Extra

1) Declare an array of 5 subject accept marks from user and calculate the percentage.

2) Write a programme to accept number n and check wether n is e or o

3) Write a programme to print prime no 1 to 15

4) Write a menu driven programme using switch case

5) Write a programme to accept user to check wether the number is palyndrome or not.

```
1) # include <stdio.h>
   int main ()
   int marks [5];
   int total = 0;
   Float percentage ;

   printf("enter mark's for 5 subject(out of 100)\n");

   for(int i=0; i<5 ;i++) {
      printf("subject %d:", i+1);
      scanf("%d", &marks [i]);
```

```
if(marks[i] < 0 ||marks[i] >100) {
    printf("Invalid input \n");
    return 1;
}
total += marks[i];
}
percentage = (float) total /5;

printf("\n total marks = %d\n", total);
printf("percentage = %.2f %%.\n", percentage);

return 0;
}
```

output.

```
Enter marks for 5 subject.
subject 1 : 85
subject 2 : 98
subject 3 : 75
subject 4 : 45
subject 5 : 56

Total marks = 359
Percentage = 71.80%.
```

2)
```
# include <stdio.h>
int main()
{
 int num;
 printf("enter 1 number");
 scanf("%d",&num);
 if (num%2==0)
 {
  printf("the number is even");
 }
 else {
      printf("the number is odd");
 }
 return 0;
}
```

output -
Enter one number 5
The number is odd

PAGE No.
DATE / /

3)
```
# include <stdio.h
int main ()
{
int i,j,isprime;
printf("prime no 1 to 50 are \n");
for (int i=2; i<=50; i++)
{
isprime = 1;
for (j=2; j<=i; j++)
{
if (i % j == 0)
{
isprime = 0;
break;
}
}
if (isprime)
{
printf("%d\n",i);
}
}
printf("\n");
return 0;
}
```

output -

2
3
5
7
11
13
17
19
23
29
31
37
41
43
47

5) # include <stdio.h
int main ()
{
int num, reversed =0, original, remainder;

printf("enter a number ");
scanf("%d", &num);

original =num;

while (num !=0) {
remainder =num % 10;
reversed = reversed * 10 +remainder;
num /=10;
}

if (original == reversed){
printf("%d is a palidrome number\n", original);
} else {
printf("%d is not a palidrome number\n", original);
}
return 0;
}

PAGE NO.
DATE / /

Output
Enter a number : 121
121 is a palidrome number

PAGE No.
DATE / /

Experiment - 1

10.2.25

1) Find average of 10 number's using arr

code -

```
# include <stdio.h>
int main()
{
int numbers [10];
int sum = 0;
float average;

printf("enter 10 number \n");
for (int i=0; i<10; i++)
{
  printf("number %d: ", i+1);
  scanf("%d", &number [i]);
  sum += numbers [i];
}
average = (float) sum /10;

printf("\n sum = %d\n", sum);
printf("average = %.2f\n", average);

return 0;
}
```

output -
enter element of array : 5
enter element of array : 6
enter element of array : 2
enter element of array : 1
average is 8

PAGE No.
DATE / /

2) Disprog pattern

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

code -

```
#include <stdio.h>
int main()
{
int rows = 3;
for (i=1; i<=rows; i++)
{
  for (j=1; j<=rows ; j++)
  {
    if ( i%2==0)
    {
      printf ("*");
    }
    else
    {
      printf ("*");
    }
  }
  printf ("\n");
}
return 0;
}
```

output -

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

3) Find 1st repeating element from array

code:

```
#include <stdio.h>
int main()
{
int arr[10], flag=0;
for(int i=0; i<10; i++)
{
printf("enter element of array");
scanf("%d", &arr[i]);
}
for(int i=0; i<10; i++)
{
for(int j=0; j<10; j++)
{
if(j!=i)
{
if(arr[j]==arr[i])
{
printf("%d is repeating", arr[i]);
}
}
}
}
return 0;
}
```

PAGE No.
DATE / /

output:

```
enter element of array : 1
enter element of array : 2
enter element of array : 2
enter element of array : 3
enter element of array : 4
enter element of array : 5
enter element of array : 6
enter element of array : 7
enter element of array : 8
enter element of array : 9
2 is repeating number
```

4) Find greatest and smallest element of an array.

code.

```
# include <stdio.h>
  int main ()
{
  int i;
  int arr [5];
printf("enter element of array");
for(i=0; i<5; i++)
{
  scanf("%d", &arr [i]);
}
int max = arr [0];
int min = arr [0];
for (i=1; i<5; i++)
{
  if (arr [i] > max)
  {
    max = arr[i];
  }
  if (arr [i] < min)
  {
    min = arr [i];
  }
}
printf("greatest element is %d", max
```

```
printf("smallest element = %d\n", min);
  return 0;
}
```

output -
enter element -
1
3
2
4
5

greatest element = 5
smallest element = 1

5) Square odd elements of an array

code:

```
# include <stdio.h>
int main()
{
int arr[5];
printf("enter 5 number");
for (int i=0; i<5; i++)
{
scanf("%d", &arr[i]);
}
for (int i=0; i<5; i++)
{
if((i+1)%2 != 0)
{
arr[i] = arr[i] * arr[i];
}
}
printf("array after squaring element at
odd position");

for (int i=0; i<5; i++)
{
printf("%d", arr[i]);
}
printf("\n");
return 0;
}
```

output:
Enter 5 number's:
1
2
3
4
5
Array after squaring odd position:
1 2 9 16 25

11/7/25

17.7.25

Experiment - 2

1 Write a C programme on binary search on an sorted array to find an element

code-

```
#include <stdio.h>
int main ()
{
int arr [10], l, h, m, k;
printf("enter 10 sorted array element\n"
for(int i=0; i<10; i++)
{
scanf
```

EXPO -

Q Patterns -

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)
```
*
* * *
* * * *
```

code -

```
1) # include <stdio.h>
   int main ()
   {
   printf("enter number of rows"); int n;
   scanf ("%d", &n);
   printf("pattern 1:\n");
   for (int i=1; i<=n i++)
    {
      for (int j=1; j<=i ; j++)
      {
        printf ("%d", i);
      } printf ("\n");
    }
   printf("pattern 2\n");
   char arr = {*, #, @, $};
   for (int i=1; i<5; i++
    {
      for (int j=1; j<=1; j++)
      {
        printf("%c", arr[i-1]);
      } printf("\n");
    }
   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 ("pattern 4:\n");
for(int i=1; i<=n; i++)
{
  for (int j=1; j<=i; j++)
  {
    if (i%2!=0)
    {
      printf ("%d", i);
    }
    else
    {
      printf ("%d", j);
    }
  } printf ("\n");
}
printf ("pattern 5\n");
for (int i=1; i<=n; i++)
{
  for (int j=1; j<=n; j++)
  {
    printf (" ");
}
```

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

O/P

Enter number of rows : 5

pattern 1 :

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

Pattern 2 :

```
*
#  #
$  $  $
?  ?  ?  ?
```

Pattern 3 :

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

PAGE No.
DATE

Pattern 4:

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

Pattern 5:

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

2 Search data using binary search

Q1) Search data using linear search. consider following list to perform linear search.
56, 36, 89, 57, 1, 0, 67, 39

code-

```
# include <stdio.h>
  int main()
  {
   int arr[] = { 56, 36, 89, 57, 1, 0, 67, 39}
   int n = sizeof [arr];
  int k, f=0;
printf("enter the element to search:");
scanf("%d", &k);
for (int i=0; i<n; i++)
 {
   if arr[i] == k
    {
    printf("element found %d", i);
    f=1;
    break;
   }
}
if (f==0)
 {
  printf("element not found in the array");
 }
 return 0;
}
```

PAGE No.
DATE / /

O/P 1:
enter the element to search : 1
element found at index : 4

O/P 2:
enter the element to search : 55
Element not found in the array.

Q2) Search data using binary search

```
-> # include <stdio.h>
     int main()
   {
     int a[100], n, m, l, h, i, found =0, k;
   printf ("enter number of array element :");
   scanf ("%d", &n);
   printf ("enter key to be searched :");
   scanf ("%d", &k);
   printf ("\n enter sorted array element \n");
   for (i=0; i<n; i++)
     {
       scanf ("%d", &a[i]);
     }
   l=0; h (n-i);
   while (h>=l)
     {
       m(int) (l+h)/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])
{
    h=m-1;
}
}
}
if (h<l)
printf("\n element not found");
return 0;
}
```

O/P 1:

Enter number of array element : 4

Enter key to be searched : 34

Enter sorted array element :

1

2

3

4

Element not found

O/P 2:

Enter number of array element : 5
Enter key to be searched : 5

Enter sorted array element:
1
2
3
5
4

Element found at index : 3

Q3) Compare Linear search vs Binary search

->	Linear	Binary
1)	unsorted or sorted data	Must be sorted
2)	sequential scanning	Divide and conquer
3	O(n)	O(log n)
4	Less efficient	More efficient
5	Simple code	More complex code.

PAGE No.
DATE / /

Q4) State limitations of linear search in terms of time complexity:

A4) Inefficient for large data sets:
-> Linear search has a time complexity of O(n), where n is the number of element's in the list.

2 No improvement for sorted data:
-> Linear search does not take advantage of sorted data.

3 Average and worst case are the same order:
-> The average case and worst case both require scanning about half or all the element respectively so it doesn't perform on average.

4) Not suitable for time-critical application
-> Due to its linear time, it is not ideal for application's where fast searching is necessary, especially with large data set's.

17/7/25

Extra

1) Write a programe to copy the element of one array into another array in a reverse order

code -

```
# include <stdio.h>
  int main ()
{
  int i, n;
printf ("enter the number of elements
scanf ("%d", &n);

int original [n], reversed [n];
printf ("Enter %d elements :\n", n);

for (i=0; i<n; i++)
{
    scanf ("%d", &original [i]);
}
for (i=0; i<n; i++)
{
  reversed [i] = original [n-1-i];
}

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

PAGE No.
DATE / /

return 0;
}

O/P:
Enter the number of in the array: 5
Enter 5 elements:
1 2 3 4 5
Reversed array:
5 4 3 2 1

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

code-

```
# include <stdio.h>
  int main ()
{
  int arr [ ], n, i, j, count =0;

printf ("Enter the number of elements :");
scanf ("%d", &n);
printf ("Enter %d element :\n", n);
for (i=0; i<n; i++)
{
  scanf ("%d", &arr [i]);
}

for (i=0; i<n; i++)
 {
   for (j=i+1; j<n; j++)
   {
     if (arr [i] == arr [j])
     {
       count ++;
       break;
     }
   }
}
```

```
printf("Total number of duplicate
        element %d\n", count);

    return 0;
}
```

O/P-

Enter the number of element's in an array : 5

Enter the 5 element's

1 2 2 3 3

Total number of duplicate element : 4

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

code-

```
# include <stdio.h>
  int main ()
{
  int arr [100], n, i, j, count;

printf("Enter the number of element in the
                array : ");
scanf("%d", &n);

printf("Enter elements : \n", n);
for (i=0; i<n; i++)
{
    scanf ("%d", &arr [i]);
}
printf("Unique element in the array are
                : \n");
for (i=0 ; i<n; i++)
{
  count=0;
  for (j=0; j<n; j++)
{
  if ( arr [i] == arr [j])
  {
      count ++;
  }
}
```

```
printf("\n");
return 0;
}
```

O/P -

```
Enter the number of element's in the
    array : 5
Enter 5 elements :
 1   2   3   4   4
Unique element in the array are :
 1   2   3
```

4 WAP to seperate odd and even integer's into seperate array

code -

```
# include <stdio.h>
    int main ()
{
 int arr [100], even [100], odd [100];
 int n, i;
 int evencount=0, odd count =0;

 printf("Enter the number element in the
        array :");
 scanf("%d",&n);

 printf("Enter %d integers :\n",n);
 for( i=0; i<n; i++)
 {
   scanf("%d",arr [i]);
 }

 for( i=0; i<n; i++)
 {
  if (arr [i] %2 ==0)
  {
   even [evencount ++] =arr [i];
  } else {
    odd [odd count ++]=* arr [i];
  }
 }
```

```
printf("Even number :");
for(i=0; i<evencount; i++)
{
  printf("%d", even[i]);
}
printf("\n");

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

O/P -

```
Enter the number element in the array :10
Enter 10 integers:
1  2  3  4  5  6  7  8  9  10
Even number's : 2  4  6  8  10
odd number's : 1  3  5  7  9
```

5 WAP to
in an a

code-

```
# includ
# inclu
  int m
 {
int arr
int n =
int i, k

for( i
{
  if (a
  {
    sec
    Fir
  } el

  }
  }
if (
{
  pr
else
{
  pr
```

5 WAP to find second smallest element in an array-

code-

```
# include <stdio.h>
# include < limit's.h>
  int main()
  {
int arr[] = { 5, 1, 9, 6, 3 };
int n = sizeof(arr)/sizeof(arr[0]);
int i, first = INT_MAX, second = INT_MAX

for( i=0; i<n; i++)
{
  if (arr[i] < first)
  {
    second = first;
    first = arr[i];
  } else if (arr[i] < second && arr[i] !
                                = first

  {  second = arr[i];
  }
}
if ( second == INT_MAX
{
  printf("No smallest element:\n");
else
{
  printf("second smallest element is
                        %d\n");
```

PAGE No.
DATE / /

return 0;
}

O/P.
second smallest element is 3

6) WAP to count the total no of word's in a string.

code-

```
# include <stdio.h>
  int main ()
{
  char str [1000];
  int i =0, wordcount =0;
  int inword =0;

  printf ("Enter a string :\n");
  fgets (str, sizeof (str), stdin);

  while (str [i] != '\0') {

   if (str [i] == ' ' || str [i]== '\t'
              || str [i] == '\n')

   {
     inword = 0;
   } else if (inword ==0)
     {
       inword =1;
       wordcount ++;
     }
     i++;
  }
  printf ("Total number of words : %d\n",
                wordcount);
```

return 0;
}

O/P
Enter a string :
bhushan Thite
Total number of words : 2

Experiment - 3

1) Sort elements in ascending order using bubble sort:-

code:-

```
# include <stdio.h>
  int main ()
{
  int arr [100], i, j, n, temp;
  printf("Enter the no of elements:");
  scanf("%d", &n);

  for (i=0; i<n; i++)
{
  printf("Enter elements:", n);
  scanf("%d", arr [i]);
}
  for (i=0; i<n; i++)
{
  for (j=0; j<n-i-1; j++)
{
  if (arr [j] > arr [j+1])
{
    temp = arr [j];
    arr [j] = arr [j+1];
    arr [j+1] = temp;
}
}
}
```

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

O/P-

```
Enter number of elements : 5
Enter Elements : 5
 —          — : 3
 —          — : 4
 —          — : 1
 —          — : 2
The sorted array is
1
2
3
4
5
```

2) Sort element in decending order using secention order.

code-

```
# include <stdio.h>
  int main ()
{
  int arr [100], i, n, j, temp;
  printf ("Enter number of elements:");
  scanf ("%d", &n);
  for ( i=0; i<n; i++)
  {
    printf ("Enter elements :", n);
    scanf ("%d", &arr [i]);
  }
  for (i=0; i<n; i++)
  {
    for ( j=i+1 ; j<n ; j++)
    {
      if (arr [j] >arr [i])
      {
        temp = arr [i];
        arr [i] = arr [j];
        arr[j] = arr [i];
      }
    }
  }
  printf ("Ther sorted array is :\n");
```

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

O/P:

```
Enter number of elements : 5
Enter element's : 23
—          —      : 1
—          —      : 55
—          —      : 49
—          —      : 8
The sorted array is:
55
49
23
8
1
```

3) Find the number of comparisons required in bubble sorte method of the following list having 5 number 100, 200, 300, 400, 500

The give list - 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

Pass 2

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

Pass 3

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

PAGE No.
DATE / /

Pass 4 100 200 300 400 500

100 $\boxed{200}$ $\boxed{300}$ $\boxed{400}$ $\boxed{500}$

∴ The sorted list is -
100, 200, 300, 400, 500

∴ Number of comparisons = 10

4)

4) Sort the given array in ascending order using selection sort method and show digramatic representation of every iteration of for loop : 500, -20, 30, 14, 50

Given list - 500, -20, 30, 14, 50

500 -20 30 14 50

-20 500 30 14 50

-20 500 30 14 50

-20 500 30 14 50

$\boxed{-20}$ 500 30 14 50

-20 500 30 14 50

-20 30 500 14 50

-20 14 500 30 50

$\boxed{-20}$ $\boxed{14}$ 500 30 50

-20 14 500 30 50

-20 14 30 500 50

$\boxed{-20}$ $\boxed{14}$ $\boxed{30}$ 500 50

PAGE No.
DATE / /

500 50

30

14

-20

50 500

30

14

-20

∴ The sorted list is -
-20, 14, 30, 50, 500

```
for (i=0; i<n; i++)
{
    for (j = i+1; j<n; j++)
    {
        if (arr[j] > arr[i])
        {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = ~~arr~~ temp;
        }
    }
}
```

11/9

PAGE No.
DATE / /

Experiment - 4

Q1) Sort elements 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", &arr[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] = 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;
}
```

O/P-

Enter the no 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

Q2) Sort elements in ascending order 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 element:");
scanf("%d", &n);
printf("enter %d number:\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 ) %. a] ++;

  for(i=1 ; i<0; i++)
   count [i] + = count [i+1];
  for (i=n-1; i>=0 ; i--)
{
```

```
int digital = (a[i]/price) % 10
output[--count[digital]] = a[i];
}
for (i=0; i<n; i++)
  printf("%d", a[i]);
  return 0;
}
```

o/p

3) What is the output of the insertion sort after the second insertion iteration of the following number.

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

A)

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

Pass2)

	0	1	2	3	4	5	6	7	8
100	100								
390									
4130				4130					
5431				5431					
225			225						
956						956			
99									

Pass3)

	0	1	2	3	4	5	6	7	8
100		100							
225			225						
4130		4130							
5431					5431				
956									
390				390					
99	99								

Pass4)

	0
99	99
100	100
4130	
225	225
390	390
5431	
956	956

Sorted

5431

2)

Pass1)

25	
6	
99	
143	
239	
20	
18	

Pass 2

20	
23	
143	
6	
18	
99	
239	

(pass 1)	0	1	2	3	4	5	6	7	8	9
99	99									
100	100									
4130					4130					
225	225									
390	390									
5431						5431				
956	956									

sorted arr = [0, 99, 100, 225, 390, 956, 4130, 5431]

2)

(pass 2)	0	1	2	3	4	5	6	7	8	9
25						25				
6							6			
99										99
145						145				
239										239
20	20									
18		18								

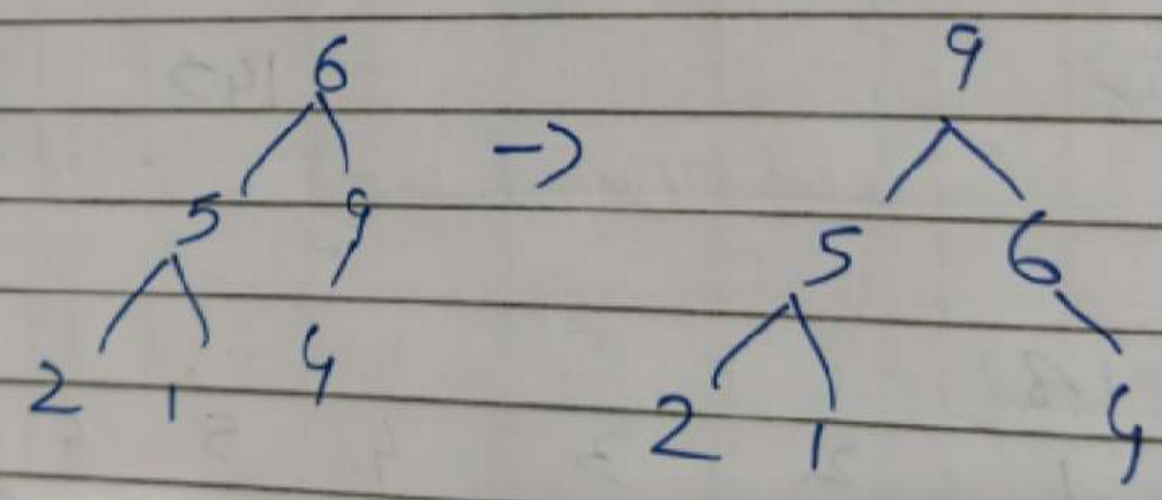
(pass 2)	0	1	2	3	4	5	6	7	8	9
20			20							
25			25							
145					145					
6	6									
18		18								99
99										
239				239						

		1	2	3	4	5	6
Pass 3	6						
6	6						
18	18						
20	20						
25	25						
239			239				
145		145					
99	99						

sorted arr = [6, 18, 20, 25, 99, 145, 239]

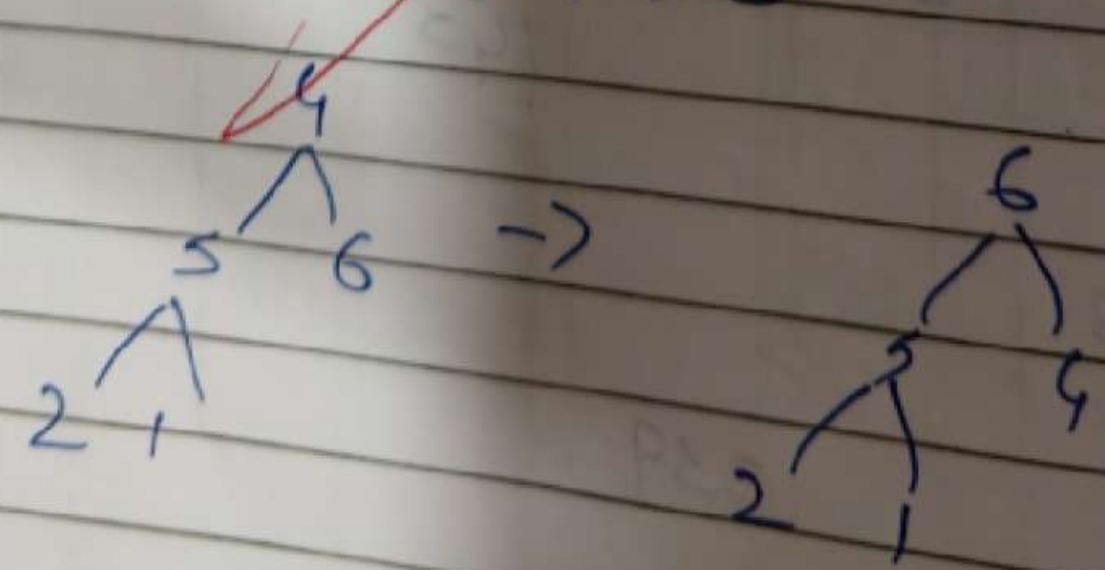
Q4) Sort the following elements using heap sort -

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



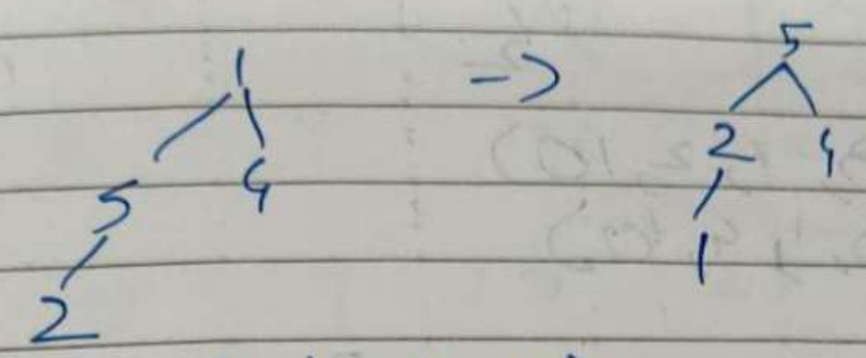
arr = [9, 5, 6, 2, 1, 4]
arr = [4, 5, 6, 2, 1, 9]

Pass 2)



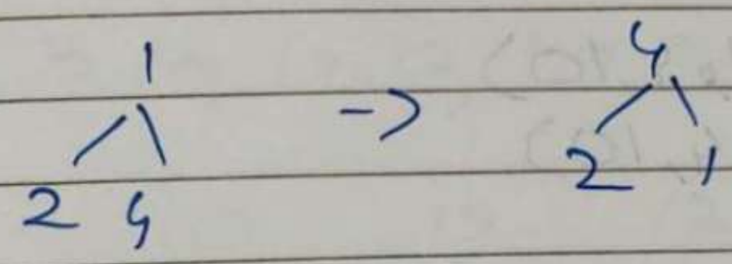
arr = [5, 5, 4, 2, 1, 9]
arr = 1, 5, 4, 2, 6, 9]

pass 3



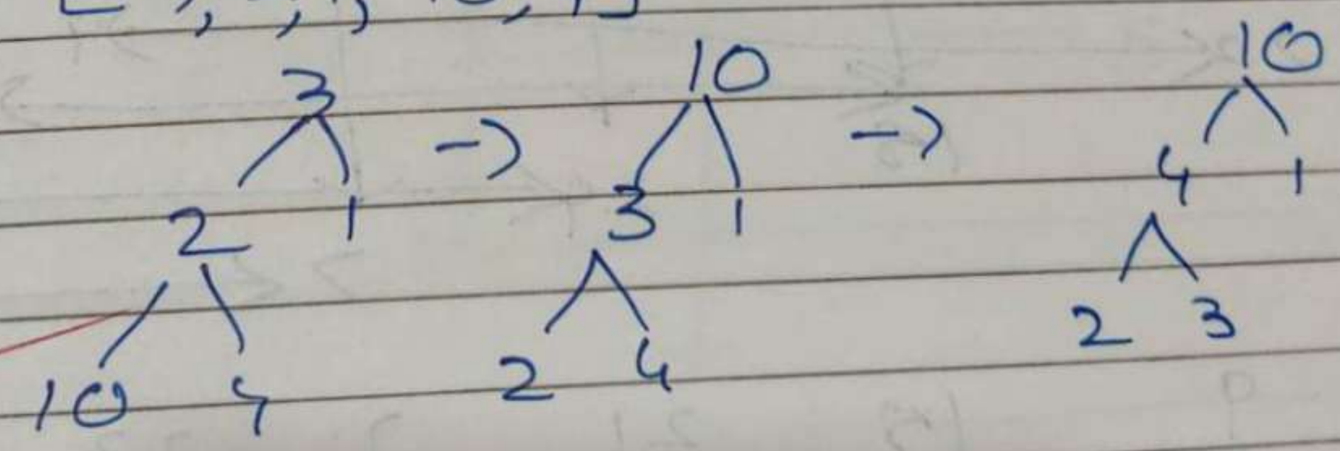
arr = [5, 2, 4, 1, 6, 9]
arr = [1, 2, 4, 5, 6, 9]

pass 4



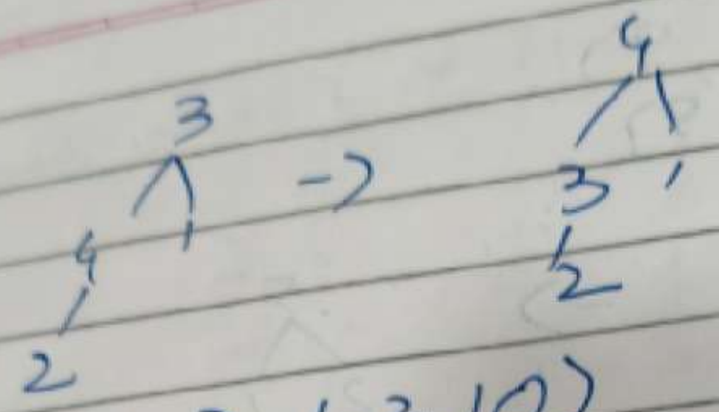
arr = [4, 2, 1, 5, 6, 9]
arr = [1, 2, 4, 5, 6, 9]

2) arr = [3, 2, 1, 10, 4]



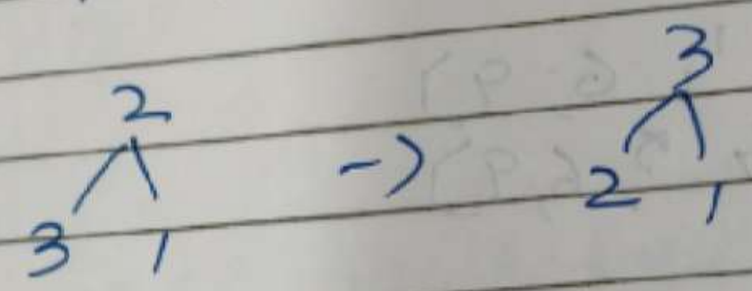
arr = [10, 4, 1, 2, 3]
arr = [3, 4, 1, 2, 10]

Pass2)



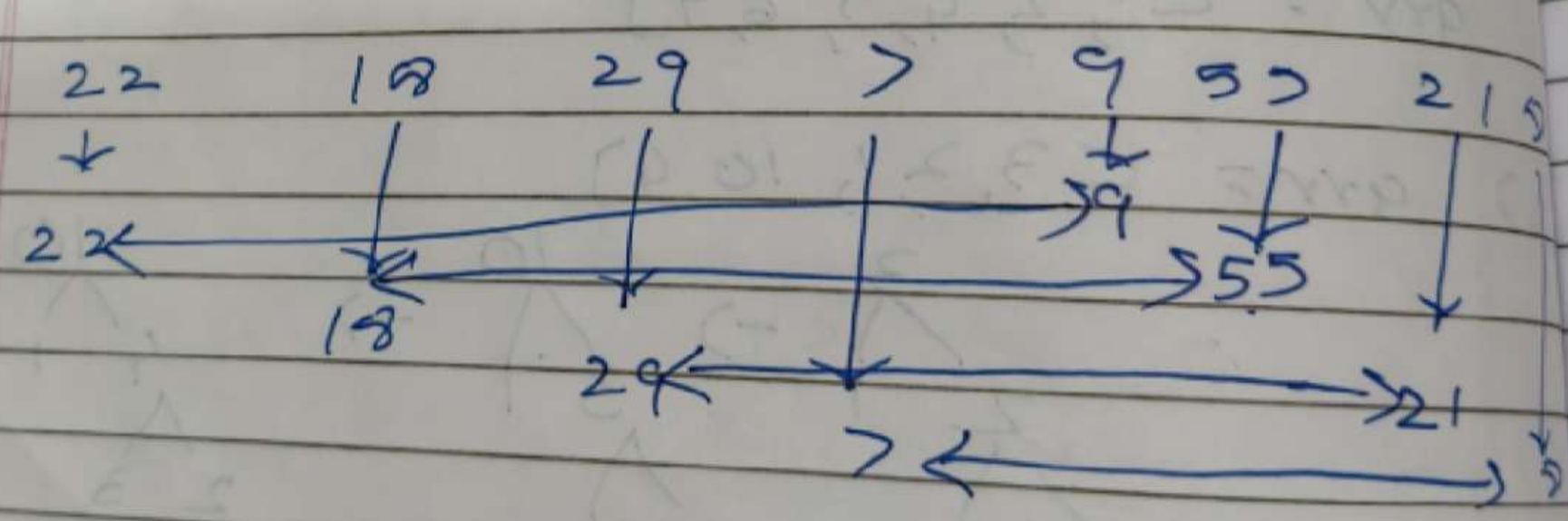
arr = [4, 3, 1, 2, 10]
arr = [2, 3, 1, 4, 10]

Pass3

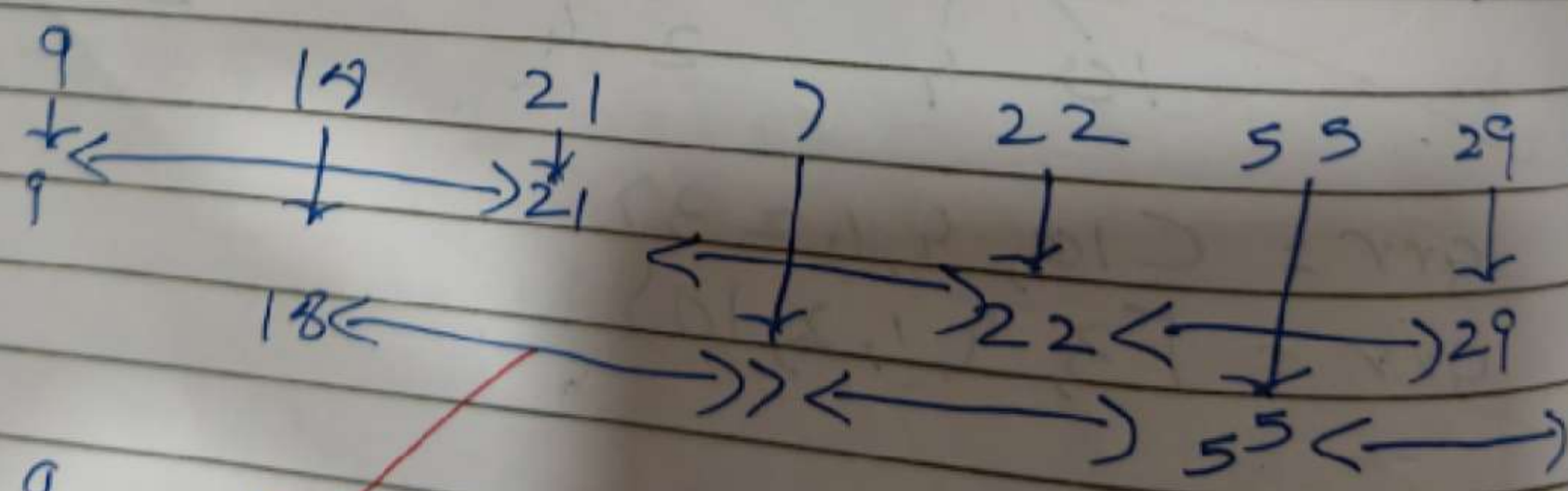


arr = [3, 2, 1, 4, 10]
arr = [1, 2, 3, 4, 10]

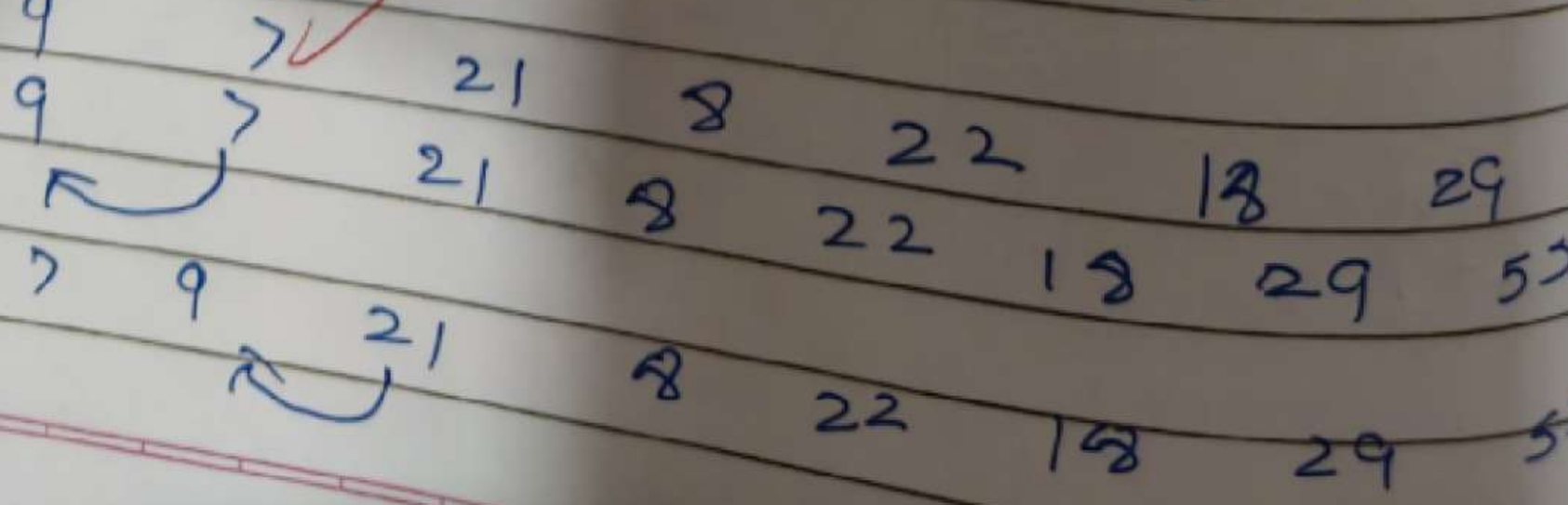
Q5) sort following elements using shell sort



Pass2



Pass3



7 9 21 8 22 18 29 55

7 8 9 21 22 18 29 55

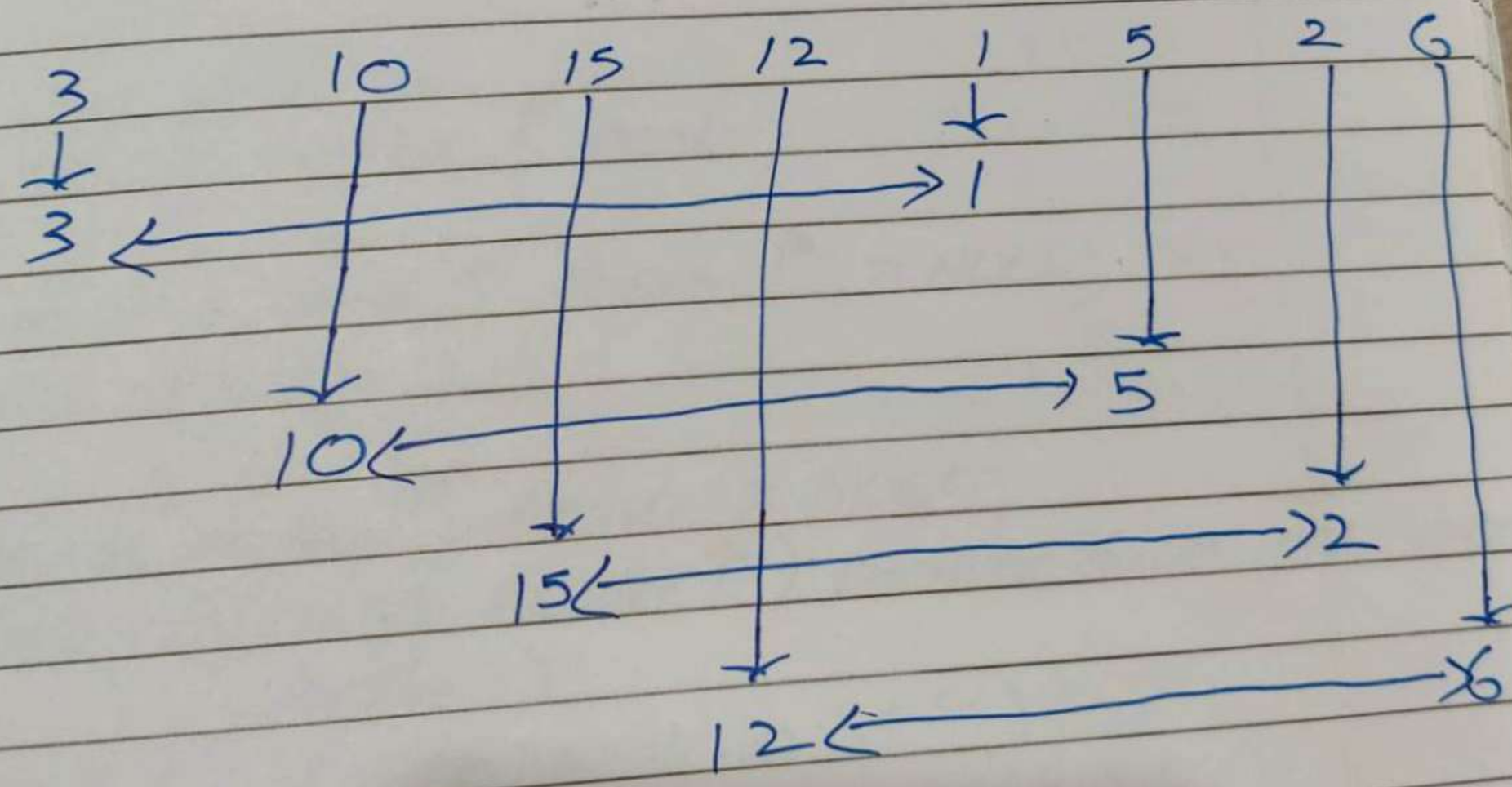
7 8 9 21 22 18 29 55

7 8 9 18 21 22 29 55

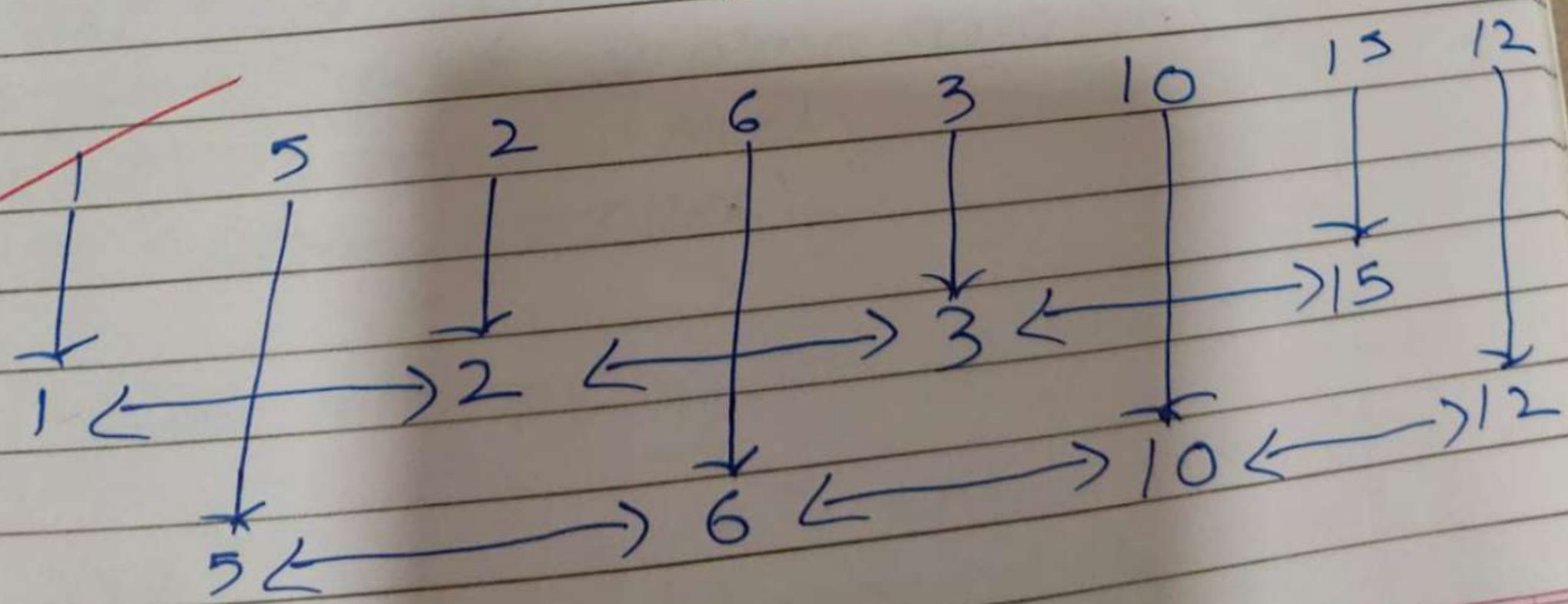
sorted arr = 7, 8, 9, 18, 21, 22, 29, 55

2) arr = [3, 10, 15, 12, 1, 5, 2, 6]

pass1)



pass2)



PAGE No.
DATE / /

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

E

Q1

Experiment -5

Q1) Write a menu driven programme that implements singly linked list for the following operations create display insert begin insert end insert mid delete begin end delete position.

```
#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 (size of struct
        node));
printf ("\n enter data \t");
scanf ("%d", &temp data \t");
temp -> next = NULL;
if (start == NULL)
{
  start = temp;
}
else
{
  ptr = start;
```

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

```
printf("\n Enter data:");
scanf("%d", &temp->info);
temp->next = start;
start = temp;
}
void insert_mid()
{
 struct node * temp, * ptr;
 int pos, i;
 printf("\n Enter position to insert:");
 scanf("%d", &temp->info);
 if (pos==1)
 {
    temp->next = start;
    start = temp;
    return;
 }
 ptr = start;
 for (i=1; i<pos-1 && ptr != NULL; i++)
 {
    ptr = ptr->next;
 }
 if (ptr==NULL)
 {
  printf("\n position or range \n");
  free(temp);
 }
 else
 {
  temp->Next = ptr->next;
  ptr->next = temp;
 }
```

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

```
start = start -> next;
printf("\n deleted array element:%d\n", temp->info);
free (temp);
}
void delete_end()
{
 struct node *, temp, *ptr;
if (start == NULL)
{
 printf("List is empty");
 return;
}
if (start -> next == NULL)
{
 printf("\n delete element:%d", start->info);
 free (start);
 start = NULL;
 return;
}
ptr = start;
while (ptr -> next -> ! = NULL)
{
   ptr = ptr = ptr -> next;
}
temp = ptr -> next;
printf("deleted element:%d", temp->info);
free(temp);
ptr -> next = NULL
}
```

```
void delete_pos()
{
struct node * temp, *ptr;
int pos, i;
if (start ==NULL)
{
printf("List is empty");
return;
}
printf("enter position to delete");
scanf("%d", &pos);
if (pos==1)
{
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, i++)
{
ptr = ptr->next;
}
if (ptr==NULL || ptr->next ==NULL)
{
printf("position out of range");
return;
}
temp= ptr->next;
ptr->next =temp->next;
```

```
printf("\n deleted element : %d\n", temp->info);
free (temp);
}
int main ()
{
  int choice;
  while (1)
  {
  printf("\n Menu :\n 1: create \n 2: Display \n
  3: insert begin \n 4: Insert end \n 5: Insert mid \n
  6 : Delete begin \n 7: Delete end \n 8: Delete pos
  \n : 9: exist );
  printf("\n enter your choice: ");
  scanf("%d", &choice);
  switch (choice)
  {
    case 1:
          create ();
          break;
    case 2:
          display ();
          break;
    case 3:
          insert - begin();
          break;
    case 4:
          insert -end ();
          break;
    case 5:
          insert -mid();
          break;
```

```
case 6 :
        delete_begin ();
        break;
case 7 :
        delete_end ();
        break;
case 8 :
        delete_pos ();
        break;
case 9 :
        exit (0);
default :
        printf ("\n invalid choice");
    }
}
return 0;
}
```

Q2) Explain the basic terminologies of single linked list with example.

-> The basic terminologies of single linked list are -

1) Node -

A node is the basic building block of a linked list

eg - struct node {
int data;
struct Node * next;
};

2) Head -

The head is a pointer / reference to the first node of the linked list. It is used to traverse or access the entire list.

eg -

Head -> Node1 -> Node2 -> Node 3 -> NULL

3) Data field -

The data field holds the value or information of a node

eg -

Node1 : data = 10
Node2 : data = 20

4) Next link
The next link contains the address / reference of the next node in the sequence. If it is the last node, this field stores NULL

eg -
Node. 1 next -> Node 2
Node. 2 next -> node 3
Node 3 next -> NULL

5) NULL
NULL signifies the end of the linked list. The next of the last node always points to NULL.

6) Traversal
Traversal means visiting each node of the linked list one by one, starting from head until NULL is reached.

eg -

```
struct * Node temp = head;
while (temp != NULL) {
    printf("%d", temp->data);
    temp = temp->next;
}
```

7) Insertion:
adding a new node to the list.

eg -
Before : 10 -> 20 -> 30 -> NULL
Insert 15 after 10
10 -> 15 -> 20 -> 30 -> NULL

8) Deletion :
Removing a node from a list
eg -
Before : 10 -> 20 -> 30 -> NULL
Delete : 20
10 -> 30 -> NULL

Q3) Difference between arrays and linked list

Arrays	Linked list
1) Uses contiguous memory	Uses non contiguous memory
2) Fixed size	Dynamic size
3) Insertion and deletion in the middle requires shifting element, so it is slower	Insertion or deletion node is faster if the pointer to the location is known
4) Easy to implement	More complex
5) Less memory	More memory
6) Direct access	sequential access

PAGE No.
DATE / /

Experiment - 6

Write a menu driven C programme that implement's doubly linked list for the following operation: Create, Display, insert a node, delete a node.

code-

```
# include <stdio.h>
# include <stdlib.h>
struct node
{
    int info;
    struct node * next;
    struct node * perv;
};
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 -> perv = NULL;
if (start == NULL)
{
    start = temp;
}
else
```

```
ptr = start;
while (ptr->next != NULL)
{
    ptr = ptr -> next;
}
  ptr -> next = temp;
  temp -> perv = ptr;
  }
}
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 (struct node));
    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 ("\n Enter position 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 position out of range !\n");
        free (temp);
    }
    else
    {
        temp -> next = ptr -> next;
        temp -> pev = ptr;
        if( ptr -> next != NULL)
        {
            ptr -> next -> pev = temp;
            ptr -> next = temp;
        }
    }
}
void insert end ()
{
    struct node * temp, * ptr;
temp = (struct node*)malloc(sizeof (struct node));
printf("\n Enter data: ");
scanf("%d", &temp -> info);
temp -> next = NULL;
temp -> pev = 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)
    {
        printf ("\n List is empty \n");
        return;
    }
    temp = start;
    start = start -> next;
    if ( start ! = NULL)
    {
        start -> prev = NULL;
printf("\n Deleted element : %d\n", temp->info);
    free (temp);
}
```

```
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);
}
void delete pos ()
{
 struct node *temp, *ptr;
 int pos, i;
```

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

```
printf("\n Deleted element : %d\n", ptr->info);
free(ptr);
}
int main ()
{
  int choice;
  while (1)
  {
  printf("\n Menu : \n 1. Createn 2. Disping
  \n 3. Insert Begin \n 4. Insert End \n 5.
  Insert Mid \n 6. Delete Begin \n 7. Delete
  End \n 8. Delete position \n 9. Exit");

  printf("\n Enter your choice");
  scanf("%d", &choice);
  switch (choice)
  {
   case 1:
      create ();
      break;
   case 2:
      display ();
      break;
   case 3:
      insert begin ();
      break;
   case 4;
      insert end ();
      break;
```

```
case 5:
    insert mid()
    break;
case 6:
    delete begin()
    break;
case 7:
    delete end()
    break;
case 8:
    delete pos()
    break;
case 9:
    Exit (0)
default:
    printf("\n Invalid choice\n");
  }
}
return 0;
}
```

Op-

Menu:

1. Create
2. Display
3. Insert Begin
4. Insert End
5. Insert mid

6. Delete Begin
7. Delete End
8. Delete position
9. Exit
Enter your choice : 1
Enter data : 23

Menu:
Enter your choice : 2
List element are : 23

Menu:

Enter your choice : 3
Enter data : 45

Menu:
//
//
//
//
//
//
//
//
//
//
//
//
//
//

Enter your choice : 4
Enter data : 90

~~Menu:~~

Q2) Write a short note on circular linked list

A2) A circular linked list is a variation of a linked list in which the last node points back to the first node forming a circle. It can be singly or doubly linked -

1) Singly - The next pointer of the last node points to the first node.

2) Doubly - The next of the pointer points to the first node and prev of the first node points to the last node.

Key Feature.

- NO NULL at the end ; traversal can starts at any node and continue indefinitely.
- can efficiently move both forward without coming about the end of the list.

Advantages.

- can traverse the entire list from any point
- Useful for implementing circular queues.
- Easy to insert nodes at the beginning or end.

17/10

PAGE No.
DATE / /

Experiment -7

Write a programe to perform primitive operation's on 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 in 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 an choice:\n push:\n pop:\n display \n exit \n");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1:
            push();
            break;
```

```
case 2:
        pop ();
        break;
case 3:
        display ()
        break;
case 4:
        printf("\n exiting");
        break;
}
}while (choice != 4);
return 0;
}
```

O/P -

enter ur choice :

1. push
2. pop
3. display
4. exit

1

enter element to enter in stack : 10

17/10

Experiment - 8

converting infix to postfix or prefix 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))
    {
      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;
}
```

o/p:
enter expression : +*abc-*

+abc+*-

2) convert infix to prefix using stack

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

Input	stack	output
H	*	H
*	*)	H
)	*)G	H
G	*)*	HG
*	*)*)	HG
)	*)*)F	HGF
F	*)*)^	HGF
^	*)*)^E	HGFE
E	*)*)^/	HGFE^
/	*)*)/D	HGFE^D
D	*)*)/(	HGFE^D/
(	*)-	HGFE^D/*
-	*)-(	HGFE^D/*(
(	*)-*	HGFE^D/*(B
*	*)-*C	HGFE^D/*(B*
B	*)-*	HGFE^D/*(B*-
*	+	HGFE^D/*(B*-*
C	+	HGFE^D/*(B*-*A
-		
*		
/		
^		
D		
E		
F		
G		
H		

+A*-*BC*/D^EFGH

3) Evaluate

+ - *

Input

2
4
4
1
2
4
1
2
1
+
+

3) Evaluate prefix expression using stack

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

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

4) Write a c programe to evaluate infix or postfix 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))
    {
      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;
}
```

O/P -

enter expression : 1-+*765

1-765*+

12/10

Experiment -9

1) perform primitive operation's on linear queue like insert delete display.

code -

```
#include <stdio.h>
# include <stdlid.h>
# define size 5
int q [size];
int front = -1, rear = -1;
void insert ()
{
int value;
if( rear == size -1)
{
printf("overflow\n");
}
else
{
printf("enter value to insert: ");
scanf("%d", &value);
if (front ==-1)
    front =0;
rear ++;
q [rear] = value,
printf("Inserted %d\n", value);
}
}
```

```
void delete ()
{
  if (front == -1 || front > rear)
  {
    printf("queue is empty\n");
  }
  else
  {
    printf("deleted %d\n", Q[front]);
    front ++;
  }
}
void display ()
{
  if (front == -1 || front > rear)
  {
    printf("queue is empty\n");
  }
  else
  {
  printf("queue element: ");
  for (int i = front; i <= rear; i++)
  {
    printf("%d", Q[i]);
  }
    printf("\n");
  }
}
```

```
int main ()
{
 int choice;
while (1)
{
printf("\n 1. Insert \n 2. Delete \n 3.
Display \n 4. Exist \n");
printf("enter your choice);
scanf("%d", &choice);
switch (choice)
{
case 1: insert ();  break;
case 2: delete ();  break;
case 3: display ();  break;
case 4: Exist ();  break;
default : printf("Invalid choice\n:");
 }
}
return 0;
}
```

O/P -

1. Insert
2. Delete
3. Display
4. Exist

enter your choice : 1

Enter value to insert : 10

Inserted 10

PAGE No.
DATE / /

1. " "
2. " "
3. " "
4. " "

enter your choice : 1
enter value to insert : 20
inserted 20

1. " "
2. " "
3. " "
4. " "

enter your choice : 3
queue elements : 10 20

2) Perform following operation on linear queque in array size (10)

i) Insert (10)

10									

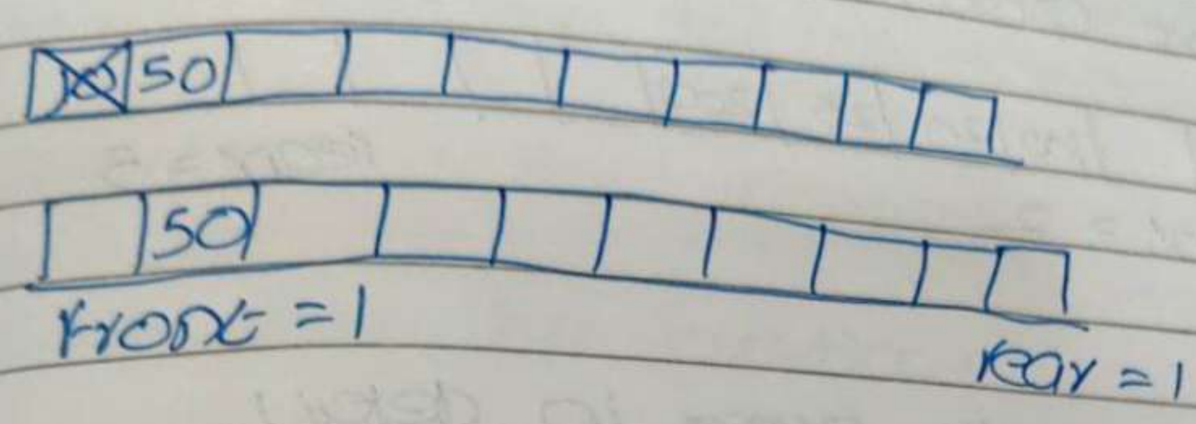
Front = 0 Rear = 0

ii) Insert (50)

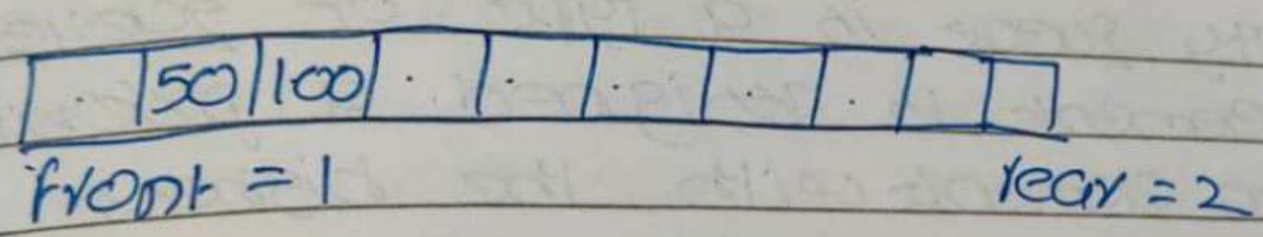
10	50								

Front = 0 Rear = 1

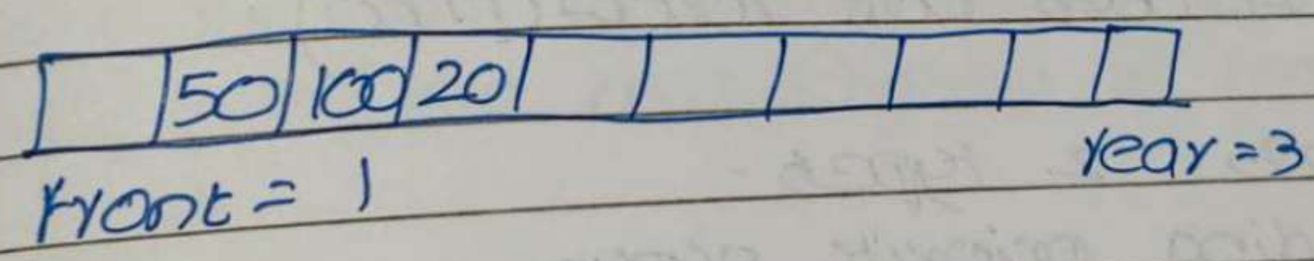
iii) Delete



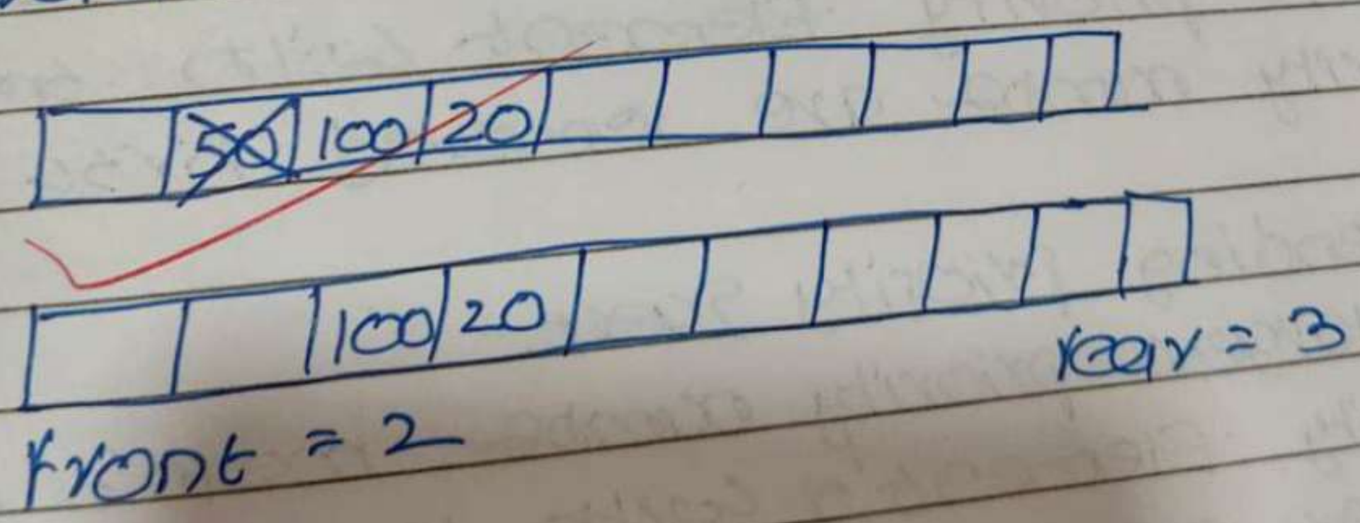
iv) Insert (100)



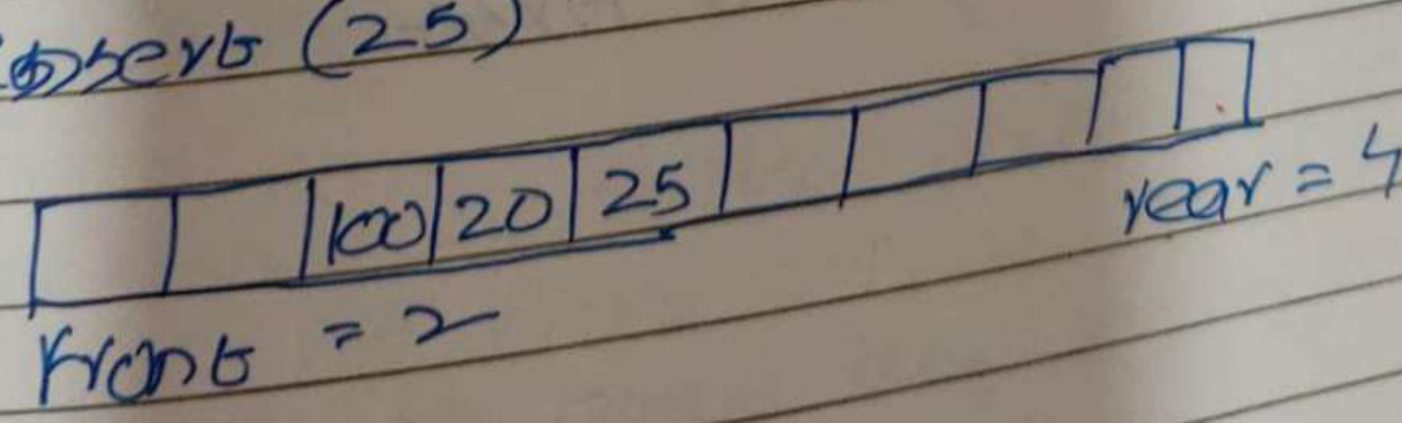
v) Insert (20)



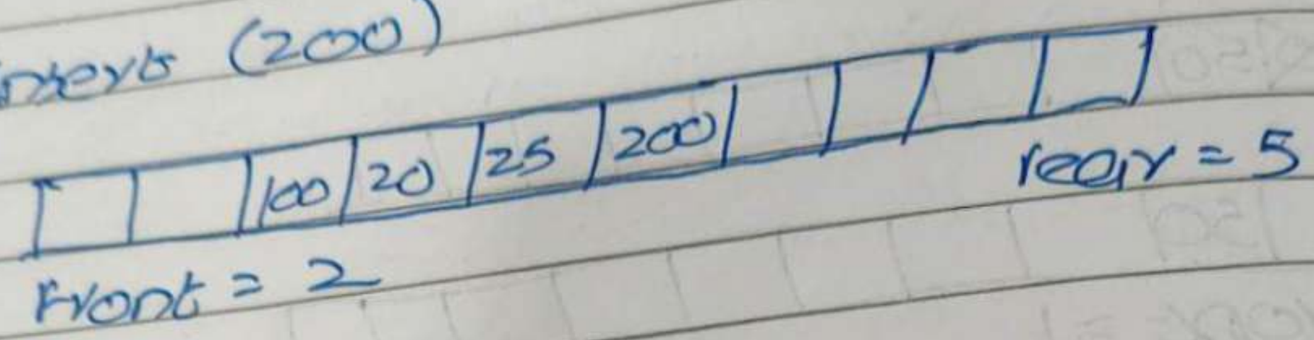
vi) Delete



vii) Insert (25)



viii) Insert (200)



3) Explain priority queue in detail

-> Priority queue is a type of queue where each element is assigned a priority and the element with the highest priority is deleted before other elements regardless of the order they were inserted. It does not follow (FIFO)

It has 2 types.

1) Ascending priority queue.
The smallest priority number mean highest priority. Element with smaller priority number are removed first

2) Descending priority queue.
The largest priority number mean's highest priority. Element's with larger priority number are removed first.

4) Write a algorithm for insertion and deletion operation for linear queue.

Algorithm - For insertion

Step 1 -
Check for the overflow

Step 2 -
Check if the queue is empty in case the queue is empty then both front and rear are set to 0 so that the new value can be stored at the 1o location. Otherwise if the queue already has some value the rear is increamented so that it points to the next location in the array

Step 3 -
The value is stored in the location pointed by the rear

Step 4 -
Exit

For deletion -
Step 1 -
Check for underflow condition an underflow occur's if front = -1 or front > rear however if queue has some value then

Front is increamented so that it now point's to the next value in the queue.

Step 2 -
Exist

17/10

Experiment-10

1) primitive operation on circular queue- Insert, Delete, Display.

code-

```
# include <stdio.h>
# include <stdlib.h>
# define SIZE 5
int queue [size];
int r =-1 , rear = -1;
void insert ()
{
 int value;
 ;r ((r==0 && r == size-1)|| (rear+1)
 -1. size ==r)
 {
   printf("queue is full \n");
 }
 else
 {
   printf("enter value to insert:");
   scanf("%d", &value);
   ;r (r==-1)
   {
     r=0;
     r=0;
   }
 else
```

```
{
    r = (r+1) % size;
}
queue [r] = value;
printf ("%d\n", value);
}
};
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;
    }
    }
}
void display ()
{
```

```
if (r==-1)
{
  printf("Queue is empty\n");
  int i = f;
  while (1)
  {
    printf("%d", queue[i]);
    if (i==r)
      break;
    i = (i+1) % size;
  }
  printf("\n");
}
int main()
{
  int choice;
  while (1)
  {
    printf("\n 1. Insert \n 2. Delete \n 3. Display \n 4. Exist \n");

    printf("enter your choice :");
    scanf("%d", &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;
}
```

O/P:

1. Insert
2. Delete
3. Display
4. Exits

Enter your choice : 1

Enter element to insert : 10

2) What are advantages of circular queue show diagramatic representation.

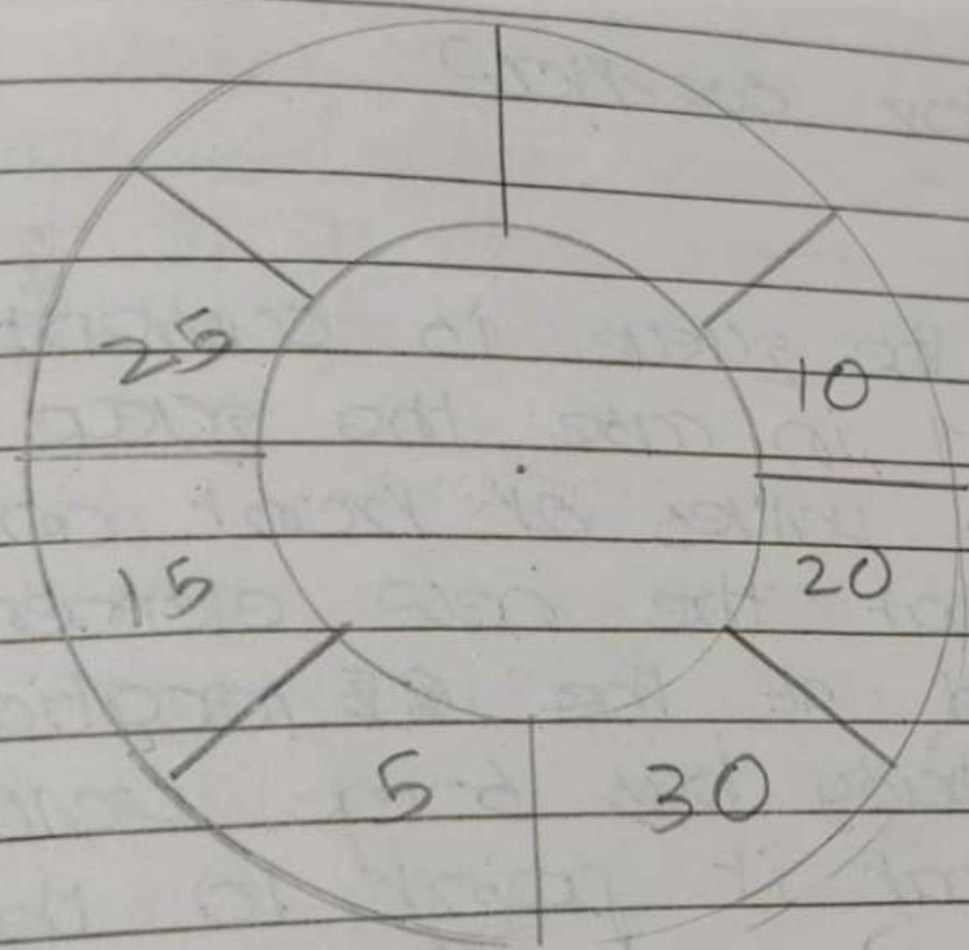
->1) Efficient memory utilization -
Unlike linear queue, space is reused after deletion.

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.
works well for fixed size memory buffers

4) Faster operation -
Insertion and deletion are O(1)

Diagram -



10	20	30	5	15	25

Front

Front = 0

rear

rear = 5

3) Algorithm - insert, delete for circular queue -

Insertion -

Step 1 -
Check for overflow

Step 2 -
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 pt r, if the rear already has some value increment it so that it point to the next position.

Step 3 -
The value inserted is stored at the position pointed by the rear

Step 4 -
Exit

Deletion -
Step 1 -
Check for underflow

Step 2 -
If the value of front is less than the value of rear then it means that the queue has some value the front is incremented.

Step 3 -
Remove the element at the front and report deletion. If the element at deleted was the last element then ~~f=r~~ front = rear if not advances for the next element front = front +1 and set front = 1 and rear = -1

Step 4 -
Exit.

4) Write and explain the application of queue -

Application of queue are -
1) Data transmission -
In networks, queue are used in router and switches.

It prevents data loss

2) Breadth First search in graphs.
BFS algorithm uses a queue to explore nodes each level by level.

3) call center's.
The first caller who entered the queue get connected to the next available operator

4) Amusement park / Hospital
people are waiting for rides in amuse-ments park or for a doctor's appointment are served in queue.

5) Differentiate between- stack and queue

stack	queue
1) It work on LIFO	It work's on FIFO
2) Insertion happens at the top	Insertion happens at the rear
3) Deletion happen's at the top	Deletion happen's at the front

4) Only top element is accessible	Both front and rear element are accessible
5) Traversal is not possible	Traversal is possible
6) can be implemented using array or linked list	can be implemented using array linked list or circular array.

17/10

PAGE No.
DATE / /

experiment -11

Q1) operation on binary tree.

```
-> #include <stdio.h>
   #include <stdio.h>
   struct node {
      int data;
      struct node * left;
      struct node * right;
   }
   struct node *create (int val);
   struct node *insertleft(struct node *root, int val);
   struct node *insertright(struct node *root, int val);
   void deletesubtree(struct node *root);
   void deleteleft(struct node *root);
   void deleteright(struct node *root);
   void display(struct node *root);

   int main ()
   {
   struct node *root = create (1);
   insertleft (root, 2);
   insertright(root, 3);
   insertleft (root->left, 4);
   insertleft(root->right, 5);
   insertleft(root->right, 6);
   insertright(root->right, 7);
   insertleft(root->left, 8);

   printf("original tree:");
```

```
display (root);
printf("\n");
deleteleft (root)
printf("tree after deleting left subtree of
root");
display (root);
return 0;
}
struct node * create (int val){
  struct node *new=(struct node *)malloc(sizeof
  node));
  if(new = NULL){
        printf("memory alloc failed");
          exit (1);
  }
new -> data =val;
new -> left =NULL;
new -> right =NULL;
  return new;
}
struct node * insert right (struct node *root int val)
if( root = =NULL) {
printf("cant insert in NULL root");
  return null;
}
right -> right =create (val);
return root ->right;
}
struct node *insertleft (struct node *root, int val)
{
```

```
if (root == null) {
    printf("can't insert in مساحة");
    return null;
}
root -> left = create (val);
    return root -> left;
}
void deleteSubtree (struct node *node) {
    if (node == null) {
        return;
    }
    delete subtree (node -> left);
    delete subtree (node -> right);
    free (node);
}
void deleteleft (struct node *root) {
    if (root == null || root -> left == NULL) {
        printf("nothing to delete");
        return;
    }
    delete subtree (root -> left);
    root -> left = null;
    printf ("inleft subtree deleted");
}
void delete right (struct node *root) {
    if (root == null || root -> right == null) {
        printf("not to delete");
        return;
    delete tree (root -> right);
    root -> right = null;
}
```

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

O/P -

original tree : 1 2 4 5 3 6 >

left subtree deleted

tree after deletion : 1 3 6 >

2> pre, post, in order traversal

```
-> #include <stdio.h>
   #include <stdlib.h>
   struct node {
      int data;
      struct node * left;
      struct node *right;
   };
   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 \n");
      printf (" 2. post order \n");
      printf (" 3. in order \n");
      printf (" 4. Exit ");
      printf ("\n choose option: ");
      scanf ("%d", & choice);
      switch (choice) {
            case 1:
                preorder (root);
                break;

            case 2:
                postorder (root);
                break;
```

```
case 3:
        inorder (root);
        break;

case 4:
        exist (0);
        break;

default:
    print f ("Invalid input");
}
}
return 0;
}
void preorder (struct node *root){
    if (root == null){
        return;
    }
print f ("%d", root -> data);
preorder (root -> left);
preorder (root -> right);
}
void post (struct node *root){
    if (root == null){
        return;
    }
postorder (root -> left);
postorder (root -> right);
print f ("%d", root -> data);
}
```

```
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 option:

1 2 4 8 5 3 0)

Experiment - 12

1) Graph using adjency matrix

```
-> #include <stdio.h>
   #define us
   void init (int arr [u] [u]){
        int i, j;
   for (i=0; i<u; i++){
        for (j=0; j<u; j++){
                arr[i] [j] =0;
        }
   }
   }
   void insertedge (int arr [u] [u], int i, int j)
   {
        arr [i] [j] = 1;
        arr [j] [i] = 1;
   }
   void printadjmatrix (int arr [u] [u]){
        int i, j;
   printf ("\n adjacency matrix");
   for (i=0; i<v; i++){
        for (i=0; i<u; i++){
                printf ("%d", arr [i] [i]);
        }
        printf ("\n");
   }
}
```

```
int main ()
{
int adjmatrix [V] [V];
init (adjmatrix);
insertedge (adjmatrix , 0,1);
insertedge (adjmatrix , 0,2);
insertedge (adjmatrix , 2,0);
insertedge (adjmatrix , 2,3);
"          "          1,3);
"          "          4,3);
"          "          2,4);
"          "
print adjmatrix (adjmatrix);
return 0;
}
```

O/P -

Adjacency matrix -

0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

2) Adjacency list

```
#include <stdio.h>
#include <stdlib.h>
#define V5
struct node {
    int vertex;
    struct node * next;
};
struct node * adj list [V];
struct node * createnode (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++){
        adjlist [i] = null;

}

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

```
new = degleNode (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 : ", i);
    while (temp){
    printf("%d" -> , temp->vertex);
    temp = temp ->next;
  }
    printf ("null \n");
  }
}
int main ){
  init();
  insert edge (0,1);
              (0,2);
              (2,0)
              (2,3)
              (1,3)
              (4,3)
              (2,4)
  print adj list ();
    return 0;
}
```

qp -
0: 2 -> 2 -> 1 -> null
1: 4 -> 0 -> null
2: 4 -> 3 -> 0 -> 0 -> null
3: 4 -> 1 -> 2 -> null
4: 2 -> 3 -> null

Theory -
1) Write all basic terminologies of graph & explain them.

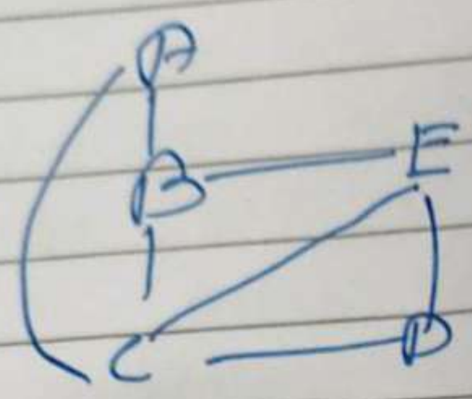
i) Vertex - it is generally related to as node, it is the fundamental unit of graph and is used to represent an entity in a graph.

ii) Edge - These are the lines or connection between vertex in a graph.

iii) Degree of vertex - it is the number of edge incident on that vertex.

iv) Path -> it is a sequence of vertices where each adjacent pair is connected by an edge.

2) Adjacency matrix and list representation



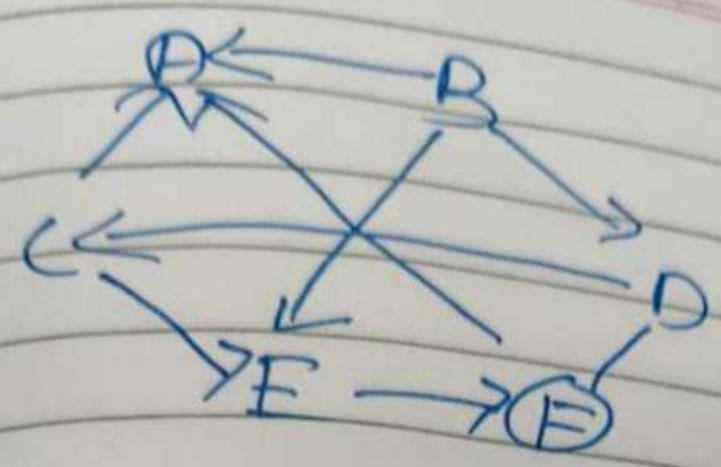
Matrix -

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

List

A [ ] -> [B |] -> [C |] -> NULL
B [ ] -> [A |] -> [C |] -> [E |] -> NULL
C [ ] -> [A |] -> [B |] -> [D |] -> [E |] -> NULL
D [ ] -> [C |] -> [E |] -> NULL
E [ ] -> [B |] -> [C |] -> [D |] -> NULL

1)



list -

A	→ B	→ C	→ F	→ NULL
B	→ A	→ D	→ E	→ NULL
C	→ A	→ D	→ E	→ NULL
D	→ B	→ C	→ F	→ NULL
E	→ B	→ C	→ F	→ NULL
F	→ A	→ D	→ E	→ NULL

matrix -

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