Exp. 1

Q.1.

```
# include <stdio.h>
int main(){
int n;
float average;
printf(" Enter the value
scanf (" ~~Enter~~ %d", &n);
int sum =(n*(n+1))/2;
average =(float) sum/n;
printf (" The average of the
    first %d natural number is:
    %.2f\n", n, average);
return 0;
}
```

Q2. Display the for loop patter

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

```
#include <iostdio.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");
}
 return 0;
}
```

Q.3 find first repetative element of array.

```
# include <stdio.h>
int main() {
int arr[100] n, i, j found=0;
~~float~~ printf("Enter the num
    of elements in array);
scanf("%d", &n);
printf("Enter %d elem :\n", n);
for(i = 0 ; i< n ; i++)
  {
   scanf("%d", & arr[i]);
  }
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 ;
}
```

```
if ( !found)
{
 printf("No repeating elem found \n);
}
 return 0;
}
```

" output

Enter number of elements in array : 6
Enter element :
53 45 23

Q. 4. find the greatest & smallest element in an array

```
# include <stdio.h>
int main()
{
 int arr[100], n, j,
 int max, min;
 printf(" Enter number of elem
   in array:");
 scanf("%d", &n);
 printf("Enter % element: \n",n
 for (i=0; i<n;i++)
  {
   scanf("%d", &arr[i]);
  }
  max = min = arr[0];
  For(i=1; i<n; i++)
  { if (arr[i] > max)
     {
      max = arr[i];
     }
  if (arr[i] < min)
     {
      min = arr[i]
     }
  }
  printf("Greatest element = %d \n", m
  printf("Smallest element = %d \n' min
```

```
return 0;
}
```

// output
Enter the number of element
in array: 5
Enter 5 element:
10 45 3 22 17
Greatest element = 45
Smallest element = 3

Q.5 C program to square the odd positioned elements.

```
# include <stdio.h>
int main()
{
 int arr[100], n, i;
 printf("Enter number of element
        in array:");
 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++)
 {
  if(i%2 = 0)
  {
   arr(i) = arr(i) arr(i);
  }
 }
 return 0;
}
```

Output

Enter the number of elements in the array: 6

Enter 6 elements:

1 2 3 4 5 6

Result

1 4 2 16 5 3 6

Q Patterns.

1.
```
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 ++)
{
printf("%d", i);
}
printf ("\n");
}
return 0;
}
```

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

```
#include <stdio.h>
int main ()
{
int i, j;
for (i = 1; i<=5 ; i++)
{
if (i % 2 == 0)
{
for (j = 1; j<= i; j++)
{
printf ("%d", j);
}
}
else{
for (j = 1; j<= i ; j++)
{
} printf ("%d", i);
}
printf ("\n");
}
return 0;
}
```

output

3 *

# #

$ $ $

9 9 9 9

```
# include <stdio.h>
int main()
{
 int i,
 for (i=0 , i<=4 ; i++)
  {
  switch (i)
  {
   case
    printf ("*\n");
    break;
   case 2
    printf ("# #\n");
    break;
   case 3;
    printf ("$ $ $\n");
    break;
   case 4:
    printf ("9 9 9 9\n");
    break;
   default;
    break;
  }
 }
 return 0;
}
```

4.

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

```
# include<stdio.h>
int main ()
{
 int i, j, Space;
 int rows = 4;
for (i = 1; i <= rows; i++)
 {
  for (Space = 1; Space <= rows - 1;
       Space ++)
  {
   printf ("  ");
  }
   printf ("* ");
 }
  printf ("\n");
 }
  return 0;
 }
```

5.

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

```
# include <stdio.h>
int main()
{
int i, j, space;
int rows = 4;
for (i=0; i< row; j++)
{
for (space = 0; space< i ; space++)
{
printf(" ");
}
for (j= 0; j< rows; j++)
{
printf ("* ");
}
printf ("\n")
}
return 0;
}
```

6. 1

1 2

1 2 3

1 2 3 4

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

7. *

* #

* # 9

* # 9 $

```
#include <stdio.h>
int main()
{
  int i, j, i
  char s[ ]= {'*','#','9','$'};
  for (i = 0; i<=i ; i++)
  {
    for (j = 0; j<=i ; j++)
    {
      printf ("%c",s[i]);
    }
    printf ("\n");
  }
  return 0;
}
```

Exp. 2

Q 1

```
#include <stdio.h>
int main(){
 int a[] = {56, 36, 89, 57, 01, 00,
            67, 599};
 int k, j, found = 0;
 printf("Enter key to be search:)
 scanf("%d", &k);
 for(i=0; i<8; i++)
 {
  if (a[i]==k)
  { printf("\nKEY FOUND AT INDEX:
            %d", i);
   found = 1;
   break;
  }
 }
 if (found == 0)
 printf("\n KEY NOT FOUND IN
          ARRAY");
 return 0;
}
```

i.

Enter key to be searched : 01
key found at index : 4

ii.

Enter key to be searched : 55
key not found in array.

Q.2

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

```
if(h > a[m])
{
    l = m+1;
}
if(h < a[m])
{
    h = m-1;
}
}
}
if(h < l)
printf("\n Element not found);
return 0;
}
```

// Output:

Enter number of array element : 6
Enter key to be search : 23
~~Enter key to be search~~ : 2
23
45
67
56
78
98

Q.3 Difference between linear search and binary search

linear Seach	Binary Seach
1. time complexity is low	1. time complexicity is O(log n)
2. works on both sorted and converted	2. only works onsorted data
3. It is less efficient, especially for large datarsete	3. It is more efficient especially for large data sets.
4 code is simpler	4. code is more complex.
5. It use sequenti searching approch	5. It uses divide and conquer approach.

Q.4. Srate limitation of linear search interms of line complexity

→ A linear search runs in an worst linear time and makes at most comparision where n is the length of the list. linear search is rarely practical because other search algorithm and schemes, such as the binary search algorithm and

Hash table allow significantly faster searching for all but short list.

Prac.

Q. Write program to copy elements of array and reverse it.

```
# include<stdio.h>
int main(){
int a[100],b[100], i,n,r;
printf("Enter the number of
        elements:");
scanf("%d", &a[n]);
printf("\n Enter array elem :\n");
for(i=0; i<n; i++){
 scanf("%d", &a[i]);
}
 r = n-1;
 for(i=0: i<n ;i++) {
  b[r] = a[i];
  --r;
 }
 printf("Reversed array: \n");
 for (i = 0; i < n ; i++) {
  printf("%d", b[i]);
 }
 printf ("\n");
 return 0;
}
```

// Output

Enter num of element : 3

Enter array elements :

1

2

3

~~Enter~~ reversed element :

3

2

1.

Q.2. Write c prog. to count total num of duplicate element in array.

```
# include <stdio.h>
int main(){
int arr[100], n, i, j;
int count = 0;
printf("Enter num of element:")
scanf("%d", &n);
printf("Enter element ");
~~scan~~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 ++;
    }
  }
}
printf("Total num of duplicate
      element")
```

// Output

Enter number of element : 4. 4

Enter array element
1 2 3 2

Number repeted
1.

3* Unique elements in array.

```
#include <stdio.h>
int main()
{
int a[100], i, j, n, check;
printf("Enter no. of array elem:")
scanf("%d", &n);
printf("\nEnter Array elem:\n")
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
printf("\nAll unique elements in
        array:\n");
for (i=0; i<n; i++)
{
if (a[i]==a[j] && i!=j)
{
check=1;
}
if(check==0)
{
printf("%d\n", a[i]);
}
}
return 0;
}
```

// Output.

Enter no. of array elements: 04

Enter array elements.
23
34
4
2

All unique elements in array:
23
4

* Odd / Even array

```
#include <stdio.h>
int main()
{
  int a [100], even [100], odd (100),
    i, j, n, c1, c2;
  printf( "Enter no. of array elem:"
  scanf("%d", &n);
  printf("\n Enter arr. element :\n");
  for(i=0: i<n; i++)
  {
    scanf("%d", &a[i]);
  }
    c1 = 0; c2 = 0;
    for( i= 0; i<n; i++)
  {
    if (a[i] % 2 == 0)
    {
      even[c1] = a[i];
      ++c1;
    }
    else
    {
      odd[c2] = a[i];
      ++c2;
    }
  }
}
```

```
printf("\nEven array:\n");
for (i=0; i<c1; i++)
{
  printf("%d\n", odd[i]);
}
return 0;
}
```

// output

Enter no. of array element : 5.

Enter array elements:
1
22
3
56
9

Even Array:
22
56

~~Eve~~ odd Array:
1
3
9

* Smallest element in array.

```
#include <stdio.h>
int main()
~~printf("E~~ {
int a [100], i, min, min2, n;
printf("Enter no. of array elem.");
scanf("%d", &n);
printf("\n Enter array elem: \n");
for (i=0; i<n; i++)
{
  scanf("%d", &a[i]);
}
main = a[0];
for (i=0; i<n; i++)
{
  if (a[i]< min)
  min = a[i];
}
for (i=0, i<n; i++)
{
  if(a[i]!= min)
  min 2 = a[i];
}
for (i=0;
{
  if (a[i]< min2 && a[i]!= min)
  min2 = a[i];
}
printf("\n Second smallest element.
in array : %d", min 2);
```

```
return 0;
}
```

// output.

Enter no. of array element : 4

Enter array elements.

1

2

4

5

Second smallest element in array: 2

* Count total no. of words in string.

```
# include <stdio.h>
int main()
{
int i=0; count=0;
char str[100], c;
printf("Enter a string:");
scanf("%d[^\n]%c", str, &c);
while (str[i]! = '\0')
{
if(i==0 || str[i]==' ')
{
++count;
}
++i;
printf("\n No. of words in
String = %d", count);
return 0;
}
```

// Output.

Enter a string : He is smart.

No. of words in string = 3

// Experiment · 3.

Q. Sort element.
Ascending order using bubble.

```
# include <stdio.h>
int main () {
int a [100] , n , i , j , r ;
printf (" Enter num. of elements
        in array :");
scanf ("%d", &n);
~~Enter array elements~~ : \n");
printf ("Enter array element : \n")
for (i=0 ; i<n ; i++) {
  scanf("%d", &a[i]);
}
for (i = 0; i < n - 1 ; i ++) ;
{
 for (j = 0 ; j < n - 1 - i ; j ++)
 {
   if ( a [j] > a[j +1]) {
     r = a [j] ;
     a[j] = a[j + 1] ;
     a [j+ 1] = r ;
   }
 }
}
printf("\n Sorted array in asendi
        elements : \n");
for (i = 0 ; i < n ; i ++) {
printf ("%d\n", a[i] ;
```

```
}
return 0;
}
```

// Output.

Enter numbers of elements
is array : 4
Enter array elements.
88
66
44
22

~~Enter~~ Sorted array in asending
22
44
66
88

Q. Sort element.
Descending order using selection

```
# include <stdio.h>
int main () {
int a[100], n, i, j, r;
printf("Enter no. of elements
        in array:");
scanf("%d", &n);
printf("Enter array elem: \n");
for (i=0; i<n; i++) {
scanf("%d", &a[i];
}
for (i=0; i<n; i++)
{
if(a[i] < a[j])
{
r=a[i];
a[i] = a[j];
a[j] = r;
}
}
}
printf("\n Sorted array element
    in desending order: \n");
for(i=0; i<n; i++) {
printf("%d \n", a[i]);
}
return 0;
```

// Output.

Enter number of elements in array : 4

Enter array elements:

22

44

66

88..

Sorted array elements in descen

88

66

44

22.

Q3. Find number of compar requir in bubble sort method & show ~~dramatic~~ the following list having 5 no : 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
×

" Algorithm.

1. Start.
2. compare each element and swap if the first element is greater than second element
3. After each full pass largest unsorted element moves to the end of array
5. Stop.

Q4. Sort the given array in assending order using selection sort methord & show diagramatic representati of every interation of for loop.
500, -20, 30, 14, 50.

Pass I)

500 -20 30 14 50
×
-20 500 30 14 50
×
-20 500 30 14 50

-20 500 30 14 50

Pass II)

-20 500 30 14 50
×
-20 30 500 14 50

-20 14 500 30 50

Pass III)

-20 14 500 30 50

-20 14 30 500 50

Pass IV)

-20 14 30 500 50

-20 14 30 50 500

// Algorithum :

1. Start
2. Search min element of array
3. Swap value of the selection of main.
4. In crement min to poin the next element location
5. Repeat till sorted
6. Stop.

29/7/25

Experiment 4

Q. Write a 'C' program to sort data from given array using.

i) insertion sort.

```
#include <stdio.h>
int main()
{
  int a [100], n, i, j, k, x, ii, n=1;
  printf("Enter no. of element:");
  scanf("%d", &n);
  printf("\n Enter elements of arr:"
  for(i=0; i<n; i++);
  scanf("%d", &a[i]);
  for(i=1; i<n; i++)
  {
    for (j=0; j<i; j++)
    {
      if(a[i]<a[j])
      {
        x=a[i]
        for(k=i-1; k>=j; k--)
        {
          a[k+1]=a[k];
        }
        a[j]=x;
      }
    }
```

```
printf("\n. Array after pass%d",
for(ii= 0; ii<n; ii++).
{
 printf("%d", a[ii]);
}
++p;
}
printf("\n\n Sorted arr in ased
          elending elements:");
for(i=0; i<n; i++)
{
 printf("\n%d", a[i]);
}
return 0;
}
```

// Output

```
Enter no of elem. in array:5
Enter elements.
9000
569
8589
6
3456
```

Array after pass 1: 569 9000 8589 6 3
Arr after pass 2: 569 85896 9000 6 3
Arr after pass 3: 6 569 8589 9000 34
Arr after pass 4: 6 569 3456 8589 90

Q.2. What is the output of insertion sort after the 2nd iteration given following. 7, 3, 5, 1, 9, 8, 4, 6

~~ii) Radix sort.~~

7 3 5 1 9 8 4 6.

1st interation.

7 3 1 9 4 8 6

3 7 1 9 4 8 6

2nd interation.

3 7 1 9 4 8 6

1 3 7 9 4 8 6

After 2nd interation the result.

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

Q.3. Sort the followin number.

1. 100 225 390 4130 956 99

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

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

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

	0	1	2	3	4	5	6	7	8	9	~~10~~
99	99										
~~5430~~ 4130	100	~~100~~									
					4130						
225	225										
390	390										
5431						5431					
956	956.										

→ 99, 100, 225, 390, 956, 4130, 5431.

2. 25, 6, 99, 145, 239, ~~239~~, 20, 18

	0	1	2	3	4	5	6	7	8	9
25						25				
6							6			
99										99
145						145				
239										239
20	20									
18									18	
20			20							
25			25							
145					145					
6	06									
18		18								
99										99
239										

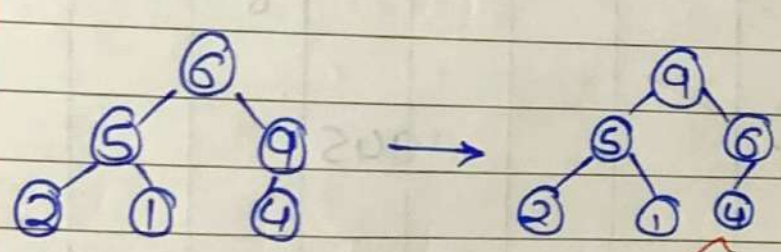
239

a	0	1	2	3	4	5	6	7	8	9
06	06									
18	18									
20	20									
25	25									
239			239							
145		145								
99	99									

→ Sorted arr. / 6, 18, 20, 25, 99, 145, 239

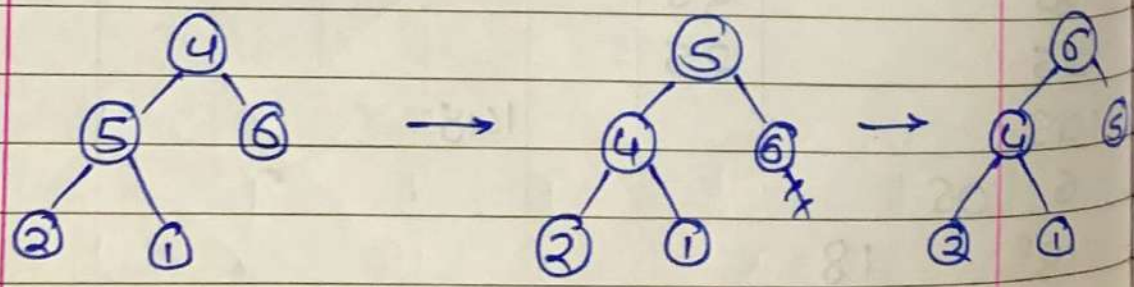
Q.4. Sort following elements using heap sort.

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



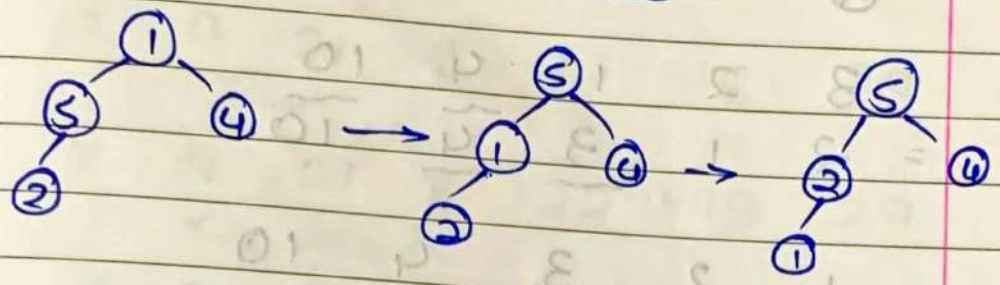
9, 5, 6, 2, 1, 4.

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



6, 5, 4, 2, 1, 9

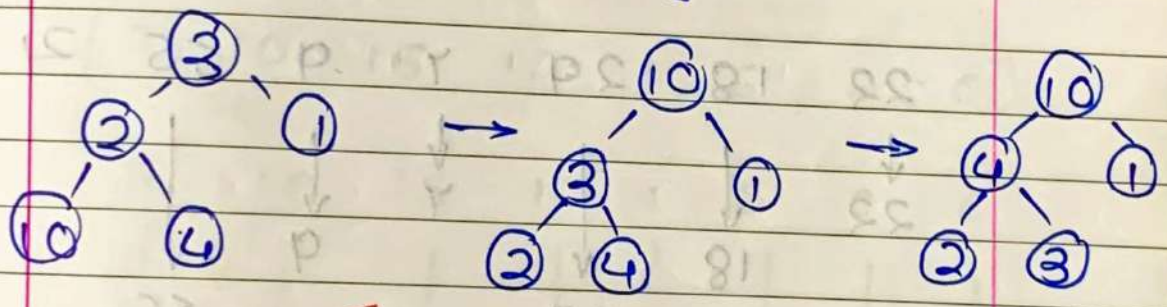
= 1, 5, 4, 2, 6, 9



5 2 4 1 6 9

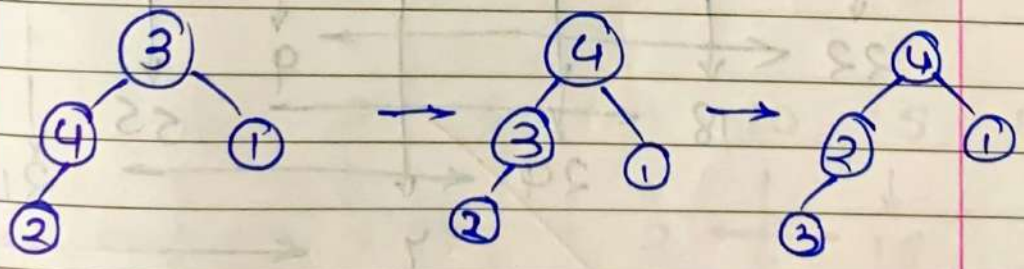
= 1 2 4 5 6 9

24 [3, 2, 1, 10, 4]



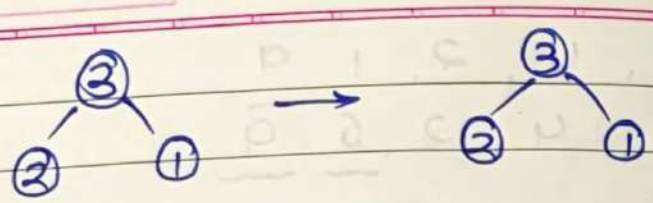
10 4 1 2 3

= 3 4 1 2 10



4 2 1 3 10

3 2 1 4 10



3 2 1 4 10

= 2 1 3 4 10

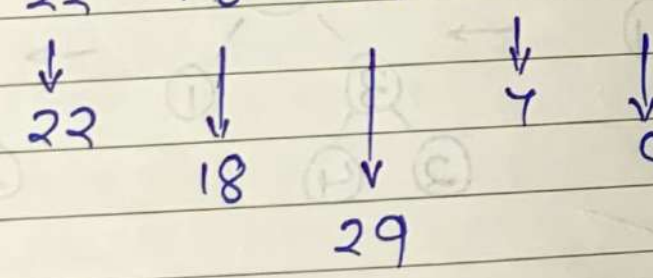
1 2 3 4 10

Q5. Sort following elements using Shalf Sort.

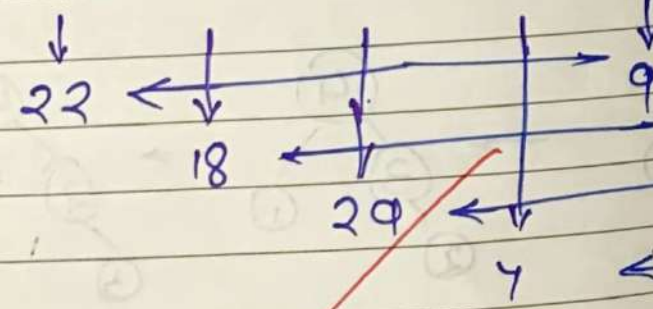
1. 22, 18, 29, 7, 9, 55, 21, 8

22 18 29 7 9 55 21 8

n/2 = 4

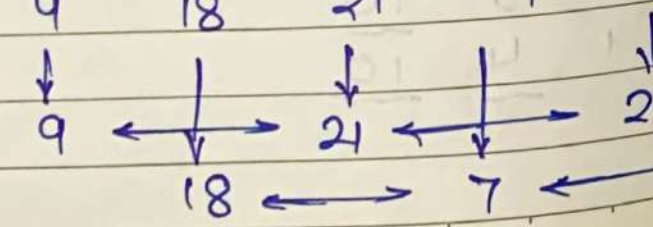


22 18 29 7 9 55 21 8



n/4 = 2

9 18 21 7 22 55 29 8



9 7 21 8 22 18 29 55

7 9 21 8 22 18 29 55

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

2. [3, 10, 15, 12, 1, 5, 2, 6]

3 10 15 12 1 5 2 6

3 10 15 12 1 5 2 6

1 5 2 6 3 10 15 12

1 2 3 15

5 6 10 12

// Exp. 5

Q. Array vs friendfun.

Feature.	Array	linklist.
• Memory allocation.	contagious, static	Non contagious dynamic.
• Size	fixed	flexible.
• Access	direct.	sequential
• Memory usage	extra pointer needed	extra memory for pointer.

Q. Terminologies for linklist.

1) Node.
Basic building block of linklist. it contain data & next. (address of next node).

2) Head.
Head is a reference to the following nod

3) Tail
Past node where Next = Null

4) Next
Each node has a next reference to following node

5) Null pointer.
Special value that indicates that ptr does not have object or memory

6) Empty linked list.
Singly linked list with no nodes in it

// Singly link list.

```
#include <stdio.h>
# include <std.h>
 void create();
 void display();
 void insert_begin();
 void insert_end();
 void delete_all();
 void delete_pos();
 int count_elems();
 void reverse_ll();
 void search ();
 struct node {
  int info;
  struct node * next;
 }
 struct node * start = NULL;
 int main()
 {
  int choice;
  while(1) {
  printf("\n 1.create \n");
  printf(" 2. display \n");
  printf(" 3. insert at begin \n");
  printf(" 4. insert at end \n");
  printf(" 5. insert at any poisition \n");
  printf(" 6. delete at given position \n");
  printf(" 7. delete all \n");
  printf(" 8. count element \n");
  printf(" 9. reverse link list \n");
  printf(" 10. search for element \n");
  printf(" 11. exit \n");
```

```
printf("Enter your choice:");
scanf("%d", & choice");
switch (choice)
{
    case 1: create () : break;
    case 2: display(): break;
    case 3: insert_begin () ;break;
    case4 : insert_end () ; break;
    case 5 : insert_pos () ; break;
    case 6 : delete_pos () ; break;
    case 7 : delete_all () ; break;
    case 8 : count_elem () ; break;
    case 9 : reverse_ll () ; break;
    case 10 : search () ; break;
    case 11 : exit (0) ; break;
      defaut : print (increase choice\n");
    }
  };
return 0;
}
void create()
{
struct node * temp * ptr;
temp = (struct node *) malloc (size of
printf("Enter data:"); (struct node));
scanf("%d" & temp → info);
temp next = Null;
if (start = = Null)
{
  start = temp;
}
else
{
```

```
ptr = Start;
while (ptr → next != Null)
{
 ptr = ptr → next;
}
}
 void display()
 {
 struct node * ptr.
  if (Start == Null)
  {
   Print ("\n Empty list \n");
   return;
  }
   ptr = Start;
   printf("\n list elements:");
   while (ptr != Null
  }
   print ("%d", ptr → info);
   ptr = ptr → next;
  }
  printf ("\n");
 }
  void insert_begin()
  {
  struct node * temp * ptr;
  temp = (struct node *) malloc(size of(struct
                                  node))
  printf("Enter data:");
  scanf("%d", temp → info);
   temp → next = Null;
   if (Start == Null)
   {
```

```
start = temp;
}
else
{ ptr = start;
  while (ptr → next != Null)
{
  ptr = ptr → next;
}
}
void insert_end ()
{
struct node * temp * ptr;
temp = (struct node*) malloc (size of (struct
printf ("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 insert_pos()
{
```

```
struct node * temp * ptr;
int i, pos;
temp = (struct node *) malloc(size of
printf("Enter data:");          (struct node));
scanf("%d", &temp -> info);
temp -> next = Null
printf("\n Enter position:");
scanf("%d", &pos);
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("Position not found \n");
 free (temp);
}
 else
{
 temp -> next = ptr -> next;
 ptr -> next = temp;
}
}
void delete all()
{
```

```
struct node * temp;
while (start != NULL)
{
 temp = start;
 start = start -> next
 free(temp);
 }
 printf("All values defered!\n");
}
 void delete pos()
{
 int i, pass;
 struct node * temp * ptr;
 if (start = NULL)
{
 print("Enter list:");
  return;
 }
 print("Enter position:");
 scanf("%d" & pos);
 if(pos == 1)
{
  temp = start;
  start = start -> next;
  free(temp);
   return;
 }
 ptr = start;
 for(i=1; i<pos-1 && ptr != NULL; i++)
{
  ptr = ptr -> next;
}
```

```
if (ptr == NULL || ptr -> == NULL)
{
printf("position not found \n");
}
else {
temp = ptr + next
ptr -> next = temp -> next;
free (temp);
printf ("Node detected \n");
}
}
int count-elems () {
int count = 0;
struct node * ptr = pstart;
while (ptr) NULL();
{
count ++;
ptr = ptr -> next
}
print("Number of elements: %d\n", count);
return 0;
}
void reverse-u () {
struset node * ptr;
int keypos = 1 = found = 0
printf("Enter element to search :"):
for (int i = len-1, i>=0, i--)
{
print f( ) %d", arr[i]);
}
```

```
printf("\n");
}
struct node *ptr;
int key, pos=1, found=0
printf("Enter element to search");
scanf("%d", &key);
ptr= start;
while (ptr!=NULL
{
 if (ptr→info==key)
{
 Printf(" element found at position %d\n"
 found=1;
 break;
}
 ptr=ptr→next;
 pos++;
}
 if (found){
 printf("Element not found \n");
}
}
```

Null

// Exp. 7

Q. Operations on stack.

```
#include <stdio.h>
#define MAX 100
  int stack [MAX]
  int top = -1;
  void push (int n);
  void pop ();
  void is Empty();
  void is full ();
  int main()
 {
  int value, choice;
  print ("Enter '1' to push : \n");
  Print ("Enter '2' to pop : \n");
  print ("Enter '3' to check if stack is em
 print ("Enter '4' to check if stack is full : \n)
 print ("Enter '5' to exit : \n");
 while(1)
{
 printf ("Enter your choice :");
 scanf ("%d", & choice);
 Switch (choice)
 {
  case 1 : print ("Enter value:");
           scanf ("%d" & value);
           push (value);
           break ;
 case 2: pop();
         break ;
 case 3; is Empty ()
         break;
```

```
case 4 : is full();
        break;
case 5 : printf("Exiting! ---\n");
        return 0;
    default : printf("Invalid input:\n")
} return 0;
}
void push (int n)
{
  if(top == MAX-1)
  {
    printf("Stack overflow!\n");
  }
  else
  {
    stack[++top] = n
    print("%d added to stack in\n" n);
  }
}
void pop()
{
  if (top==-1)
  {
    printf("Stack underflow\n");
  }
else
  {
    printf("~~Stack underflow~~ elements pop out.\n");
  }
}
void isEmpty()
{
  if(top ==-1)
```

```
{
 ifff printf("Stack is empty!\n")
 }
  else
 {
  printf("Stack is not empty!\n");
  }
}
 void is full()
 {
  if(top == Max-1)
  {
  printf("Stack is full!\n");
  }
  else
  {
  printf("Stack is not full!\n");
  }
 }
```

Q. ii) Stack size: 8 for push(10), push(20), pop. push(25), push(50), push(70), pop, pop pus(100), pop & draw final output.

	push(10)	push(20)	~~push(70)~~ pop	push(25)
7				
6				
5				
4				
3				~~50~~
2			~~25~~	~~25~~
1		20	~~20~~	~~20~~
0	10	10	10	10

	push(50)	push(70)	pop	pop	push(100)	pop.
7						
6						
5						
4						
3		70				
2	50	50	50		100	
1	25	25	25	25	25	25
0	10	10	10	10	10	10

final.

stack [S] → (size 8)
push (10) → stack [10]
push (20) → stack [10, 20]
pop → stack [10]
push (25) → stack [10, 25]
push (50) → stack [10, 25, 50]
push (75) → stack [10, 25, 50, 75]
pop → stack [10, 25, 50]
pop → stack [10, 25]
push (100) → stack [10, 25, 100]
pop → stack [10, 25]

Final output : stack [10, 25]

Experiment . 8

Q1 Convert Infix to Postfix or Prefix 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;
   if (x == '^')
     return 3;
   return -1;
```

```
}
int main() {
  char exp[100];
  char *e, x;
  printf("Enter infix expression:");

  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 (top != -1 && priority(stack[top])
        >= priority(*e))
      printf("%c", pop());
      push(*e);
    }
    e++;
    while (top != -1) {
      printf("%c", pop());
    }

    return 0;
}
```

// Output

Enter infix expression : A+(B*C-(B*C^F)
Postfix expression : ABC* DEF^/-(+

Q ii) Convert infix expression into prefix using sta

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

Input	Stack	Output.
H	*	H
*	*)	H
)	*)	H
G	*)	HG
*	*) *	HG
)	*)*)	HG
F	*) *)	HGF
^	*) *) ^	HGF
E	*) *) ^	HGFE
/	*) *) /	HGFE^
D	*) *) /	HGFE^D
(	*)*	HGFE^D/
-	*) -	HGFH^D/*
C	*) -	HGFH^D/*C
*	*) - *	HGFH^D/*C
B	*) - *	HGFH^D/*CB
(	*	HGFH^D/*CB*-
+	+	HGFH^D/*CB*-*
A	+	HGFH^D/*CB*-*A
		HGFH^D/*CB*-*A+

Q iii Write a C program to evaluate infix or Postfix Expression.

```
# include <stdio.h>
# include <ctype.h>
# include <stdlib.h>
~~# define S~~
 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("\n The postfix evaluation is: %d\n"
return 0;
}
void push (int n)
{
 if (top < size -1)
 {
   stack[++top]= n;
 }
 else
 {
   printf("Stack is full.\n");
```

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

// Output

Enter a postfix expression : 12+42/*1-42$+

The postfix evaluation is : 6

Q IV Evaluate prefix expression using stack.

+ - * + 1 2 / 4 2 1 $ 42

Input	Opr1	Sig	Opr2	Result	Output
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

// Algorithum

Q 1Y i) Scan the infix expression from left to right
ii) If the scanned character is an operand output
iii) Else,
a. If the precedence of the scanned operater is greater than the prerandence of the operator in the stack
b. else, pop all the operators from the stack which are greater or equal to in the the precedence than that of the scanned operator to the stack
iv) If the scanned character is an '(' push it to the stack.
x v) If the scanned character is an ')', ~~pop the stack~~ push it to the stack
vi) If the scanned character is an ')' pop the stack & output it until a '(' is encounred and discard both parenthess
vi) Repeat steps 2-6 until infix exp is scanned
vii) Print the output
viii) Pop & output from the stack until it is not empty.

Q2) i) Scan the expression from right to left or reverse the infix expression
ii) Scan the char one by one
iii) If the char is an opreand, copy it to prefix notation output
iv) If the char is a closing parenthesis, then push it to the stack
v) If the char is a opening parenthesis, pop the elements in the stack until we find the corresponding closing parenthis
vi) If char scanned is an operator.
a. if the operator has precedence greater than or equal to the top of the stack, push the operator to the stack.
b. If the operator has precedance lesser than the top of the stack, output it to the prefix notation output & then check the above condition again with the new top of the stack
vii) After all the char are scanned, reverse the prefix notation output

Q3) i) Initalize an empty stacks
ii) Scan postfix expression from left to ri
iii) If operand → push into S.
iv) If operator → pop two operands from S. apply operation, push result back to S
v) After Scaning → Top of the stack is result.

Q.4. i) Initialize an empty stack S
ii) Scan postfix expression from right to left
iii) If operand → push into S
iv) If operator → pop two operands from S apply operation, push result back to S
v) After scanning → Top of stack is result.

// Exp. 9

Queue:

```
# include <stdio.h>
# include <stdlib.h>
# define size 5
 int queue [SIZE];
 int front = -1, rear = -1;
 void insert () {
  int value;
  if (rear == SIZE -1) {
   printf("Queue is full overflow)\n");
  } else {
   printf("Enter value to insert : ");
   scanf (" %d", & value);
   if (front == -1) {
   front = 0
   rear ++ ;
   queue [rear] = value;
   printf("Inserted %d \n" ; value);
   }
  }
 void delete () {
  if (front == -1 || front > rear) {
   printf("Queue is empty (underflow)\n"
  } else {
   printf ("Deleted %d\n", queue[front]);
   front ++ ;
  }
 }
```

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

int main (){
 int choice;
  while (1){
  printf ("\n 1. insert \n 2. Delete \n3. Displ
        \n 4. Exit \n");
  printf (" Enter your choice :");
  Scanf ("%d", &choice);

  Switch (choice){
  case 1: insert ();
          break;
  case 2:
          delete ();
          break;
  case 3:
          display ();
          break;
  case 4:
          exit(0);
          default:
  printf ("Invalid choice \n");
  }
}return 0;
```

// Output

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

Enter your choice : 1
Enter value to insert : 10
inserted 10.

Mean...
Enter your choice : 1
Enter value to insert : 20
inserted 20

Mean...
Enter your choice : 1
Enter value to insert : 20
inserted 20

menu...
Enter your choice : 3
queue element : 10, 20

menu...
Enter your choice : 2
Deleted 10

Menu...
Enter your choice : 4.

Q.2. Perform following operation on linear queue in array size 10 with this operation insert(10), insert(50), delete, insert(100) insert(20), delete, insert(25), insert(200)

insert(10)

									10

Rear = 0 Front = 0

insert(50)

								50	10

Rear = 1 Front = 0

Delete

								50	~~10~~

Rear = 1 front = 0.

insert(100)

							100	50	~~10~~

insert(20)

						20	100	50	~~10~~

Rear = 2 Front = 0

Delete

						20	100	~~50~~	~~10~~

insert(25)

				25	20	100	~~50~~	~~10~~	

insert(200)

			200	25	20	100	~~50~~	~~10~~	

Q Explain the concept of priority queue.

→ A priority queue is a special type of queue where each element is assigned a priority. & elements are derved based on their priority, not just their arrange order

Q Write algorithm of insertion & deletion of linear queue

→ Insertion:

1. Check if the queue is in overflow condition
2. if no overflow condition check if front is equal to -1 & change it to 0.
3. Increase rear by 1, equate the rear to the value entered
4. Print the value inquered.

Nly

// Exp 10.

- Circular queue.

```
#include <stdio.h>
# include <stdlib.h>
# define Max 6
 int front = -1;
  int rear = -1;
  int queue [max];

 void enqueue (int val);
 void dequeue();
 void display ();
 int main(){
    int choice value;
    while (1) {
    printf("--- Menu ---");
    printf ("1. enqueue\n");
    printf ("2. Dequeue \n");
    printf ("3. Display\n");
    printf ("4. Exit\n");
    printf (" Choose option; ");
    scanf("%d", &choice);
    switch ("%d" &choice);
     case1 :
          printf(" Enter value to enqued");
          scanf ("%d", &value);
          enqueue(value);
        break;
    case 2:
          dequeu();
          break;
```

```
case 3:
   display();
   break;
case 4:
   exit(0);
   break;
 default:
   printf("invalid input!");
  }
}
return 0;
}
void enqueue (int val){
   if (front == 0 && rear == MAX-1){
      printf("Queue full;");
 } else if (front == rear+1){
      printf("queue full!");
 }
 else{
   if (front == -1 && rear == -1){
         front = rear = 0;
         queue [rear] = val;
   } else if (front != 0 && rear = Max-1){
         rear = 0;
         queue [rear] = val;
 else{
      rear++;
      queue[rear] = val;
   }
 printf("%d inserted successfully", val);
}}
```

```
void dequeue(){
  if(front == -1){
    printf("queue empty");
  }else
  { int iteam = queue[front];
    if(front==rear){
      front = rear = -1;
    } else
  {
    front = (front +1) % max;
  }
  printf("%d" dequeued ",item);
  }
}
void display(){
  if(front == -1){
    printf("queue empty");
  }
  else{
  int i =front; pf("queue elements:")
    while(1){
      printf("%d", queue[i]);
      if(i == rear){
        break.
      }
  i=(i+1) % max;
  }}
}
```

// output

--- Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Choose option: 1

Enter value to be enqueue: 12

12 enque sucessfully

--- Menu ---

choose option : 3

queue elements : 12 13

--- Menu ---

choose option: 2

12 dequeued succesfully

--- Menu ---

choose option : 3

queue elements : 13

--- Menu ---

choose option: 4.

Nill
4/11

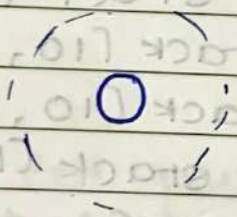
// Exp. 10 (theory)

Q1 Advantages of circular queue over normal queue

→ 1. Efficient memory use:
A circular queue reverse empty space. In a normal queue, once an el is removed, its spot remains empty, wasting memory.

2. Constant time complexity
Both insertion and deletion are performed in constant time, O(1) because pointer just move without need to shift elements

3. Simple points operators
logic to move front & rear pointer is straight forward.



Q2 Algorithms to insert in queue (enqueue)

Step 1 : Start

Step 2 : check for overflow (queue fal):
- if (front = = 0 & & rear = Max-size
- if (front = rear+1)
- the queue if full if either of the condition is true
- Display

Step 3 : Insert element at rear

- if front = rear = = -1 → set both to 0
- else → rear = (rear +1) % max-size
- insert at queue[rear]

step 4: exit

Algo to delete

Step 1. Start

Step 2. Check for underflow

if front = -1 → queue empty

Step 3: Retrive the element

data = queue[front]

step 4: increment front

- if front = rear → front = rear = -1
- else → front = (front +1) % max.

step 5: Return deleted element

Step 6: exit.

Q3 Applications of queue

1. CPU & task scheduling
2. Data buffering
3. Breath-first search (BFS)
4. Printer spooling

Applications of circular queue

1. Memory management.
2. Traffic light systems
3. Resource pooling

Q.4. Differ between stack & queue.

Stack	queue.
1. Last in first out (LIFO)	1. First in first out (FIFO)
2. Has only one active end pointer (TOP)	2. Has two pointer pointing to first & la.
3. Push → add elem Pop → Remove ele	3. Enqueue → add elem Dequeue → Remove.
4. only top is easily accessible	4. only front & rear are involved.

Nay
4/11

// Exp. 11

Q.1. Operations on Binary tree

```
# include <stdio.h>
# include <stdlib.h>
struct node {
    int data;
    struct node * left;
    struct node * right;
}
Struct node * create (int val)
struct node * insertleft (Struct node* roo
Struct node * insert right (struct node * root)
void delete subtree (struct node * root;
void delete left (struct node * root);
void display (struct node * root);

int main () {
    struct node * root = create (1);
    insert left (root, 2);
    insert right (root, 3);
    insert left (root → left, 4);
    insert right (root → left, 5);
    insert left (root → right, 6);
    insert right (root → right, 7)
    insert left (root → lift → left, 8);

    printf ("Original tree: ");
    display (root);
    print ("\n");
```

```
delete left (root)
printf (" Tree after deleting left subtre
        of root ;");
display(root);
return 0;
}
struct node create(int val){
    struct node new (struct node) malloc
        (size of( struct 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 (" can't insert in NULL root");
        return NULL;
    }
    root → right = create (val);
    return root → right;
    struct node insertLeft (struct node *
        root, int val)
    {
    if (root = = NULL) {
        printf (" Cant insert
```

```
}
root → left = create(val);
return root → left;
}
void delet subtree (node → left);
    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("\n left subtree deleted");
}
void delete right (struct node root){
    if (root == Null || root → left = Null){
    printf("Nothing to delete");
    return;
    }
void delete subtree (root → right);
    root → right = Null;
}
void display (struct node * root){
    if (root == Null){
        return;
```

```
}
printf("%d", root -> data);
  display (root -> left);
  display (root -> right);
}
```

// Output
original tree 1 2 4 8 5 3 6 7
left subtree deleted
Tree afte deletion: 1 3 6 7

Q.2 Pre Order, Post order, in order Transversal

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

  void peorder (struct node* root);
  void post orde (struct node * root);
  void inorder (struct node * root);

  int main (){
     int choice;
     while (1){
      printf("--- Menu ---\n");
      printf("1. Pre order trave\n");
      print f("2. post order traves\n");
```

```
printf("3. In-order traversal\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:
   exit(0);
   break;

  default.
   printf("\n valid input);
  }
 }
 return 0;
}
void preorder (struct node * root){
    if(root == Null){
      return;
    }
    printf("%d", root -> data);
    preorder( root -> left)
    preorder (root -> right);
  }
```

```
void postorder (struct node root){
    if( root = NULL){
        return;
    }
    postorder(root → left)
    postorder (root → right);
    printf ("%d", root → data);
}
void inorder (struct node root){
    if( root = = NULL){
        return;
    }
    inorder (root → left);
    printf("%d", root → data);
    inorder (root → right);
}
```

// Exp. 12 (Code).

```
# include <stdio.h>
# define V 5
void init (int arr[][V]){
    for(int i=0; i<V, i++){
        for (int j =0 ; j<V ; j++){
            arr[i][j]=0;
        }
    }
}
void insert Edge (int arr[][V], int i, int j
    arr[i][j] = 1;
    arr[j][i] = 1;
}
void print Adj Matrix (int arr[][V]){
    printf("Adjacency Matrix: \n");
    for (int i=0; i <V; i++){
        for (int j= 0; j<V; j++){
            printf(" %d ", arr[i][j];
        }
    }
}
int main (){
    int adj Matrix [V][V];
    init (adj Matrix);
    insert Edge (adj Matrix, 0, 1);
    insert Edge (adj Matrix, 0,4))
    insert Edge (adj Matrix, 1, 8);
    insert Edge (adj Matrix, 1, 2);
    insert Edge (adj Matrix. 1, 4)
    insert Edge (adj Matrix, 2, 3);
    print AdjMatrix (adj Matrix);
```

// Exp 12. (Theory)

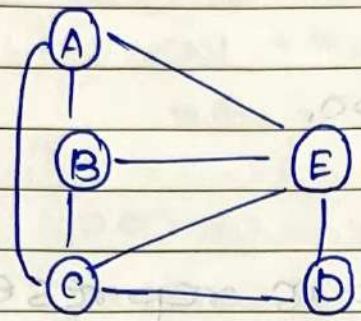
Explain Terminologies.

1. vertex
Each node at graph is represented as vertex. In the examples. labelled circle is known as vertex.

2. Edge
Edge represents a path between 2 vertices or line between two vertices

3. Adjacency
Two nodes or vertice are adjancent if they are connect to each other through edge

4. Path
Path represents a sequence of edge between two vertices



Matrix

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

list :

|A| → |B| → |C| → |E|

|B| → |C| → |A| → |E|

|C| → |B| → |A| → |D| → |E|

|D| → |E| → |C|

|E| → |A| → |B| → |C| → |D|