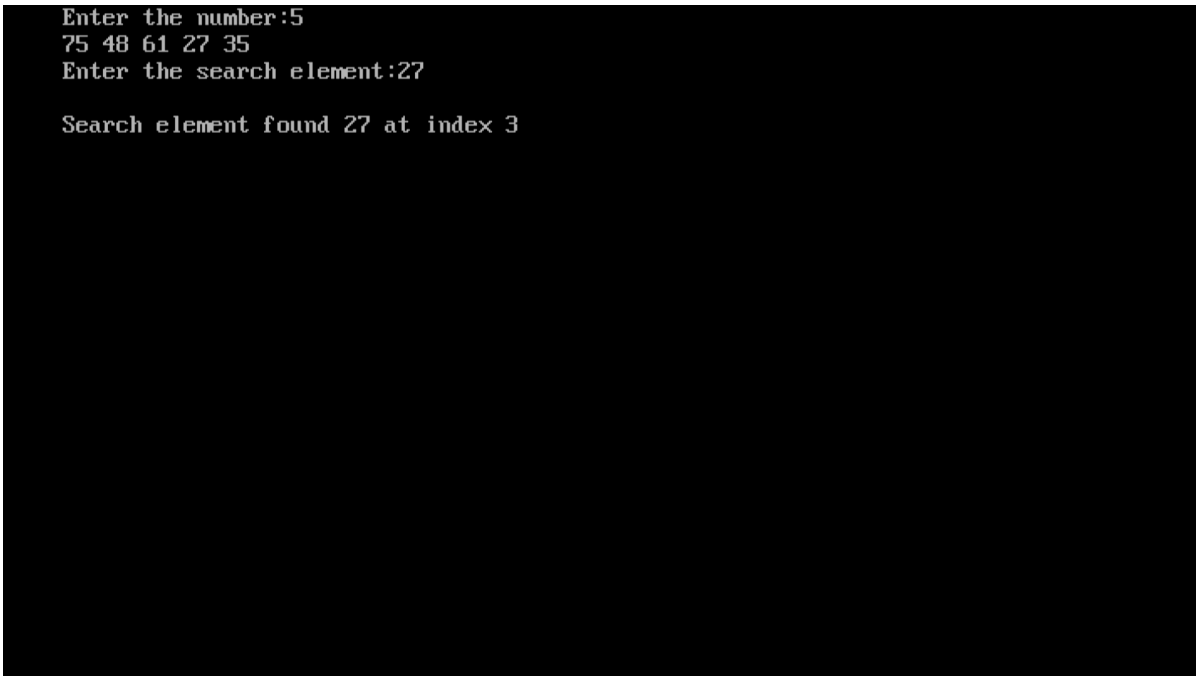
SEARCHING

1.a).Linear search

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int x[30],n,i,a;
	printf("Enter the number:");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
	scanff("%d",&x[i]);
	}
	printf("Enter the search element:");
	scanf("%d",&a);
	for(i=0;i<n;i++)
	{
	if(x[i]==a)
	{
	printf("\nSearch element found %d at index %d",a,i);
	break;
	}
	}
	if(i==n)
	{
	printf("Search element %d is not found",a);
	}
}
```

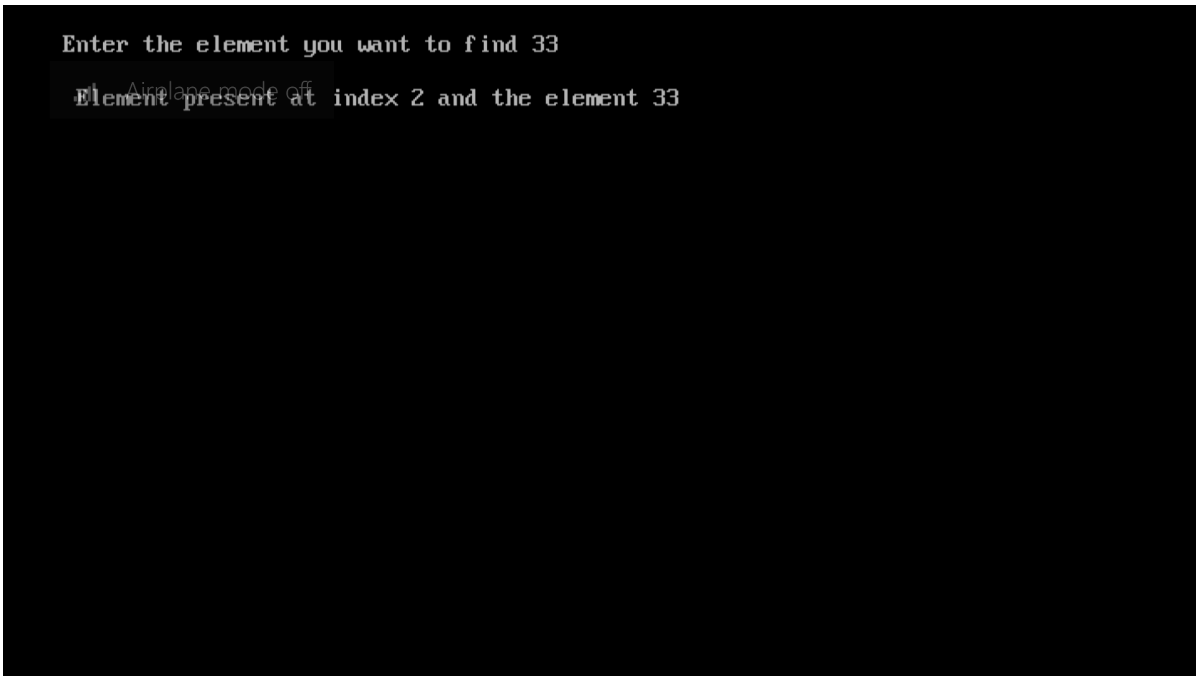
OUTPUT:


Enter the number:5
75 48 61 27 35
Enter the search element:27
Search element found 27 at index 3

1. b) Binary search

```
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x)
{
        if (r >= l) {
                int mid = l + (r - l) / 2;
                if (arr[mid] == x)
                        return mid;
                if (arr[mid] > x)
                        return binarySearch(arr, l, mid - 1, x);
                return binarySearch(arr, mid + 1, r, x);
        }
        return -1;
}
int main(void)
{
        int arr[] = { 11,22,33,44,55,66,77,88 };
        int n = sizeof(arr) / sizeof(arr[0]);
        int x ;
        printf("\nEnter the element you want to find");
    scanf("%d",&x);
        int result = binarySearch(arr, 0, n - 1, x);
        if(result==-1)
    {
       printf("\n Element not present in array");
    }
    else
    {
       printf("\n Element present at index %d and the element %d",result,x);
    }
}
```

OUTPUT:

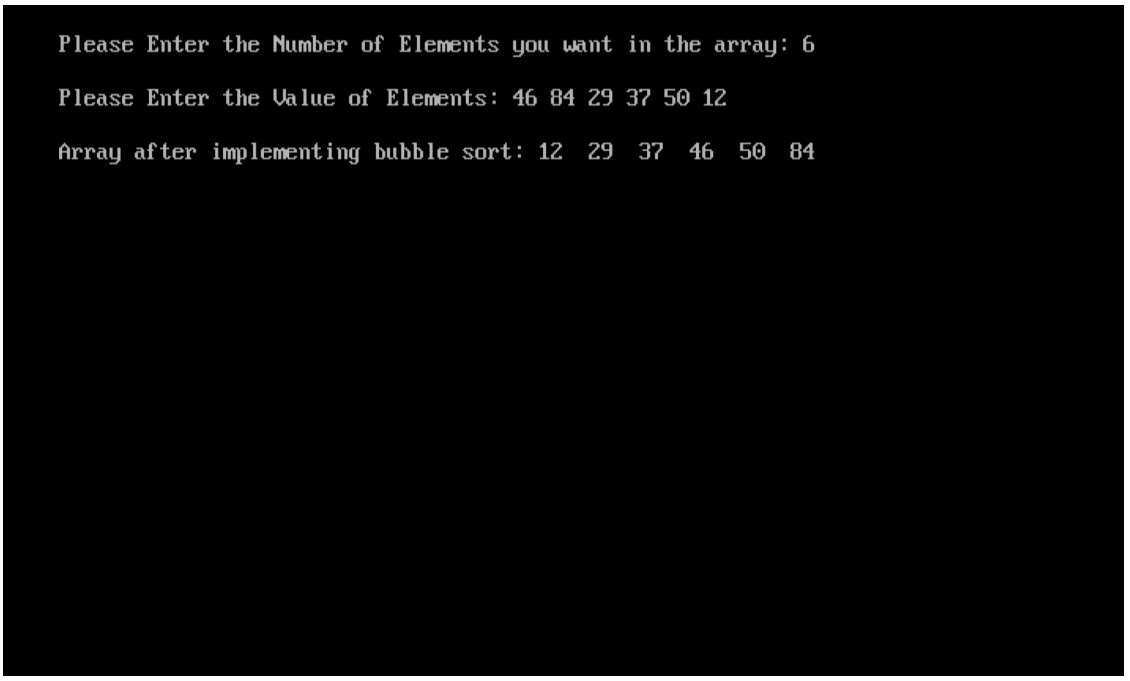

Enter the element you want to find 33
Element present at index 2 and the element 33

SORTING

2 a).Bubble sort

```
#include <stdio.h>
#include<conio.h>
void main()
{
   int arr[50], num, x, y, temp;
   clrscr();
   printf("\nPlease Enter the Number of Elements you want in the array: ");
   scanf("%d", &num);
   printf("\nPlease Enter the Value of Elements: ");
   for(x = 0; x < num; x++)
                scanf("%d", &arr[x]);
  for(x = 0; x < num - 1; x++)
  {
                for(y = 0; y < num - x - 1; y++)
                {
                 if(arr[y] > arr[y + 1])
                 {
                        temp = arr[y];
                        arr[y] = arr[y + 1];
                        arr[y + 1] = temp;
                 }
                }
  }
   printf("\nArray after implementing bubble sort: ");
   for(x = 0; x < num; x++)
        {
                printf("%d  ", arr[x]);
        }
 getch();
}
```

OUTPUT:


Please Enter the Number of Elements you want in the array: 6
Please Enter the Value of Elements: 46 84 29 37 50 12
Array after implementing bubble sort: 12 29 37 46 50 84

b).Insertion sort

```
#include <stdio.h>
#include<conio.h>
void main()
{
   int n, i, j, temp;
   int arr[50];
   clrscr();
   printf("\t\t INSERTION SORT");
   printf("\nEnter number of elements\n");
   scanf("%d", &n);
   printf("Enter %d integers\n", n);
   for (i = 0; i < n; i++)
        {
                scanf("%d", &arr[i]);
        }
   for (i = 1 ; i <= n-1 ; i++)
        {
                 j=i;
                while (j>0 && arr[j-1]>arr[j])
                        {
                        temp=arr[j];
                        arr[j]=arr[j-1];
                        arr[j-1]=temp;
                        j--;
                }

        }
        printf("Sorted list in ascending order:\n");
   for (i = 0; i <= n-1; i++)
        {
                printf("%d\n", arr[i]);
        }
   getch();
}
```

OUTPUT:


INSERTION SORT
Enter number of elements
6
Enter 6 integers
76 63 92 41 8 50
Sorted list in ascending order:
8
41
50
63
76
92

c).Merge sort

```
#include <stdio.h>
#include<conio.h>
int a[10],b[10];
void merging(int low, int mid, int high) {
   int l1, l2, i;
   for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
         if(a[l1] <= a[l2])
          b[i] = a[l1++];
         else
          b[i] = a[l2++];
   }
   while(l1 <= mid)
         b[i++] = a[l1++];
   while(l2 <= high)
         b[i++] = a[l2++];
   for(i = low; i <= high; i++)
         a[i] = b[i];
}
void sort(int low, int high) {
   int mid;
   if(low < high) {
         mid = (low + high) / 2;
         sort(low, mid);
         sort(mid+1, high);
         merging(low, mid, high);
   } else {
         return;
   }
}
void main()
 {
   int i,n,k;
   clrscr();
   printf("\n\t\t\t MERGE SORT");
   printf("\nEnter the no.of elements:");
   scanf( "%d",&n);
```

```
printf("\nEnter %d elements",n);
   for(k=0;k<n;k++)
   {
        scanf("%d",&a[k]);
   }
   printf("List before sorting\n");
   for(i = 0; i<n; i++)
        printf("%d ", a[i]);
   sort(0, n);
 printf("\nList after sorting\n");
  for(i = 1; i<= n; i++)
        printf("%d ", a[i]);
   getch();
}
```

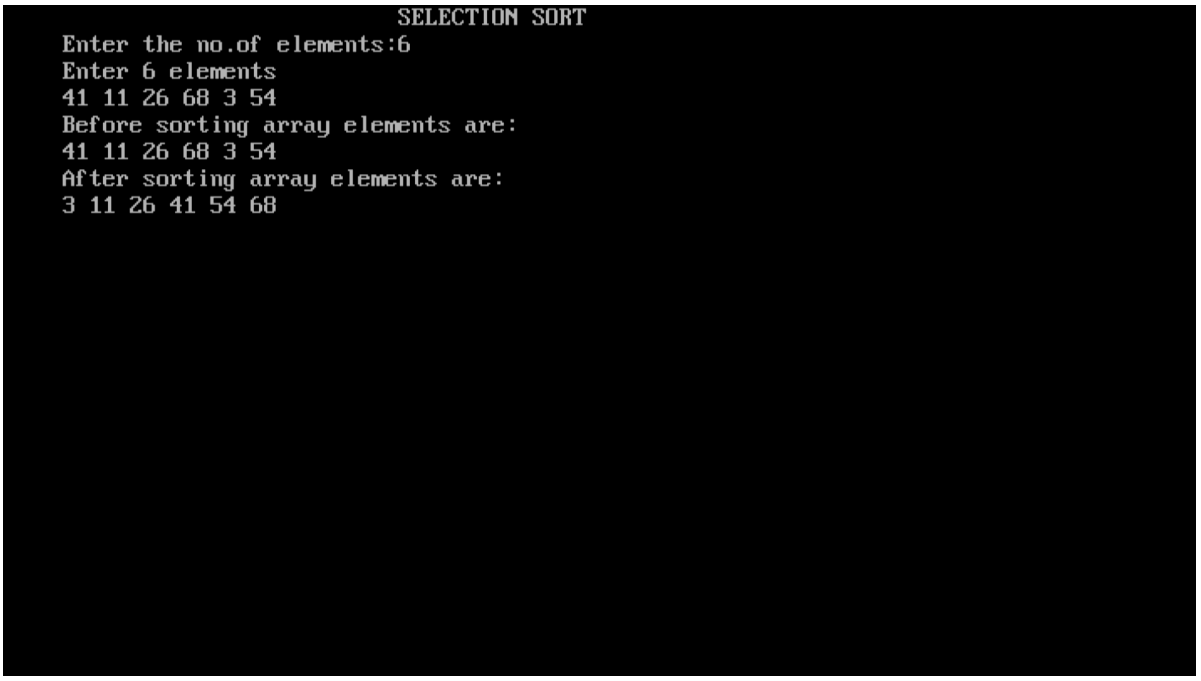
OUTPUT:


MERGE SORT
Enter the no.of elements: 6
Enter 6 elements 74 59 82 19 42 7
List before sorting
74 59 82 19 42 7
List after sorting
7 19 42 59 74 82

d).Selection sort

```
#include <stdio.h>
#include<conio.h>
void selection(int arr[], int n)
{  int i, j, min,temp;
   for (i = 0; i < n-1; i++)  {
                  min = i;
                  for (j = i+1; j < n; j++)
                  if (arr[j] < arr[min])
                  min = j;
   temp = arr[min];
   arr[min] = arr[i];
   arr[i] = temp;
   } }
void printArr(int a[], int n) {
   int i;
   for (i = 0; i < n; i++)
   printf("%d ", a[i]);
}
void main()
{
   int a[10],n,i;
   clrscr();
   printf("\t\t\t SELECTION SORT");
   printf("\nEnter the no.of elements:");
   scanf("%d",&n);
   printf("Enter %d elements",n);
   for(i=0;i<n;i++) {
     scanf("%d",&a[i]);
    }
   printf("Before sorting array elements are: \n");
   printArr(a, n);
   selection(a, n);
   printf("\nAfter sorting array elements are: \n");
   printArr(a, n);
        getch(); }
```

OUTPUT:


SELECTION SORT
Enter the no.of elements:6
Enter 6 elements
41 11 26 68 3 54
Before sorting array elements are:
41 11 26 68 3 54
After sorting array elements are:
3 11 26 41 54 68

LINKED LIST IMPLEMENTATION

3 a).Singly linked list

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void main ()
{
   int choice =0;
   clrscr();
   printf("\t\t\tSINGLY LINKED LIST\n");
   while(choice != 9)
   {
        printf("\nChoose one option ...");
printf("\n1.Insert in begining\n2.Insert at last
\n3.Insert at any intermediate location
\n4.Delete from Beginning\n5.Delete from last
\n6.Delete node after specified location\n7.Show
\n8.Exit\n");
        printf("\nEnter your choice: ");
        scanf("\n%d",&choice);
        switch(choice)
        {
           case 1:beginsert( );break;
```

```
            case 2: lastinsert();break;
            case 3:randominsert(); break;
            case 4:begin_delete();break;
            case 5:last_delete();break;
            case 6:random_delete();break;
            case 7:display(); break;
            case 8:exit(0);break;
            default:
            printf("Please enter valid choice..");
        } }
    getch();
}
void beginsert() {
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }
}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");}
```

```
  else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
           ptr -> next = NULL;
           head = ptr;
           printf("\nNode inserted");
        }
        else
        {
           temp = head;
           while (temp -> next != NULL)
           {
                 temp = temp -> next;
           }
           temp->next = ptr;
           ptr->next = NULL;
           printf("\nNode inserted");
}}}
void randominsert()
{
   int i,loc,item;
   struct node *ptr, *temp;
   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
   {
        printf("\nOVERFLOW");
   }

   else{
        printf("\nEnter element value");
        scanf("%d",&item);
        ptr->data = item;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d",&loc);
```

```
        temp=head;
        for(i=0;i<loc;i++)
        {
           temp = temp->next;
           if(temp == NULL)
           {
                printf("\ncan't insert\n");
                return;
           } }
        ptr ->next = temp ->next;
        temp ->next = ptr;
        printf("\nNode inserted");
   } }
void begin_delete()
{
   struct node *ptr;
   if(head == NULL)
   {
        printf("\nList is empty\n");
   }
   else
   {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
   } }
void last_delete()
{
   struct node *ptr,*ptr1;
   if(head == NULL)
   {
        printf("\nlist is empty");
   }

   else if(head -> next == NULL)
   {
        head = NULL;
```

```
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }
    else
    {
        ptr = head;
        while(ptr->next != NULL)  {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    } }
void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion
\n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;

        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        } }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("\nDeleted node %d ",loc+1);
}
```

```
void display()
{
   struct node *ptr;
   ptr = head;
   if(ptr == NULL)
   {
        printf("Nothing to print");
   }
   else    {
        printf("\nprinting values . . . . .\n");
        while (ptr!=NULL)
        {
           printf("\n%d",ptr->data);
           ptr = ptr -> next;
        }
}
}
```

OUTPUT:

```
                        SINGLY LINKED LIST

    Choose one option ...
    1.Insert in begining
    2.Insert at last
    3.Insert at any intermediate location
    4.Delete from Beginning
    5.Delete from last
    6.Delete node after specified location
    7.Show
    8.Exit

    Enter your choice: 1

    Enter value
    22_
```

```
    2.Insert at last
    3.Insert at any intermediate location
    4.Delete from Beginning
    5.Delete from last
    6.Delete node after specified location
    7.Show
    8.Exit

    Enter your choice: 2

    Enter value?
    33

    Node inserted
    Choose one option ...
    1.Insert in begining
    2.Insert at last
    3.Insert at any intermediate location
    4.Delete from Beginning
    5.Delete from last
    6.Delete node after specified location
    7.Show
    8.Exit
```

```
    Enter your choice: 3

    Enter element value11

    Enter the location after which you want to insert 1

    Node inserted
    Choose one option ...
    1.Insert in begining
    2.Insert at last
    3.Insert at any intermediate location
    4.Delete from Beginning
    5.Delete from last
    6.Delete node after specified location
    7.Show
    8.Exit
```

```
Enter your choice: 6

 Enter the location of the node after which you want to perform deletion
1

Deleted node 2
Choose one option ...
1.Insert in begining
2.Insert at last
3.Insert at any intermediate location
4.Delete from Beginning
5.Delete from last
6.Delete node after specified location
7.Show
8.Exit
```

```
Enter your choice: 7

printing values . . . . .

22
11
Choose one option ...
1.Insert in begining
2.Insert at last
3.Insert at any intermediate location
4.Delete from Beginning
5.Delete from last
6.Delete node after specified location
7.Show
8.Exit

Enter your choice:
```

b).Doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        struct node *prev;
        struct node *next;
        int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void main ()
{
int choice =0;
clrscr();
        while(choice != 9)
        {
        printf("\n*********Main Menu*********\n");
        printf("\n1.Insert in begining \n2.Insert at last
 \n3.Insert at any random location \n4.Delete from Beginning
\n5.Delete from last \n6.Delete the node after the given data
        \n7.Show \n8.Exit \n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice) {
           case 1: insertion_beginning();
                        break;
           case 2: insertion_last();
                        break;
           case 3: insertion_specified();
                       break;
```

```
            case 4: deletion_beginning( );
                        break;
           case 5: deletion_last( );
                        break;
            case 6: deletion_specified( );
                         break;
            case 7:display( );
                        break;
            case 8:exit(0);
                        break;
            default:printf("Please enter valid choice..");
 }}}
void insertion_beginning()
{
  struct node *ptr;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter Item value");
       scanf("%d",&item);
  if(head==NULL)  {
       ptr->next = NULL;
       ptr->prev=NULL;
       ptr->data=item;
       head=ptr;
  }
  else   {
        ptr->data=item;
       ptr->prev=NULL;
       ptr->next = head;
       head->prev=ptr;
       head=ptr;
  }
  printf("\nNode inserted\n");
}
void insertion_last()
{
  struct node *ptr,*temp;
  int item;
```

```
   ptr = (struct node *) malloc(sizeof(struct node));
   printf("\nEnter value");
   scanf("%d",&item);
   ptr->data=item;
  if(head == NULL)
   {
        ptr->next = NULL;
        ptr->prev = NULL;
        head = ptr;}
else {
          temp = head;
          while(temp->next!=NULL)
          {
             temp = temp->next;
          }
          temp->next = ptr;
          ptr ->prev=temp;
          ptr->next = NULL;
          }
        printf("\nnode inserted\n");
 }
void insertion_specified()
{
   struct node *ptr,*temp;
   int item,loc,i;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
        printf("\n OVERFLOW");
   }
   else
   {
        temp=head;
        printf("Enter the location");
        scanf("%d",&loc);
        for(i=0;i<loc;i++)
        {
          temp = temp->next;
```

```
            if(temp == NULL)
            {
               printf("\n There are less than %d elements", loc);
               return;
            } }
         printf("Enter value");
         scanf("%d",&item);
         ptr->data = item;
         ptr->next = temp->next;
         ptr -> prev = temp;
         temp->next = ptr;
         temp->next->prev=ptr;
         printf("\nnode inserted\n");
   } }
void deletion_beginning()
{
         struct node *ptr;
         if(head == NULL)
         {
         printf("\n UNDERFLOW");
         }
         else if(head->next == NULL)
         {
         head = NULL;
         free(head);
         printf("\nnode deleted\n");
         }
         else
         {
         ptr = head;
         head = head -> next;
         head -> prev = NULL;
         free(ptr);
         printf("\nnode deleted\n");
         }}
void deletion_last()
{
         struct node *ptr;
```

```
        if(head == NULL)
        {
        printf("\n UNDERFLOW");
        }
        else if(head->next == NULL) {
        head = NULL;
        free(head);

        printf("\nnode deleted\n");
        }
        else
        {
        ptr = head;
        if(ptr->next != NULL)
        {
           ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
        }}
void deletion_specified()
{
        struct node *ptr, *temp;
        int val;
        printf("\n Enter the data after which the node is to be deleted : ");
        scanf("%d", &val);
        ptr = head;
        while(ptr -> data != val)
        ptr = ptr -> next;
        if(ptr -> next == NULL)
        {
        printf("\nCan't delete\n");
        }
        else if(ptr -> next -> next == NULL)
        {
        ptr ->next = NULL;
        }
```

```
        else
        {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\nnode deleted\n");
        }}
void display()
{
        struct node *ptr;
    printf("\n printing values...\n");
        ptr = head;
        while(ptr != NULL)
        {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
        }
}
```

OUTPUT:

```
*********Main Menu*********

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Show
8.Exit

Enter your choice?
1

Enter Item value 50_
```

```
Node inserted

*********Main Menu*********

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Show
8.Exit

Enter your choice?
2

Enter value 90
```

```
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Show
8.Exit

Enter your choice?
4

node deleted

*********Main Menu*********

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Show
8.Exit
```

```
Enter your choice?
7

 printing values...
90

*********Main Menu*********

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Show
8.Exit

Enter your choice?
```

STACK

4 a). Stack using array

```
#include<stdio.h>
#include<conio.h>
#define N 5
int top=-1;
int stack[N];
void push()
{
 int x;
        printf("Enter data");
        scanf("%d",&x);
        if(top==N-1)
        {
        printf("Overflow");
        }
        else
        {
        top++;
        stack[top]=x;
        }
}
void pop()
{
        int item;
        if(top==-1)
        {
        printf("underflow");
        }
        else
        {
        item=stack[top];
        top--;
        printf("poped element : %d",item);
        }
}
```

```
 void peek()
{
        if(top==-1)
        {
        printf("stack is empty");
        }
        else
        {
        printf("Peek element : %d",stack[top]);
        }
}
void display()
{
        int i;
        for(i=top;i>=0;i--)
        {
        printf("\n%d",stack[i]);
        }
}
void main()
{
        int ch;
       printf("\n1.Push\t2.Pop\t3.peek\t4.display");
        do{
        printf("\nEnter your choice");
        scanf("%d",&ch);
        switch(ch)
        {
        case 0: exit(0);
        case 1: push(); break;
        case 2: pop(); break;
        case 3: peek(); break;
        case 4: display(); break;
        default : printf("invalid number");
        break;
        }while(ch!=0);
        getch();
}
```

OUTPUT:

PUT:

b). Stack using linked list

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node*next;
};
struct node *top=0;
void push()
{
        struct node *newnode;
        newnode=(struct node *)malloc(sizeof(struct node));
        printf("enter data:");
         scanf("%d",&newnode->data);
        newnode->next=top;
        top=newnode;
}
void display()
{
```

```
        struct node *temp;
        temp=top;
    if(top==0)
        {
        printf("Stack is empty");
        }
        else
        {
        while(temp!=0)
        {
        printf("\n %d",temp->data);
        temp=temp->next;
        }
        }
}

void peek(){
        if(top==0)
        {
        printf("Stack is empty"); }
        else  {
        printf("Top element is %d",top->data);
        } }
 void pop( )
 {
        struct node *temp;
        temp=top;
        if(top==0){
        printf("Stack is empty");}
        else{
        printf("Poped element : %d",top->data);
        top=top->next;
        free(temp);
        } }
 void main() {
        int ch;
       printf("\n1.Push\t2.Pop\t3.peek\t4.display\t5.Exit");
        do {
```

```
        printf("\nEnter your choice");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1: push( ); break;
        case 2: pop( ); break;
        case 3: peek( ); break;
        case 4: display( ); break;
        case 5:exit(0);
        default :printf("invalid number");
        break;
         } }
        while(ch!=0);
        getch( );
}
```

OUTPUT:

QUEUE

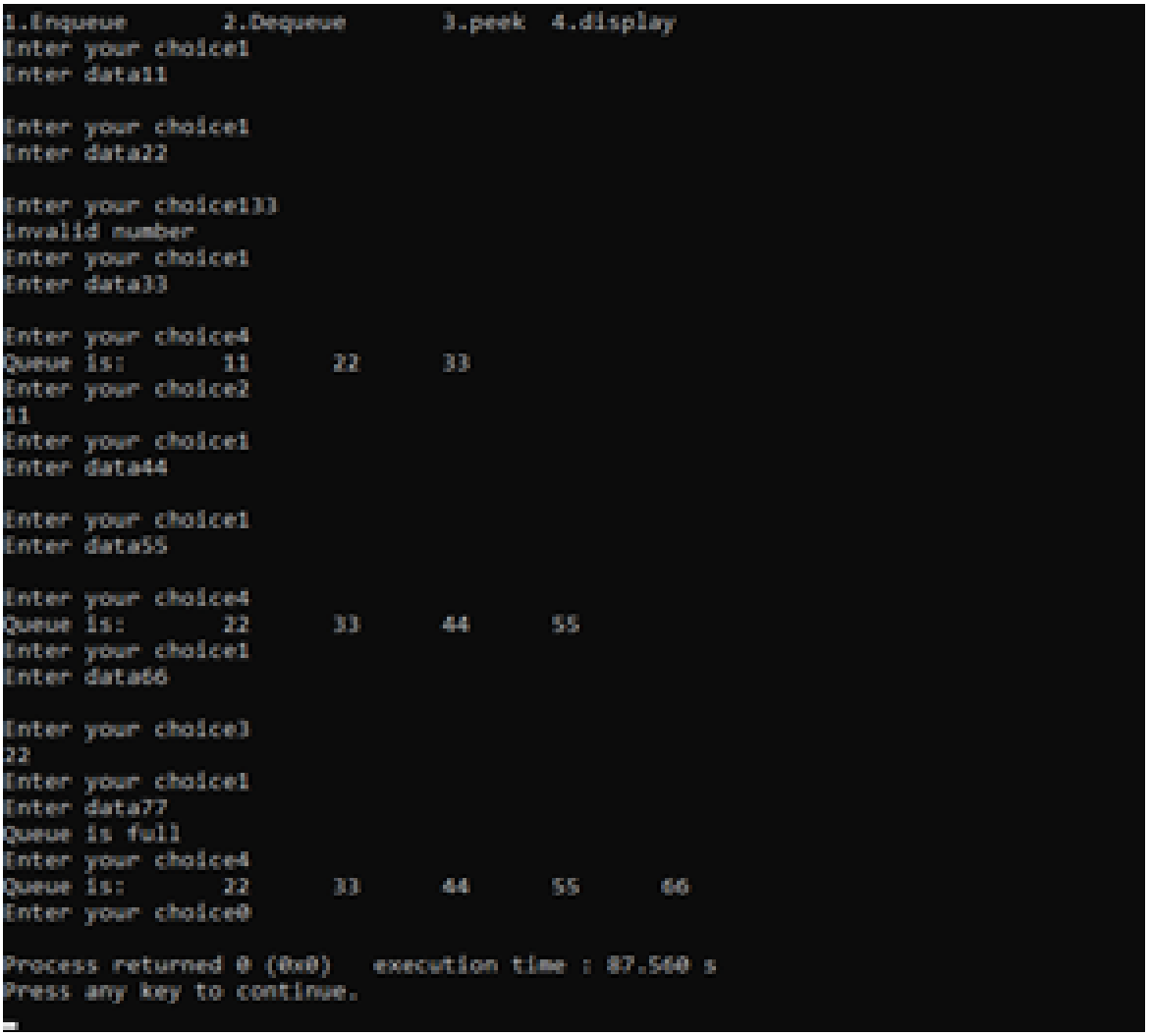
5 a).Circular queue

```
#include<stdio.h>
#include<conio.h>
#define N 5
int queue[N];
int front=-1;
int rear=-1;
void enqueue()
{
        int x;
        printf("Enter data");
        scanf("%d",&x);
        if(front==-1&&rear==-1){
        front=rear=0;
        queue[rear]=x;
        }
        else if(((rear+1)%N)==front){
        printf("Queue is full");
        }
        else{
        rear=(rear+1)%N;
        queue[rear]=x;
        } }
void dequeue() {
        if(front==-1 && rear==-1){
        printf("underflow");
        }
        else if(front==rear){
        front=rear=-1; }
        else {
        printf("%d",queue[front]);
```

```
        front=(front+1)%N;
        } }
void display( ) {
        int i=front;
        if(front==-1&& rear==-1){
        printf("Queue is empty");
        }
        else {
        printf("Queue is:");
        while(i!=rear) {
        printf("\t%d",queue[i]);
        i=(i+1)%N;
        }
        printf("\t%d",queue[rear]);
        } }
void peek(){
        if(front==-1 && rear==-1){
        printf("Queue is empty");
        }
        else{
        printf("%d",queue[front]);
        } }
void main() {
        int ch;
       printf("\n1.Enqueue\t2.Dequeue\t3.Peek\t4.Display\t5.Exit");
        do{
        printf("\nEnter your choice");
        scanf("%d",&ch);
        switch(ch){
        case 1: enqueue();
        break;
        case 2:dequeue();
        break;
        case 3:peek();
        break;
        case 4:display();
        break;
        default :printf("invalid number");
```

```
        } }
while(ch!=0);
        getch();
}
```

OUTPUT:



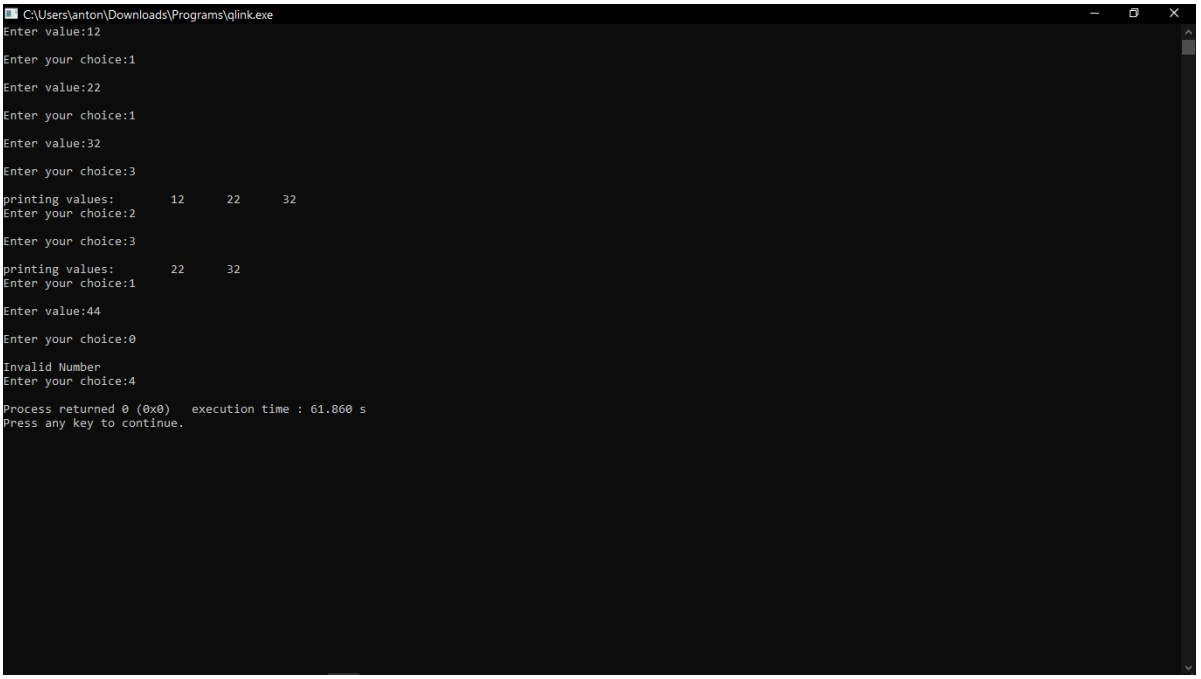
b).Queue using linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *front;
struct node *rear;
void insert( );
void delete( );
void display( );
void main ( )
{
   int choice;
      printf("\n1.insert an element\n2.Delete an element
\n3.Display the queue\n4.Exit\n");
      while(choice != 4) {
      printf("\nEnter your choice:");
      scanf("%d",& choice);
      switch(choice) {
         case 1: insert( ); break;
         case 2: delete( ); break;
         case 3: display( ); break;
         case 4: exit(0); break;
         default:
         printf("\nInvalid Number");
      } } }
void insert() {
   struct node *ptr;
   int item;
```

```
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL) {
       printf("\nOVERFLOW");
       return; }
    else {
       printf("\nEnter value:");
       scanf("%d",&item);
       ptr -> data = item;
       if(front == NULL) {
          front = ptr;
          rear = ptr;
          front -> next = NULL;
          rear -> next = NULL;
       }
       else {
          rear -> next = ptr;
          rear = ptr;
          rear->next = NULL;
       } } }
 void delete ( ) {
    struct node *ptr;
    if(front == NULL){
       printf("\nUNDERFLOW");
       return;
    }
    else {
       ptr = front;
       front = front -> next;
       free(ptr); } }
 void display( ) {
    struct node *ptr;
    ptr = front;
    if(front == NULL){
       printf("\nEmpty queue");
    }
    else
    {   printf("\nprinting values:");
       while(ptr != NULL)
```

```
{
   printf("\t%d",ptr -> data);
   ptr = ptr -> next;
} } }
```

OUTPUT:



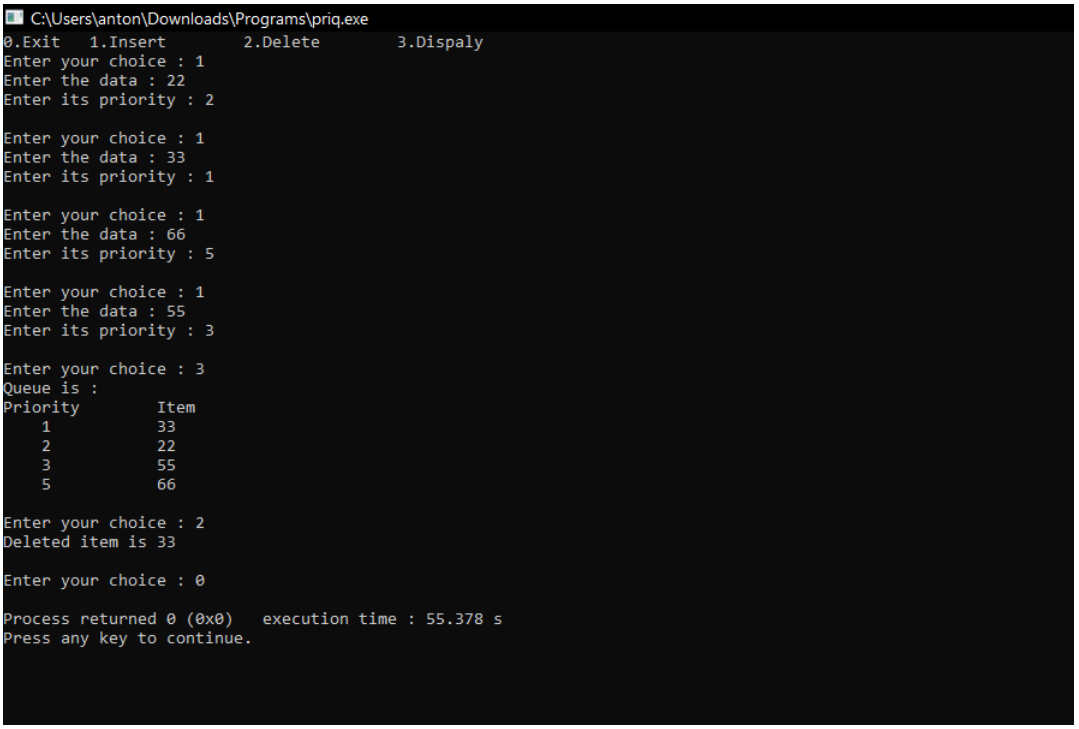
c).Priority queue

```
# include<stdio.h>
# include<malloc.h>
typedef struct node
        {
        int priority;
        int info;
        struct node *link;
}NODE;
NODE *front = NULL;
void insert()
{
        int data, priority;
        NODE *temp,*q;
        printf("Enter the data : ");
        scanf("%d",&data);
        printf("Enter its priority : ");
        scanf("%d",&priority);
        temp = (NODE *)malloc(sizeof(NODE));
        temp->info = data;
        temp->priority = priority;
        if( front == NULL || priority < front->priority ){
        temp->link = front;
        front = temp;
        }
        else {
        q = front;
        while( q->link != NULL && q->link->priority <= priority )
        q=q->link;
        temp->link = q->link;
        q->link = temp;
        } }
        void del( ) {
```

```
        NODE *temp;
        if(front == NULL)
        printf("Queue Underflow\n");
        else {
        temp = front;
        printf("Deleted item is %d\n", temp->info);
        front = front->link;
        free(temp);
        } }
void display( )
{
        NODE *ptr;
        ptr = front;
        if(front == NULL)
        printf("Queue is empty\n");
        else {
        printf("Queue is :\n");
        printf("Priority        Item\n");
        while(ptr != NULL)
        {
        printf("%5d       %5d\n",ptr->priority,ptr->info);
        ptr = ptr->link;
        } } }
int main()
{
        int choice;
        printf("0.Exit\t 1.Insert\t 2.Delete\t 3.Dispaly");
        while(1) {
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch(choice){
        case 0: exit(0);
        case 1: insert();
        break;
        case 2: del( );
        break;
        case 3: display();
        break;
```

```
        default : printf("Invalid number\n");
        }}
        return 0;
}
```

OUTPUT:



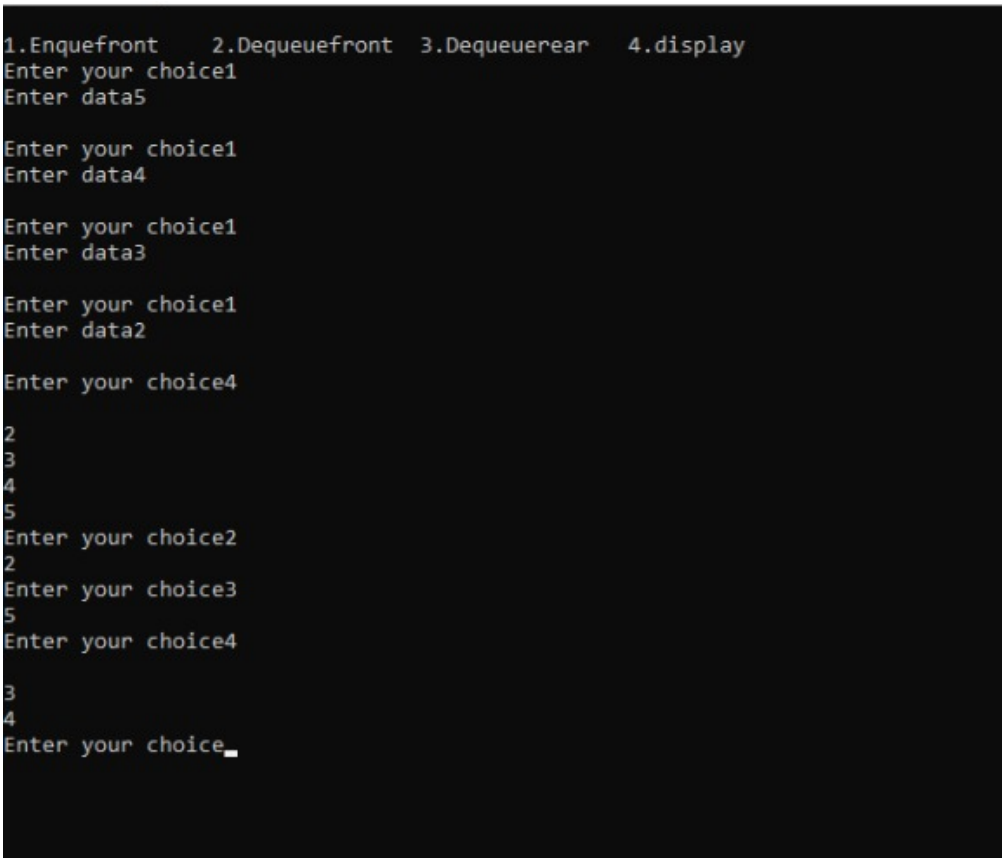
d). Input restricted deque

```
#include<stdio.h>
#include<conio.h>
#define N 5
int deque[N];
int f=-1,r=-1;
void enqueuefront()
{
   int x;
   printf("Enter data");
   scanf("%d",&x);
   if((f==0 && r==N-1) || (f==r+1))
   {
      printf("Queue in full");
   }
   else if(f==-1 && r==-1)
   {
      f=r=0;
      deque[f]=x;
   }
   else if(f==0)
   {
      f=N-1;
      deque[f]=x;
   }
   else
   {
      f--;
      deque[f]=x;
   }
}
void display() {
   int i=f;
```

```
    while(i!=r)
    {
        printf("\n%d",deque[i]);
        i=(i+1)%N;
    }
    printf("\n%d",deque[r]);
}
void dequefront()
{
    if(f==-1&&r==-1)
    {
        printf("Empty");
    }
    else if(f==r)
    {
        f=r=-1;
    }
    else if(f==N-1)
    {
        printf("%d",deque[f]);
        f=0;
    }
    else
    {
        printf("%d",deque[f]);
        f++;
    } }
void dequeurear()
{
    if(f==-1&&r==-1)
    {
        printf("Empty");
    }
    else if(f==r)
    {
        f=r=-1;
    }
    else if(r==0) {
```

```
        printf("%d",deque[r]);
        r=N-1;
    }
    else {
        printf("%d",deque[r]);
        r--;
    }
}
void main()
{
    int ch;
        printf("\n1.Enquefront \t2.Dequeuefront \t3.Dequeuerear \t4.display");
        do
        {
        printf("\nEnter your choice");
        scanf("%d",&ch);
        switch(ch)
        {
        case 0:exit(0);
         case 1: enqueuefront();
         break;
        case 2:dequefront();
         break;
        case 3: dequeurear();
         break;
        case 4:  display();
         break;
        default :  printf("invalid number");
          break;
        } }
while(ch!=0);
        getch();
}
```

OUTPUT:


1.Enquefront 2.Dequeuefront 3.Dequeuerear 4.display
Enter your choice1
Enter data5
Enter your choice1
Enter data4
Enter your choice1
Enter data3
Enter your choice1
Enter data2
Enter your choice4
2
3
4
5
Enter your choice2
2
Enter your choice3
5
Enter your choice4
3
4
Enter your choice

TREE TRAVERSAL

6 a).Traversal- Inorder, Preorder, Postorder

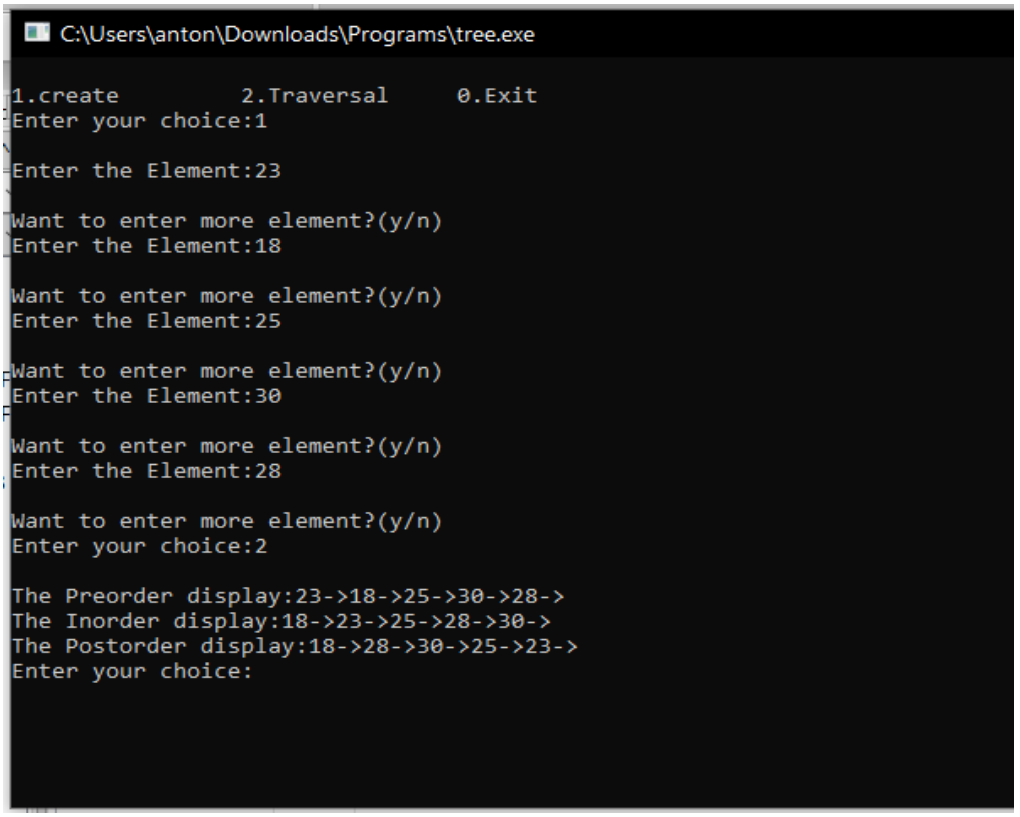
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
typedef struct node
{
        int data;
        struct node *lchild,*rchild;
}node;
node *getnode()
{
        node *temp;
        temp=(node *)malloc(sizeof(node));
        temp->lchild=NULL;
        temp->rchild=NULL;
        return temp;
}
void insert(node *root,node *newnode)
{
        if(newnode->data<root->data)
        {
        if(root->lchild== NULL)
        root->lchild=newnode;
        else
        insert(root->lchild,newnode);
        }
        if(newnode->data>root->data)
        {
        if(root->rchild==NULL)
        root->rchild=newnode;
        else
```

```
        insert(root->rchild,newnode);
        }
}

void preorder(node *temp)
{
        if(temp==NULL) return;
        inorder(temp->lchild);
           printf("%d->",temp->data);
        inorder(temp->rchild);
}
void inorder(node *temp)
{
        if(temp==NULL) return;
         printf("%d->",temp->data);
        preorder(temp->lchild);
        preorder(temp->rchild);
}
void postorder(node *temp)
{
        if(temp==NULL) return;
        postorder(temp->lchild);
        postorder(temp->rchild);
         printf("%d->",temp->data);
}
void main()
{
        int choice;
        char ans='N';
        int key;
        node *newnode,*root,*temp,*parent;
        node *getnode();
        root=NULL;
        printf("\n1.create \t 2.Traversal \t 0.Exit");
        while(1)
        {
        printf("\nEnter your choice:");
        scanf("%d",&choice);
```

```
switch(choice)
        {
        case 1:
        do{
        newnode=getnode();
        printf("\nEnter the Element:");
        scanf("%d",&newnode->data);
         if(root==NULL)
           root=newnode;
        else
        insert(root,newnode);
        printf("\nWant to enter more element?(y/n)");
        ans=getch();
        }while(ans=='y');
        break;
        case 2:
        if(root==NULL)
                {
                printf("Tree is not created");
                }
        else{
                printf("\nThe Preorder display:");
                inorder(root);
                printf("\nThe Inorder display:");
                preorder(root);
                printf("\nThe Postorder display:");
                postorder(root);
                }break;
        case 0:
        exit(0);
        }
}
}
```

OUTPUT:


C:\Users\anton\Downloads\Programs\tree.exe
1.create 2.Traversal 0.Exit
Enter your choice:1
Enter the Element:23
Want to enter more element?(y/n)
Enter the Element:18
Want to enter more element?(y/n)
Enter the Element:25
Want to enter more element?(y/n)
Enter the Element:30
Want to enter more element?(y/n)
Enter the Element:28
Want to enter more element?(y/n)
Enter your choice:2
The Preorder display:23->18->25->30->28->
The Inorder display:18->23->25->28->30->
The Postorder display:18->28->30->25->23->
Enter your choice:

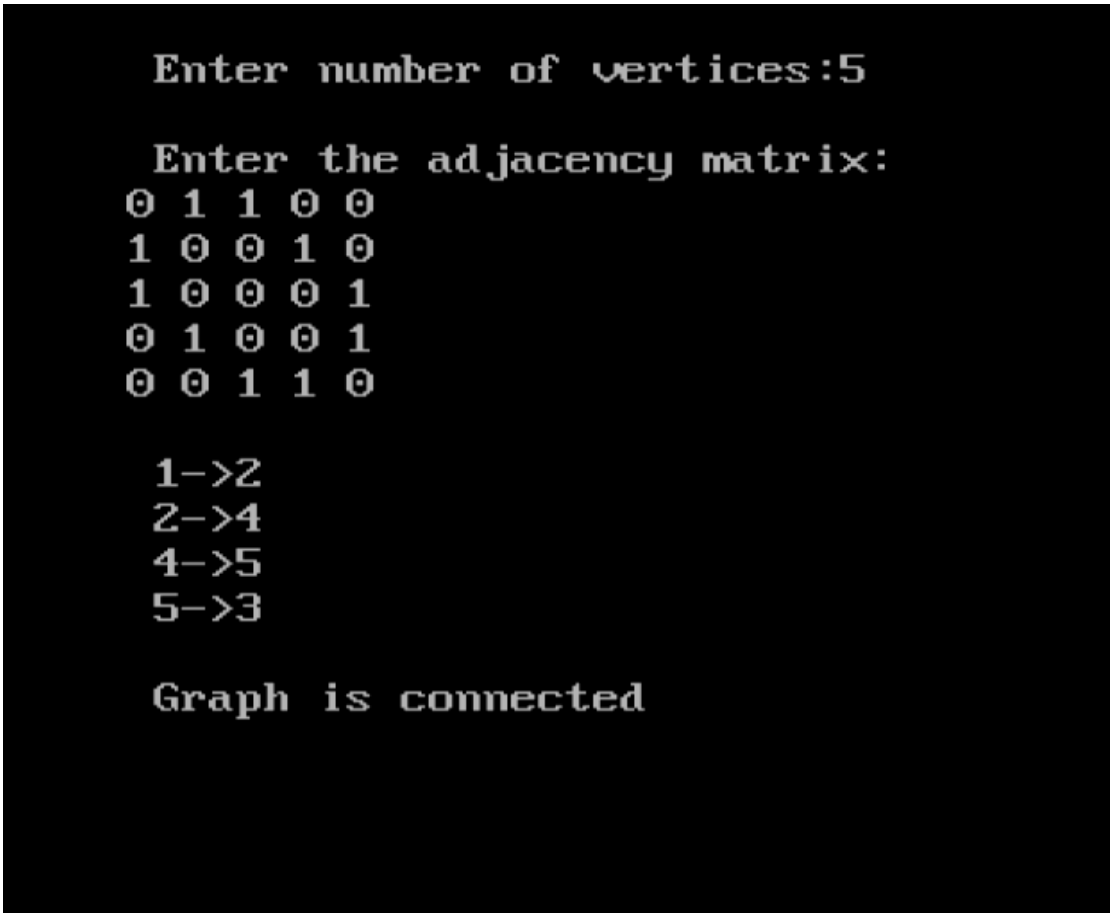
GRAPH

7 a).Graph Traversal DFS

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v){
 int i;
 reach[v]=1;
 for (i=1;i<=n;i++)
  if(a[v][i] && !reach[i]){
   printf("\n %d->%d",v,i);
   dfs(i);
  }}
void main(){
  int i,j,count=0;
  printf("\n Enter number of vertices:");
  scanf("%d",&n);
  for (i=1;i<=n;i++){
    reach[i]=0;
    for (j=1;j<=n;j++)
      a[i][j]=0;}
   printf("\n Enter the adjacency matrix: \n");
   for(i=1;i<=n;i++){
    for(j=1;j<=n;j++){
       scanf("%d",&a[i][j]);
    }}
   dfs(1);
   printf("\n");
   for (i=1;i<=n;i++){
    if(reach[i])
       count++;
    }
    if(count==n)
       printf("\n Graph is connected");
```

```
    else
       printf("\n Graph is not connected");
getch();
}
```

OUTPUT:

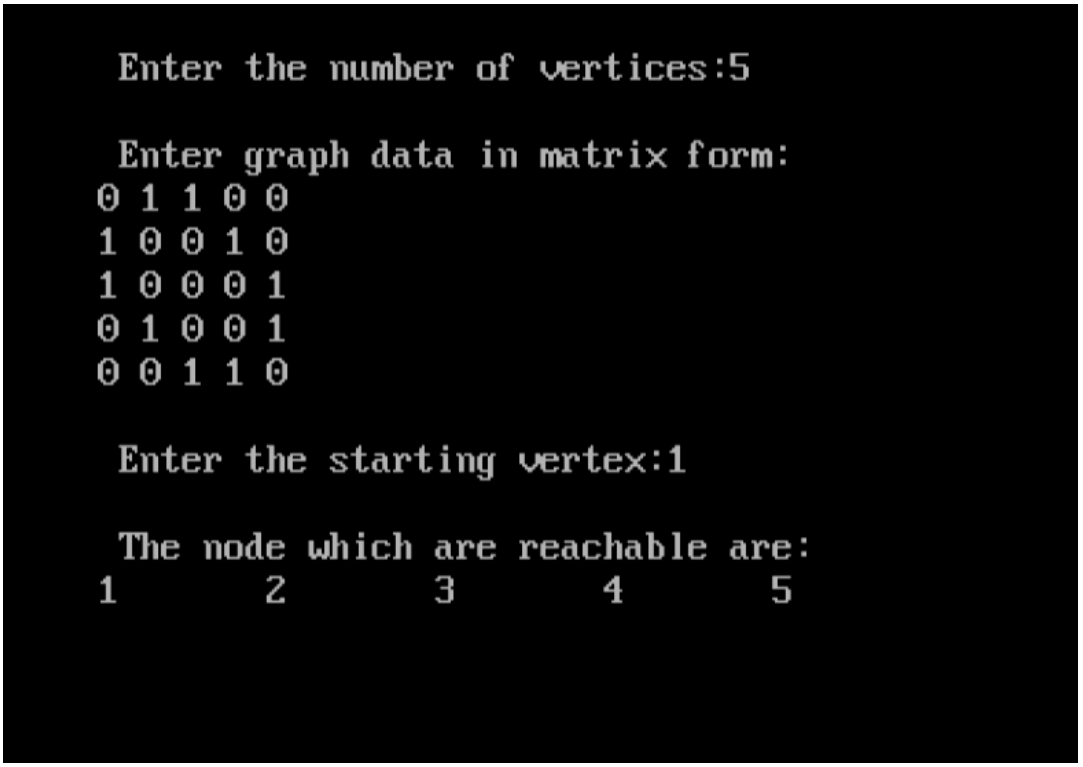


b).Graph Traversal BFS

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v) {
        for (i=1;i<=n;i++)
          if(a[v][i] && !visited[i])
           q[++r]=i;
        if(f<=r) {
                visited[q[f]]=1;
                bfs(q[f++]);
        }
}
void main() {
        int v;
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++) {
                q[i]=0;
                visited[i]=0;
        }
        printf("\n Enter graph data in matrix form:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
           scanf("%d",&a[i][j]);
        printf("\n Enter the starting vertex:");
        scanf("%d",&v);
        bfs(v);
        printf("\n The node which are reachable are:\n");
        for (i=1;i<=n;i++)
          if(visited[i])
           printf("%d\t",i); else
           printf("\n Bfs is not possible");
```

```
    getch();
}
```

OUTPUT:



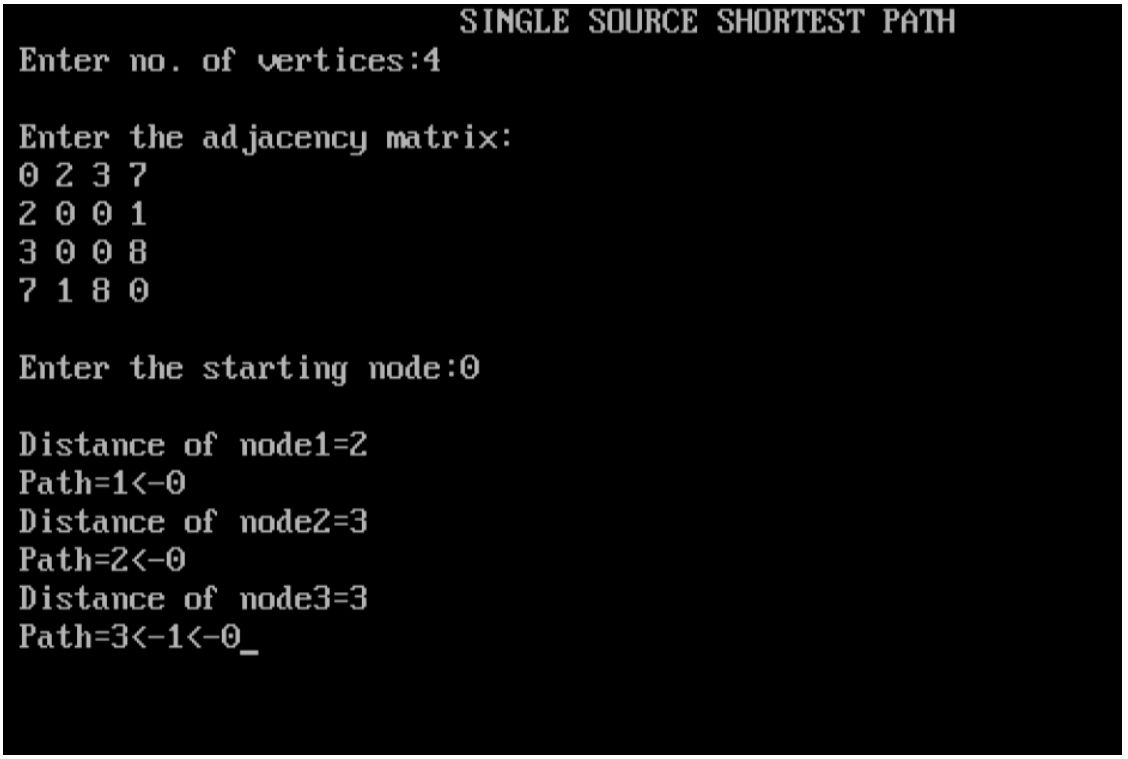
c).Single source shortest path

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
void main()
{
int G[MAX][MAX],i,j,n,u;
clrscr();
printf("\t\t\tSINGLE SOURCE SHORTEST PATH");
printf("\nEnter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
getch();
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
for(i=0;i<n;i++){
```

```
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
```

}

OUTPUT:


SINGLE SOURCE SHORTEST PATH
Enter no. of vertices:4
Enter the adjacency matrix:
0 2 3 7
2 0 0 1
3 0 0 8
7 1 8 0
Enter the starting node:0
Distance of node1=2
Path=1<-0
Distance of node2=3
Path=2<-0
Distance of node3=3
Path=3<-1<-0