i<n-1; i++)
{
    max = i;
    for (j = 1+i, j<n; j++)
{
    if (a[j] > a[max])
{
        max = j;
}
}
    temp = a[i];
    a[i] = a[max];
    a[max] = temp;
}
printf("\n sorted array in descending order:");
for (i=0; i<n; i++)
{
    printf("%d", a[i]);
}
    return 0;
}
```

Output :-

Enter number of elements : 5
enter 5 elements : 8 6 3 78 98
sorted array in descending order :
98 78 8 6 3

iii) Find the number of comparison required in bubble sort method of the following list having 5 number : 100, 200, 300, 400, 500.

Pass1 100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

Pass2 100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

Pass3 100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

Pass 4:-

100 200 300 400 500

100 200 300 400 500

Total comparison

$4 + 3 + 2 + 1 = 10$

iv) Sort the given array in ascending order using selection sort method and show diagramatic representation of every iteration 500, -20, 30, 14, 50

Pass1:-

500 -20 30 14 50

-20 500 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

-20 14 500 30 50

Pass 3:

-20 14 500 30 50

-20 14 30 500 50

+20 14 30 500 50

Pass 4 -20 14 30 500 50

-20 14 30 50 500.

Exp 4

Q.1 Sort element in ascending order using insertion sort.

→
```
# include <stdio.h>
int main()
{
  int a[50], n, i, j, temp;
  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<1; j++)
{
      if (a[i] < a[j]))
{
      temp = a[i];
    for (int k=i-1; k>=j; k--)
{
      a[j+1] = a[k];
}
    a[j] = temp;
   break;
}
}
}
    printf("after pass %d : ", i);
    for (p=0; p<n; p++)
{
      printf("%d", arr[p]);
}
```

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

output :-

```
Enter the number of array
elements : 5
Enter array elements : 9
8
7
6
5
after pass 1 : 8 9 7 6 5
after pass 2 : 7 8 9 6 5
after pass 3 : 6 7 8 9 5
after pass 4 : 5 6 7 8 9
Sorted array:
5
6
7
8
9
```

Q.2 Sort elements in ascending orders using radix sort:

```
#include <stdio.h>
int main()
{
    int a[50], output[50], count[10];
    int n, j, i, max=0, place=1;

printf("enter number of elements:");
scanf("%d", &n);
printf("enter %d numbers:\n", n);
for (i=0; i<n; i++) {
    scanf("%d", &a[i]);
    if (a[i] > max)
        max = a[i];
}
while (max/place > 0) {
for (i=0; i<10; i++)
    count[i] = 0;
for (i=0; i<n; i++)
    count[(a[i]/place)%10]++;

for (i=1; i<10; i++)
    count[i] += count[i+1];
for (i=n-1; i>=0; i--) {
    int digit = (a[i]/place)%10
  output[--count[digit]] = a[i];
}

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

3. what is the output of the insertion sort after the second iteration following numbers.

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

→ 1st Iteration:-

7 3 1 9 4 8 6

3 7 1 9 4 8 6

2nd Iteration:-

3 7 1 9 4 8 6

1 3 7 9 4 8 6

∴ After 2nd Iteration the result is

→ 1, 3, 7, 9, 4, 8, 6.

4) Sort the following number using radix sort.

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

Bucket →
Input ↓

A)

	0	1	2	3	4	5	6	7	8	9
0100	100									
0225			225							
0390	390									
4130	4130									
0956							0956			
0099										
5431		5431								99

B)

	0	1	2	3	4	5	6	7	8	9
0100	100									
0390										390
4130				4130						
5431				5431						
0225			225							
0956										956
0099										~~99~~ 99

C)

	0	1	2	3	4	5	6	7	8	9
0100		100								
0225			225							
4130		4130								
5431					5431					
0390				0390						
0956										0956
0090	0090									

a)

	0	1	2	3	4	5	6	7	8	9
0090	99									
0100	0100									
4130					4130					
0225	0225									
0390	0390									
5431						5431				
0956										

⇒ ∴ 99, 100, 225, 390, 956, 4130, 5431

∴ Hence, sorted.

(ii) 256, 99, 145, 239, 20, 18

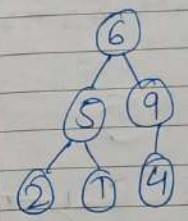
	0	1	2	3	4	5	6	7	8	9
256							256			
099										099
145						145				
239										239
020	020									
018									018	

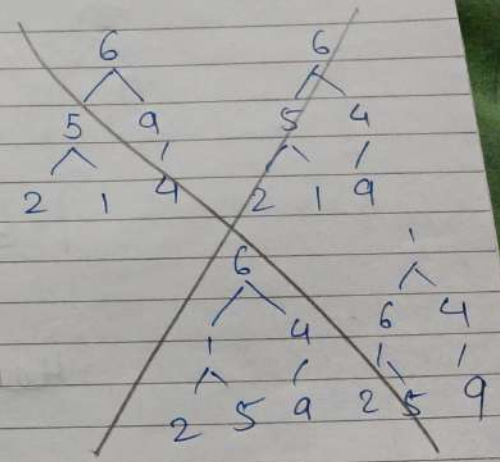
	0	1	2	3	4	5	6	7	8	9
020			020							
145					145					
256						256				
018	018									
099										099
239				239						

	0	1	2	3	4	5	6	7	8	9
018	18	R								
020	20									
239			239							
145		145								
256			256							
099	099									

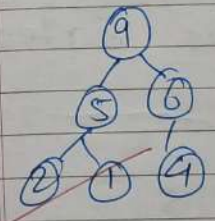
5. Sort the following elements using heap sort.

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





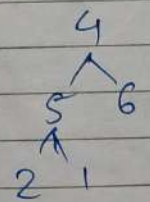
→

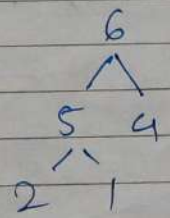


9, 5, 6, 2, 1, 4

4, 5, 6, 2, 1, 9

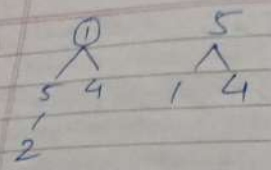
2)



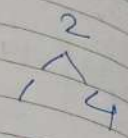


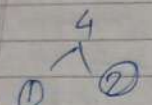
6, 5, 4, 2, 1, 9

1, 5, 4, 2, 6, 9

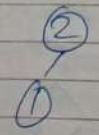


5, 1, 4, 2, 6, 9
2, 1, 4, 5, 6, 9

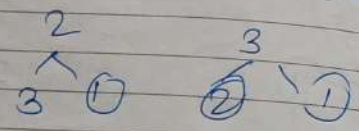




4, 1, 2, 5, 6, 9
2, 1, 4, 5, 6, 9



2, 1, 4, 5, 6, 9
1, 2, 4, 5, 6, 9.



3, 2, 1, 4, 10
1, 2, 3, 4, 10

∴ Hence, it is sorted.

6) Sort using shell sort

(i) 22, 18, 29, 7, 9, 55, 7, 8, 21, 29

= n = 8

n/2 = 4

classmate
Date
Page

22 18 29 7 9 55 21 8

{2, 2, 9, 7}, {18, 55}, {29, 21}, {7, 8}

= ~~9 →~~ 9, 22, 18, 55, 21, 29, 78

n/4 = 8/4 = 2

{9, 22, 18, 55}, {21, 29, 7, 8}

= 9 18 22 55 7 8 21 29

n/8 = 8/8 = 1

{9, 18, 22, 55, 7, 8, 21, 29}

Sorted array: 7, 8, 9, 18, 21, 22, 29, 55.

(ii) 3, 10, 15, 12, 1, 5, 2, 6

→ n = 8
n/2 = 8/2 = 4

3, 10, 15, 12, 1, 5, 2, 6

{3, 1}, {10, 5}, {15, 2}, {12, 6}

1, 3, 5, 10, 2, 15, 6, 12

$n/4 = 8/4 = 2$

{1, 3, 5, 10} , {2, 15, 6, 12}

1, 3, 5, 10, 2, 6, 12, 15

$n/8 = 8/8 = 1$

{1, 3, 5, 10, 2, 6, 12, 15}

1, 2, 3, 5, 6, 10, 12, 15

Sorted array: 1, 2, 3, 5, 6, 10, 12, 15

Nm 119

Exp 5

Q.1 write a menu driven program that implements singly linked list for following operations.
create, insert node at begining insert node & middle & end.

Q.2 delete first node, delete node at middle, delete node at end.

Q.3 Create, display, count no. of nodes, reverse, search.

→
```
# include <stdio.h>
# include <stdlib.h>
Struct node{
        int data;
        struct Node * Next;
};

Struct node * head = NULL;

void  createlist();
void  insertAtBeginning();
void  insertAt end();
void  insertAt position();
void  deleteat end();
void  deleteatbegining();
void  delete at position();
void  display();
void  countNodes();
void  reverse();
void  search();
```

```
int main()
{
 int choice;
 while (1) {
 printf("singly linked list menu");
 printf("1. create list ");
 printf("2. Insert at beginning");
 printf("3. Insertat end ");
 printf("4. Insert at position");
 printf("5. Delete at beginning");
 printf("6. Delete at end ");
 printf("7. Delete at position");
 printf("8. Display ");
 printf("9. count Nodes");
 printf("10. Reverse list");
 printf("11. search node");
 printf("12. Exit");
 printf("enter your choice:");
 scanf("%d", &choice);

 switch(choice) {

 case1:
 createlist();
 break;
 case 2:
 Insertat beginning();
 break;
 case 3:
 Insert at end();
 break;
 case 4:
 Insertatposition();
 break;
 case5:
 delete at beginning();
 break;
 case6:
 delete at end();
 break;
 case7:
 delete at position();
 break;
 case 8:
 display();
 break;
 case9:
 countNodes();
 break;
 case10:
 reverse();
 break;
 case11:
 search();
 break;
 case12:
 exit(0);
 default: print("Invalid choice!");
 }
 }
 return0;
 }
 struct Node* CreateNode(int data){
 Struct Node* NewNode = (Struct Node*)
```

```
malloc (size of (structNode));
newNode -> data = data;
newNode -> next = NULL;
return newNode; }

void createlist()
{
  int n, data;
printf("enter number of node:");
scanf("%d", &n);
head = NULL;
for (int i=1; i<=n; i++)
{
  printf("enter data for node %d:", i);
  scanf("%d", &data);
  struct Node *NewNode = CreateNode(data)
  if (head == NULL) head = newNode;
else {
    struct Node *temp = head;
    while (temp -> next != NULL) temp
      = temp -> ~~Ne~~ next;
      temp -> next == newNode;
  }
 }
}
void insertatbegining()
{
  int data;
  printf("enter data:");
  scanf("%d", &data);
  struct Node * newNode = CreateNode
  (data);
```

```
newNode -> next = head;
 head = newNode;
}
void InsertAtend()
{
    int data;
    printf("enter data:");
    scanf("%d", &data);
    struct Node *newNode = CreateNode(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    struct Node *temp = head;
    while (temp => next != NULL)
      temp = temp -> next;
      temp -> next = newNode;
}
    void insertAtPosition()
{
      int data, pos;
      printf("enter position & data:");
      scanf("%d %d", &pos, &data);
      struct Node *newNode = CreateNode(data);
    if (pos == 1)
  {
    newNode -> next = head;
        head = newNode;
    return;
  }
  struct Node *temp = head;
  for (int i=1; temp != NULL && i<pos
```

```
1, i++)
    temp = temp -> next;
if (temp == NULL) {
        printf(" position out of range!\n");
        free (newNode);
        return ;
}
newNode -> next = temp -> Next;
temp -> Next = newNode;

void deleteAtbeginning()
{
    if (head == NULL)
{
        printf("List empty!");
        return;
}
struct Node *temp = head;
    head = head -> next;
    free (temp);
void deleteatend()
{
    if(head == NULL)
{
        printf("list is empty!");
        return ;
if (head -> next == NULL)
{
        free (head)
        head = NULL;
        return ;
}
```

```
struct Node * temp = head ;
while ( temp -> next -> next != NULL)
temp = temp -> next ;
free (temp -> next);
temp -> next = NULL ;

void deleteatposition()
{
    int pos , i ;
    struct node *current = head ,* temp;
    if (head == NULL)
{
        printf("list is empty");
        return ;
}
printf(" enter position :");
scanf("%d" ,&pos);
if (pos < = 1)
{
        printf("Invalid position!");
        return 0;
    if (pos = = 1)
{
        deleteatbeginning();
        return ;
}
struct *Node * temp = head;
if (int i = 1 ; temp != NULL && i < pos-1 ; i++)
    temp -> next;
if (temp = = NULL || temp -> next == NULL)
{
        printf (" position out of range");
```

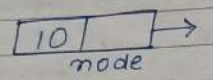
```
return 0;
}
struct node *del = temp -> next;
temp -> Next = del -> Next;
free(del);
}
void display()
{
    if (head = = NULL)
{
        printf("list is empty ! ");
        return ;
}
struct Node *temp =head ;
printf("list :");
while (temp ! = NULL)
{
    printf ("%d -> " temp -> data);
        temp = temp -> next;
}
printf("NULL");

void count Nodes()
{
    int count = 0 ;
  struct Node *temp = head ;
 while ( temp ! = NULL)
{
    count + + ;
 temp = temp -> next ;
}
printf ("Total Nodes = %d", count);
```

```
void reverse ()
{
    struct Node * prv = NULL, *CURR =head;
 * next = NULL;
while (curr ! = NULL) next = curr -> next;
curr -> next = prev ;
        prev = curr ;
        curr = next ;
}
    head = prev ;
}
void search()
{
    int key , pos =1 ;
  printf (" enter element to search :");
  scanf ("%d", &key);
  struct Node* temp =head;
  while (temp ! = NULL)
{
        if ( temp -> data = =key)
{
    printf (" element %d found at
                position %d", key, pos);
    return ;
}
  temp = temp -> next ;
  pos + + ;
}
printf ("element not in list");
}
```

Q.4 Basic Terminology of linked list :-

a) NODE : Basic building block of linked list. It contains data (value) & next (pointer to next node)

eg.



b) head : The head is a reference or pointer to first node of list. NULL or nun.

c) Tail:
Last node where next == NULL

d) Next:
each node has a next reference to following node.

e) NULL pointer:
Special value that indices that pointer does not have object memory.

f) empty linked list:
Singly linked list with no nodes in it
head → NULL

Q.5

Array	linkedlist
· Resizing not possible	· can grow or shrink.
· memory allocation is fixed size.	· size of array thus memory can change.
· stored in contiguous memory	· stored in non-contiguous memory.
· memory usage is efficient.	· uses extra memory for pointers.
· simple to implement	· complex implementation.
· O(1) access time.	· O(n)

11/9

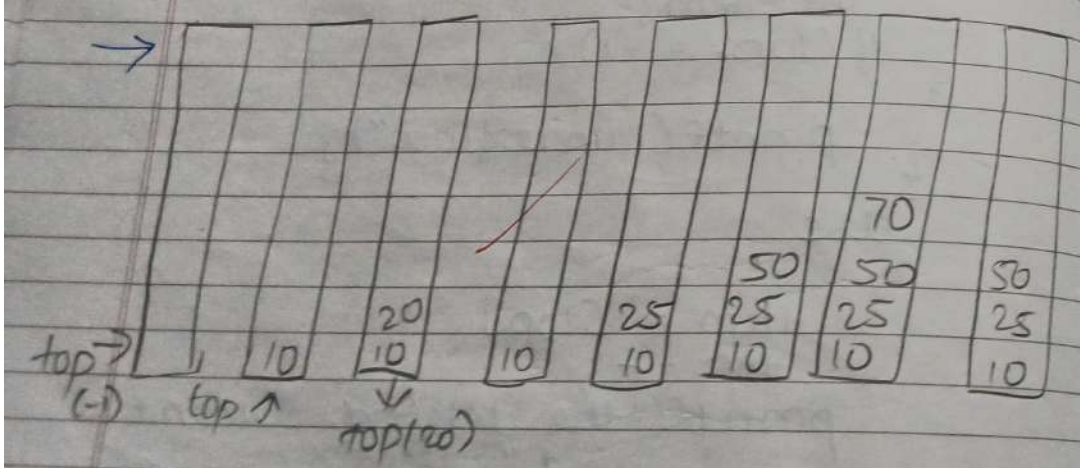
Exp7

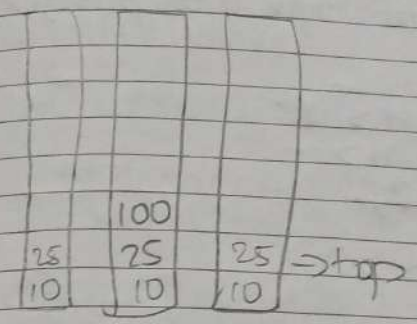
Stack code :-

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

void push();
void pop();
void display();

int S [Max];
int top = -1;

int main() {
    int choice;
    printf("\n Menu:-");
    printf("\n 1. Push");
    printf("\n 2. POP");
    printf("\n 3. Display");
    printf("\n 4. Exit");
    do {
    printf("Enter your choice :");
    Scanf("%d", &choice);
Switch (choice)
{
    case 1: Push(); break;
    case 2: POP(); break;
    case 3: display(); break;
    case 4: Exit(0);
    default: printf("Invalid choice!");
}}
while (choice != 4);
return 0;
}
void push()
{
    int element;
    if (top = = Max - 1)
    {
        printf("overflow");
    }
    else {
    printf("enter value");
    Scanf("%d", &element);
    top = top + 1;
    S[top] = element;
}}
void pop() {
    int item;
    if (top = = -1)
    {
        printf("underflow");
    }
    else
    {
        item = S[top];
        top - -;
        printf("the deleted element is
                %d", item);
}}
void display()
{
    if (top = = -1)
    {
```

```
} printf("stack empty");
else {
printf("stack elements are :");
for (int i=0; i<=top; i++)
{
  ~~max~~ printf("%d", S[i]);
}
}}
```

Q.2 Stack size 8 for push(10), push(20), pop, push(25), push(50), push(70), pop, pop, push(100), pop, and draw final output





6/11

Exp 8

Q.1 WAP to evaluate infix to postfix expression.

→
```
#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 the expression");
scanf ("%s", &exp);
printf("\n");
    e = exp;
while (*e != '\0')
{
    if (isalnum(*e))
    printf("%c", *e);
else if (*e == '(')
push (*e);
else if (*e == ')')
{
    while ((x = pop()) != '(')
    printf ("%c", x);
}
else
{
    while (priority (stack [top]) >= priority
        (*e))
    printf ("%c", pop());
    push (*e);
}
    e++;
}
while (top != -1)
{
    printf ("%c", pop());
}
    return 0;
}
```

Q.2 convert infix expression into prefix expression using stack.

→ expression - A+(B*C-(D/E^F)*G)*H

I/P		stack	O/P
H	H		H
)	*	*	
G	)	*(	
*	G		HG
)	*	*(*	
F	)	*(*(	
^	F		HGF
E	^	*(*(^	
/	E		HGFE
D	/	*(*(/	HGFE^
(	D		HGFE^D
-	(	*(*	HGFE^D/
C	-	*(*-	
	C	*(-	HGFE^D/*C
	*	*(-*	
	B		HGFE^D/*CB
	(	*	HGFE^D/*CB*-
	+	*+	
	A	+	HGFE^D/*CB*-*

+A HGFE^D/*CB*-*A+

→ +A*-*BC*/D^EFGH

Q.3 write a C program to evaluate infix or postfix expression.

→
```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
# define SIZE 40

int pop();
void push(int);

char postfix (SIZE);
int stack [SIZE], top=-1;
int main()
{
    int i, a, b, result, peval;
    char ch;
    for(i=0; i<size; i++)
    {
        stack [i] = -1;
    }
printf("\n Enter a postfix expression:");
scanf("%s", postfix);

for(i=0; postfix[i] != '\0'; i++)
{
    ch = postfix [i];
    if (isdigit (ch))
    {
        push (ch-'0');
    }
else if (ch == '+' || ch == '-' || ch == '*' ||
```

```
ch == '/')
{
    b = pop();
    a = pop();

  switch(ch)
  {
    case '+' : result = a+b;
    break;
    case '-' : result = a-b;
    break;
    case '*' : result = a+b;
    break;
    case '/' : result = a/b;
    break;
  }
  push(result);
}
}
pEval = pop();
printf("The postfix evaluation is : %d",
      pEval);
return 0;
}
void push(int n){
  if (top< size -1)
{
      stack[++top] = n;
}
else
{
  printf("stack is full");
  exit(-1);
}}
int pop()
{
    int n;

  if (top> -1)
{
    n = stack[top];
    stack [top--] = -1;
    return n;
  }
  else
{
  printf("stack is empty!!n");
      exit(-1);
}
}
```

4) Evaluate prefix expressions using stack.

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

Input	Op_1	Op_2	Result	stack
2				2
4				2,4
$	4	2	16	16
1				16,1
2				16,1,2
4				16,1,2,4
/	4	2	2	16,1,2

2				16, 1, 2, 2
1				16, 1, 2, 2, 1
+	1	2	3	16, 1, 2, 3
*	3	2	6	16, 1, 6
-	6	1	5	16, 5
+	16	5	21	21

* Algorithms

1. conversions from infix to postfix expression.

- Read expressions from left to Right.
- If an operand (A-Z), (0-9) is encountered add it directly to output
- If an operator is encountered pop operator from stack to output if they have higher or equal priority.
- push the operator to stack.
- If an opening parenthesis '(' is found, push it into stack.
- If closing parenthesis ')' is found, pop operators from stack to output until opening parenthesis is encountered.
- After scanning entire expression pop any remaining operators from stack to output.

2. conversion from infix to prefix expression.

- Reverse the infix string. Note that while reversing you must interchange left and Right parenthesis.
- obtain the postfix expression of the given infix expression obtained from step 1.
- Reverse the obtained postfix expression (o/p) to get prefix expression.

3. Evaluating postfix expression

- Scan the elements of string from left to right.
- If the element is in operand, push it into stack.
- If the element is an operator, pop two operands from the stack.
- The first pop is second operand 2 Second pop is first operand.
- perform arithmatic operation & push result back into stack.
- when input expression is completely transversed pop operand stack & return the value.

4. Evaluating prefix expression

- Start from last element of expression
- check the current element.
- If it is operand, push to stack.
- If it is an operator, pop 2 operands from stack (last & second last)
- perform operation 2 push elements back to stack.

- Do this till all elements of expression are transversed & return the top of stack which will be the result of the operation.

EXPREIMENT 9

Q.1 perform primitive operation of linear Queue - Insert, delete, Display.

→
```
# include <stdio.h>
# include <std.lib>
# define SIZE 5

void enqueue();
void dequeue();
void display();
int queue[size];
int front = -1, rear = -1;

int main()
{
    int choice;
   printf("1. enque \n 2. Dequeue \n 3. Display
          \n 4. Exit \n);
  do{
   printf("enter your choice :");
   Scanf("%d", &choice);
   Switch (choice)
   {
    case1 : enqueue();
            break;
    case2 : dequeue();
            break;
    case 3: Display();
            break;
    case4 : exit(0);
    default : printf("invalid choice");
```

```
}
}
while (choice !=4)
return 0;
}
void enqueue()
{
int element;
if (rear ==SIZE-1)
{
printf(" Queue is full");
}
else
{
printf(" Enter element to insert :");
Scanf("%d", &element);
if (front == -1)
front =0;
rear ++;
queue [rear] = element;
}
}
void dequeue()
{
if (front == -1 || front>rear)
{
printf(" Queue is empty");
}
else
{
printf("Deleted element : %d",
queue [front]);
front ++
}
}
void display()
{
if (front = = -1 || front >rear)
{
printf (" underflow , Queue is
empty");
}
else
{
printf( " Queue elements are: ");
~~for (front = = 1 +~~
for ( int i = front ; i <= rear ; i++)
{
printf("%d \t", Queue [i]);
}
print ("\n");
}
}
```

Q.2 Write algorithm to enqueue (insert) & dequeue (delete) on linear queue.

a) Insertion

- Check the overflow condition.
- Check if queue is empty. In case it is, then both front and rear are set to 0. So that new value can be stored at 0th location.
- otherwise, If queue already has some

value, then rear is increamented such that it points to next location in array.
- The value is stored in queue of the location pointed by REAR.
- EXIT.

(b) Deletion:
1. we check for underflow condition. An underflow occurs if front = -1 or front > rear.
2. If queue has some values, the front is increamented so that it now points to next value in queue.
3. Exit.

Q.3 write algorithm of insertion & deletion for circular queue

(a) Insertion
- If front = 0 and rear = max-1 the queue will overflow.
- If front is rear +1 it will also overflow queue.
- If front = -1 and rear = -1, front = rear = 0.
- else if rear = max-1 and front! ~~= rear~~ = 0 Rear = 0.
- else, Rear = rear + 1 end
- queue [rear] = val
- exit.

(b) Deletion
- If front = -1, write underflow.
- Go to step four (4) at end of if.
- Set VAL = queue [front]
- If front = rear, set front = rear = -1
- else, if front = Max - 1 Set FRONT = 0
- else, set front = front + 1
- Exit.

Q.4 Applications of queue:

1. Serving requests on a single source resources, like printer, CPU task scheduling etc.
2. In real life senarios, call centers phone system uses queues to hold people calling them in an order, until a service representative is free.
3. Handling interupts in real-time systems. The interrupts are handled in some order as they arrive i.e. first come first serve.
4. FCFS job sheduling
5. online banking fund transfers requests.

Q.5 Explain concept of priority queue

→ - An element with higher priority is processed before an element with lower priority.

- Two element with same priority are processed on a first come first serve (FCFS) basis.
- Types :-
- Ascending PQ
- descending PQ

Q.6 Array size 10, insert (10), insert (50), Delete, Insert (100), Insert (20), delete, insert (25), Insert (200)

	0	1	2	3	4	5	6	7	8	9	
→ front	10										rear
	10	50									Insert 50
	~~10~~	50									delete
	~~10~~	50	100	20							ins 100
	~~10~~	50	100	20							ins 20
	~~10~~	~~50~~	100	20							del
	~~10~~	~~50~~	100	20	25						ins 25
	~~10~~	~~50~~	100	20	25	200					ins 200

NW 11/10

Doubly Linked List.

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

void create();
void display();
void insert_begin();
void insert_end();
void insert_position();

struct node {
    int info;
    struct node* next;
    struct node* prev;
};

struct node* start = NULL;

int main() {
    int choice;
    while (1) {
        printf("\n MENU ");
        printf("\n 1. CREATE");
        printf("\n 2. DISPLAY");
        printf("\n 3. INSERT_BEGIN");
        printf("\n 4. INSERT_END");
        printf("\n 5. INSERT_POSITION");
        printf("\n 6. EXIT");
        printf("\n Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                display();
                break;
            case 3:
                insert_begin();
                break;
```

Exp 10

Q.1 Write a C program to perform primitive operations on circular queue.

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

void enqueue();
void dequeue();
void display();

int queue [SIZE];
int front = -1, rear = -1;
int main()
{
        int ch;
printf("1. Enqueue \n 2. Dequeue \n
        3. Display \n 4. Exit");
do
{
printf("enter your choice");
scanf("%d", &ch);
Switch (ch)
{
case 1 : enqueue(); break;
case 2 : dequeue(); break;
case 3 : display(); break;
default : printf("Invalid choice! =");
}
while (ch != 4);
```

```
return 0;
}
void enqueue()
{
    int value;
    if ((front == 0 && rear == size-1)
        || (rear + 1 == front))
{
    printf("In Queue is full");
}
else
{
    printf("Enter value to insert");
    scanf("%d", &value);
}
}
Void dequeue
{
    if (front == -1)
{
    printf("Queue is empty");
}
else
{
    printf("Deleted %d from the
        queue", queue[front]);
    if (front == rear)
{
    front = rear -1;
}
else
{
    front = (front+1) %size;
}
}
}
Void display()
{
    if (front == -1)
{
    printf("Queue is empty)
}
else
{
    int i = front;
    printf("In Queue element");
    while(1)
{
    printf("~~In Que~~ %d", queue[i]);
        if (i == rear)
            break;
        i = (i+1) %size;
}
    ~~printf("\n");~~
}
}
```

Q. Applications on priority queue?

→ Huffman codes:

priority queue helps build huffman trees for efficient data.

- OS Algorithms

used for process scheduling to execute high-priority tasks first

- Dijikstra's Algorithm
  select next vertex with smallest dist using priority queue.

Experiment 11

* write C program to implement following operations on binary tree : create, display.

→
```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int item;
    struct node* left;
    struct node* right;
};
void inorder (struct node* root)

    if (root == NULL)
        return;
    inorder (root → left)
    printf("%d", root → item);
    inorders (root → right);

void preorders (structnode *root)
{
    if (root == NULL)
        return;
    printf("%d", root → item);
    preorder (root → left);
    preorder (root → right);
}
void postorder (struct node *root)
~~void pos~~
```

```
{
    if (root == NULL)
        return;
    post order (root -> left);
    post order (root -> right);
    printf ("%d", root -> item);
}
struct node * createnode (item)
{
    struct node * newnode = mallocc
    size of (struct node));

    newnode -> item = item;
    newnode -> left = NULL;
    newnode -> right = NULL;
    return newnode;
}
struct node * insert at left (
    struct node * root int item)
{
    root -> left = create Node
                (item);
    return root -> left;
}
struct node * insert At right
    (struct node root, int item)
{
    root -> right = Create
                node (item);
    return root -> right;
```

```
int main()
{
    struct node * root = createnode (7);
    insert At left (root 15)
    insert At right (root, 6);
    insert At left (root -> left, 9);
    insert At right (root -> left -> 4)
    insert At right (root -> right -> 40)
    insert At left (root -> right -> 50)
    insert At left (root -> left -> left
                , 100);
    insert At right (root -> right ->
                right, 260);

    printf ("inorder traversal :");
    inorder (root);
    printf (" preorder traversal :");
    preorder (root);
    printf (" postorder traversal :");
    postorder (root);
```

2.
```
# include <stdio.h>
# include <stdlib.h>
struct node
{
    int item;
    struct node * left;
    struct node * right;
};
struct node * create node (item)
{
```

```
Structnode * newnode = malloc (size
    of (struct node));

newnode -> item = item;
newnode -> left = NULL;
newnode -> right = NULL;
return newnode;
}
struct node * insertatleft (struct node
  * root int item)
{
    root -> left = createnode (item);
    return root -> left;
}
struct node * insertatright (struct
  node * root, int item)
{
    root -> right = createnode
                (item);
    return root -> right;
}
```

Exp 12

1. Matrix:

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

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 (intarr [v][v])
{
    int i, j;
for (i = 0; i < v; i++)
{
    printf ("%d", i);
for (j = 0; j < v; j++)
{
        printf ("%d", arr [i][j]);
}
```

```
    printf("\n");
  }
}
int main()
{
  int adjmatrix[v][v];
  init(adjmatrix);
  insert edge(adjmatrix, 0, 1);
  insert edge(adjmatrix, 0, 2);
  insert edge(adjmatrix, 1, 2);
  insert edge(adjmatrix, 2, 0);
  insert edge(adjmatrix, 2, 3);
  insert edge(adjmatrix, 0, 3);
  insert edge(adjmatrix, 0, 0);
  insert edge(adjmatrix, 1, 3);
  insert edge(adjmatrix, 4, 3);
  insert edge(adjmatrix, 2, 4);

  print Adj matrix (adj.matrix);
  return 0;
}
```

O/P:

```
0 : 1 1 1 1 0
1 : 1 0 1 1 0
2 : 1 1 0 1 1
3 : 1 1 1 0 1
4 : 0 0 1 1 0
```

2. Linked list

```
#include <stdio.h>
#include <stdlib.h>
#define V 5
struct node {
    int dest;
  struct node *next;
}
struct graph {
    struct Node *adj(v);
};
  struct node *newnode *new
      node (int dest)
{
    struct node* temp =
    (struct node *)
malloc (size of (struct node));
  temp -> dest = dest;
  temp -> next = NULL;
  return temp;
}
struct graph * create graph
()
{
  struct graph * graph = (struct
graph *) malloc (size of
(struct graph))
  for (int i=0; i<v; i++)
    graph -> adj[i] = NULL;
    return graph;
}
```

```
void addedge (struct graph* graph, int src, int dest)
{
    struct node *temp = newnode (dest);
    temp -> next = graph -> adj (src);
    graph -> adj (src) = temp;

    temp = newnode (src);
    temp -> next = graph -> adj (dest);
    graph -> adj (dest) = temp);
}
void printgraph (struct Graph* graph)
{
    printf ("%d ->", i);
    struct node *temp = graph -> adj[i];
    while (temp)
    {
        printf("%d", temp -> dest);
        temp = temp -> next;
    }
    printf("\n");
}
}
int main()
{
    struct graph *graph = creategraph (5);
    addedge (graph (0, 1);
    addedge (graph (0, 2);
    addedge (graph (0, 3);
    addedge (graph (0, 4);
    addedge (graph (1, 3);
    addedge (graph (2, 3);
    addedge (graph (2, 4);
    addedge (graph (3, 4);

    printgraph (graph);

    return 0;
}
```

* Adjacent lists:

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

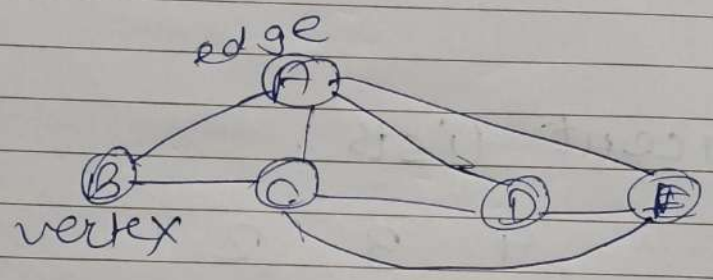
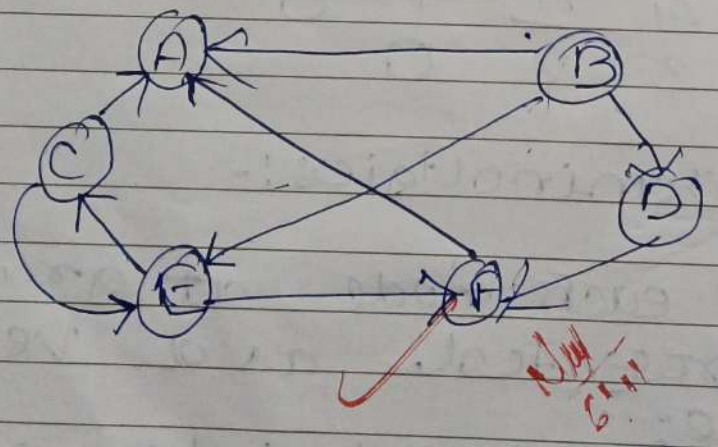
Q.2 Explain terminoligies:-

① vertex - each node at graph is represented as a vertex. In the examples, labeled circle is vertex.

② edge :-
edge represented a path between 2 vertices or a line between 2 vertices

3) adjacency :- Two node or vertex are adjacent if they are connected to each other through an edge.

Q3



Q.4