**PRIORITY QUEUE USING HEAPS**

**AIM:**

To write a ‘C’ PROGRAM to implement the priority queue using heaps.

**ALGORTHIM:**

* 1. Start the program
  2. Declare the variables
  3. Check the condition of for qfull() and we can insert the elements

if(rear==SIZE-1) we can insert the element

* 1. Check the condition for qempty and delete the elements in the queue

if(front==-1)||(front>rear)) we can delete the element.

* 1. Check the condition for qempty and display the elements in the queue

if(front==-1)||(front>rear)) we can dispaly the element.

* 1. Stop the program

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define SIZE 5

int rear= -1,front=0,que[SIZE],choice;

void insert()

{

int item,j;

printf("\nEnter the Element: ");

scanf("%d",&item);

if(front== -1)

{

front++;

}

j=rear;

while(j>=0&&item<que[j])

{

que[j+1]=que[j];

j--;

}

que[j+1]=item;

rear=rear+1;

}

int Qfull()

{

if(rear==SIZE-1)

{

return 1;

}

else

{

return 0;

}

}

void delet()

{

int item;

item=que[front];

printf("\nThe item deleted is: %d",item);

front++;

}

int Qempty()

{

if((front==-1)||(front>rear))

{

return 1;

}

else

{

return 0;

}

}

void display()

{

int i;

printf("\nThe queue is:\n");

for(i=front;i<=rear;i++)

{

printf("%d\t",que[i]);

}

}

void main()

{

char ans;

clrscr();

do

{

printf("\n Priority Queue");

printf("\n Main Menu");

printf("\n1.Insert\n2.Delete\n3.Display");

printf("\nEnter your Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:if(Qfull())

{

printf("\nQueue is Full");

}

else

{

insert();

}

break;

case 2:if(Qempty())

{

printf("\nQueue is Empty");

}

else

{

delet();

}

break;

case 3:if(Qempty())

{

printf("\nQueue is Empty");

}

else

{

display();

}

break;

default:

printf("\nWrong Choice");

break;

}

printf("\nDo u want to continue?(y for YES/n for NO): ");

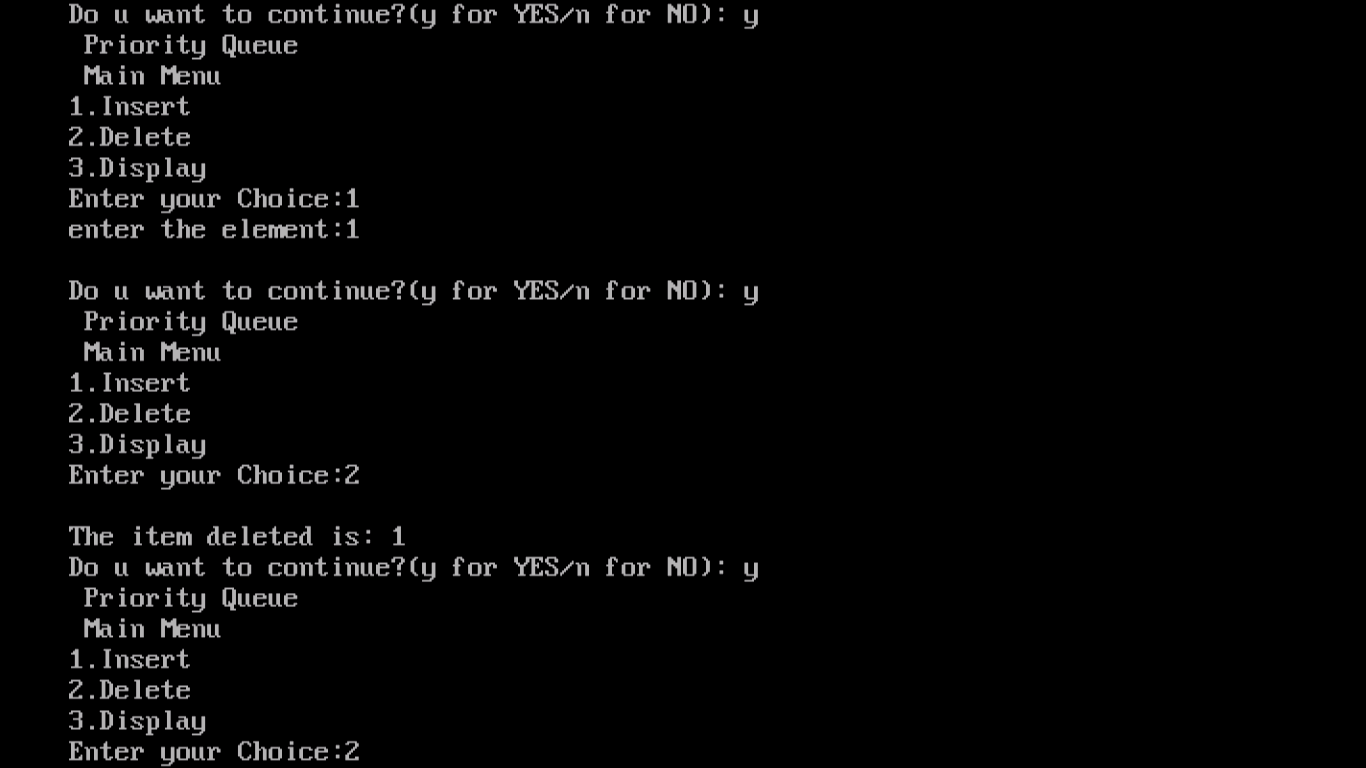
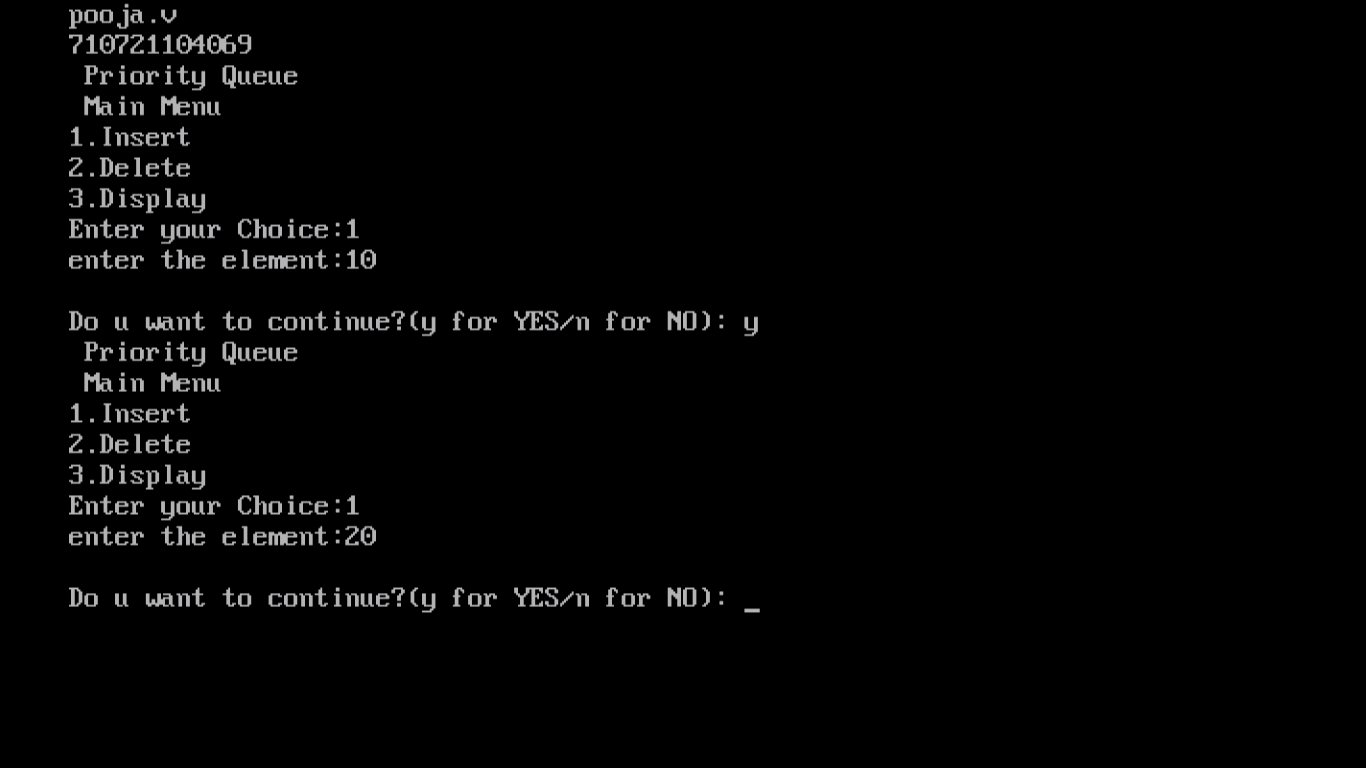
ans=getche();

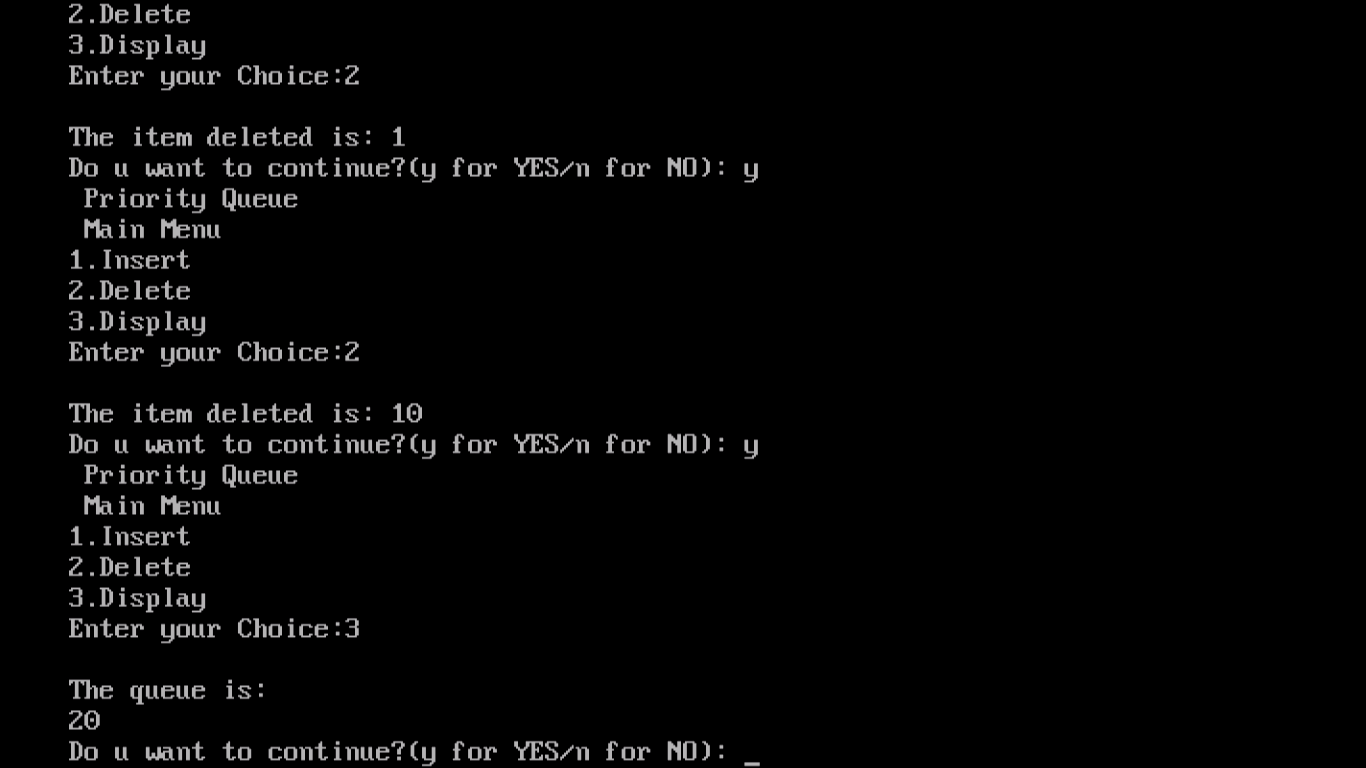
}while(ans=='y');

getch();

}

**OUTPUT:**





**RESULT:**

Thus, the Program for implementation of priority queue using heaps is executed and its output is verified.

**DIJKSTRA’S ALGORITHM**

**AIM:**

To write a ‘C’ PROGRAM to implement the Dijkstra’s algorithm.

**ALGORITHM**

Step1: Start the program.

Step2: Read the number of vertices

Step3: Read the weight of every pair of vertices.

Step4: Get the source vertex & destination.

Step5: Construct the graph.

Step6: Start finding the start node to all other neighboring nodes.

Step7: Nearest path is selected using array until the end node is reached.

Step8: Print the shortest path.

Step9: Terminate the program.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main(){

int G[MAX][MAX],i,j,n,u;

clrscr();

printf("Enter 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 \n",i,distance[i]);

printf("\nPath=%d \n",i);

j=i;

do{

j=pred[j];

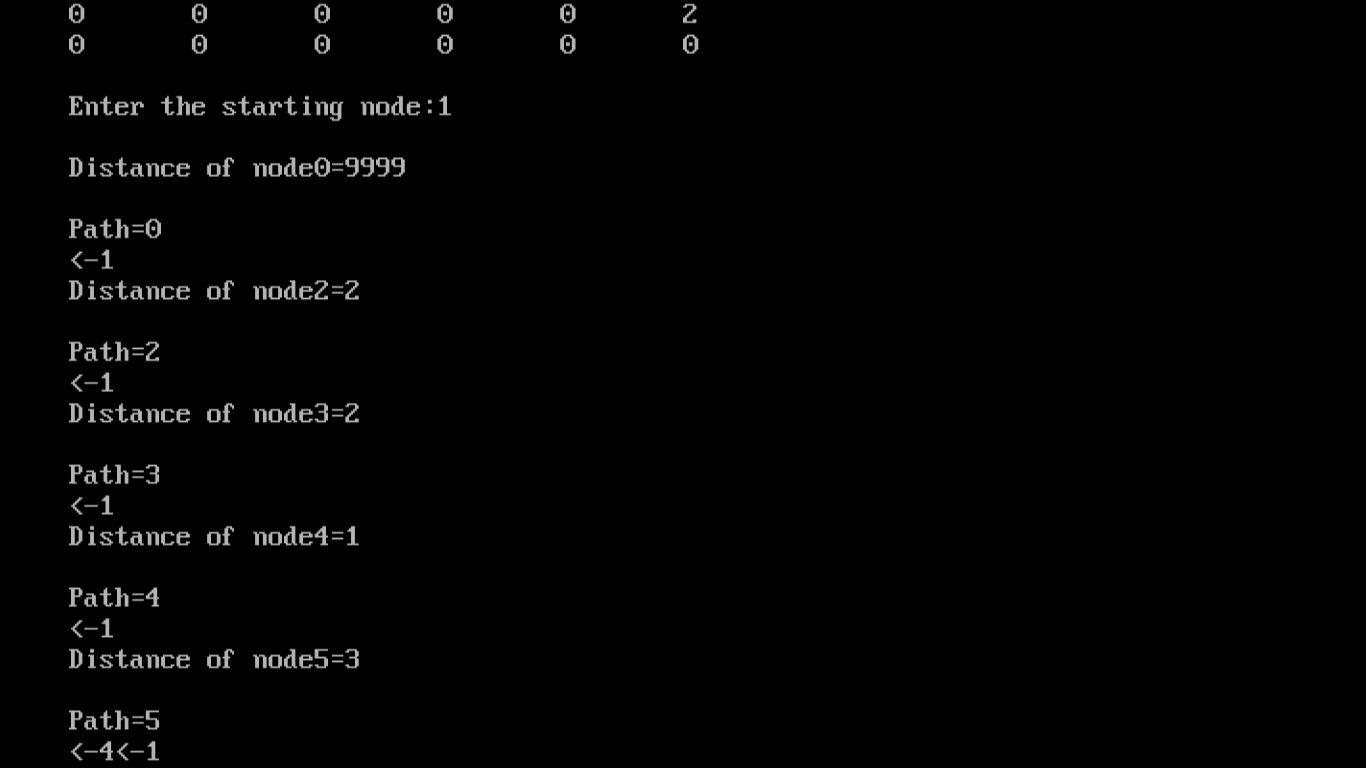
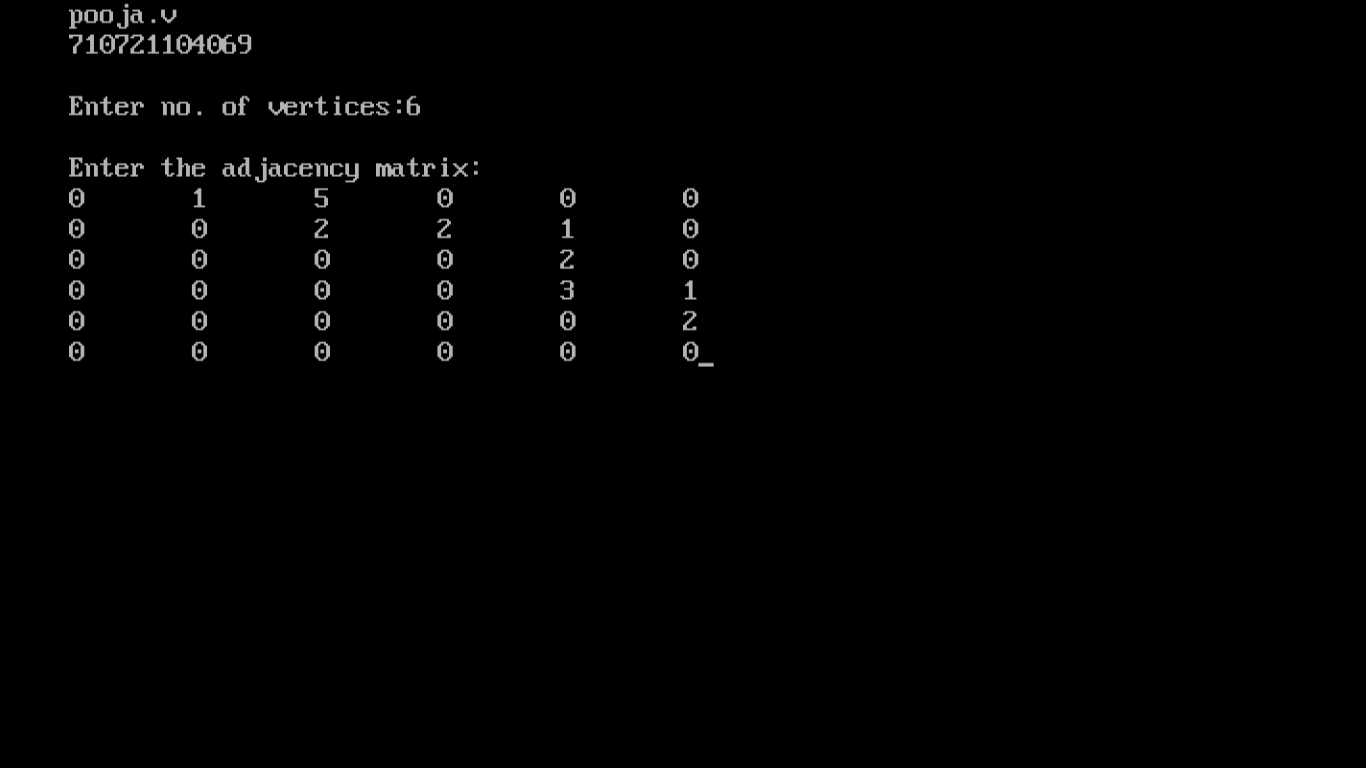
printf("<-%d",j);

}while(j!=startnode);

}

}

**OUTPUT:**

****

**RESULT:**

Thus, the Program for implementation of Dijkstra ‘s algorithm is executed and its output is verified.

**PRIM’S ALGORITM**

**AIM:**

To write a ‘C’ PROGRAM to implement the Prim’s algorithm.

**ALGORTHIM:**

1. Start the program
2. Declare the variables.
3. Enter the edges and its edges in the graph.
4. Construct the graph.
5. Pick one arbitrary vertex and consider it as visited.
6. The unvisited vertices in the new edge are considered.
7. Repeat the step 6 until u cover all the edges.
8. Print the minimum spanning with the corresponding weights.
9. End the program.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

clrscr();

printf("\nEnter the number of nodes:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

for(j=1;j<=n;j++)

if(cost[i][j]< min)

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\nEdge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

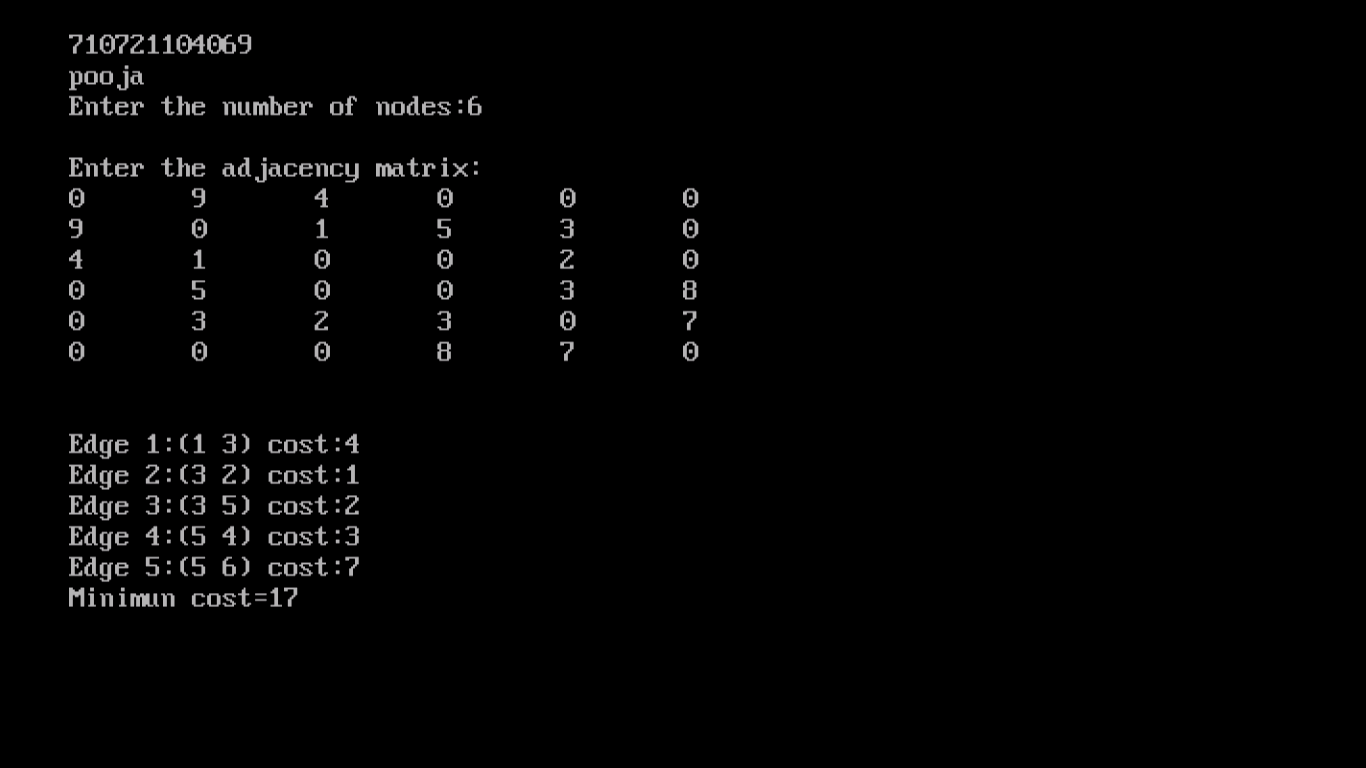
}

printf("\nMinimun cost=%d",mincost);

getch();

}

**OUTPUT:**

****

**RESULT:**

Thus, the Program for implementation of Prim’s algorithm is executed and its output is verified.

**IMPLEMENTATION OF LINEAR SEARCH**

**AIM:**

To write a ‘C’ program to implement linear search.

**ALGORITHM:**

Step 1: Start the program

Step 2: Define function for linear search as

1. Read the data to be searched ‘X’
2. Scan the array from the left to right
3. Compare ‘X’ with the first element
4. If equal then

Print ‘The number is found’ and return

Else

Compare ‘X’ with second element and so on.

Step 3: Stop the program

**PROGRAM:**

#include<stdio.h>

int a[10],i,n,flag=0;

void insert()

{

printf("Enter the size of an array: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("Enter the elements of the array %d:",i+1);

scanf("%d",&a[i]);

}

}

void delet()

{

int del;

printf("Enter the number to be delete: ");

scanf("%d",&del);

for(i=0;i<n;i++){

if(a[i]==del){

a[i]=-1;

flag=1;

break;

}

}

if(flag==0)

printf("The number is not in the list");

else

printf("The number is deleted");

}

void display()

{

printf("\nThe Element are:");

for(i=0;i<n;i++)

{

printf("%d\t",a[i]);

}

}

void search()

{

int key;

printf("Enter the number to be search: ");

scanf("%d",&key);

for(i=0;i<n;i++)

{

if(a[i]==key)

{

flag=1;

break;

}

}

if(flag==0)

printf("The number is not in the list");

else

printf("The number is found");

}

int main()

{

int ch;

while(ch!=5)

{

printf("\n1. Insert \n2. Delete \n3. Display \n4. Search\n5. Exit");

printf("\n Enter your Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delet();

break;

case 3:

display();

break;

case 4:

search();

break;

case 5:

exit(0);

break;

default:

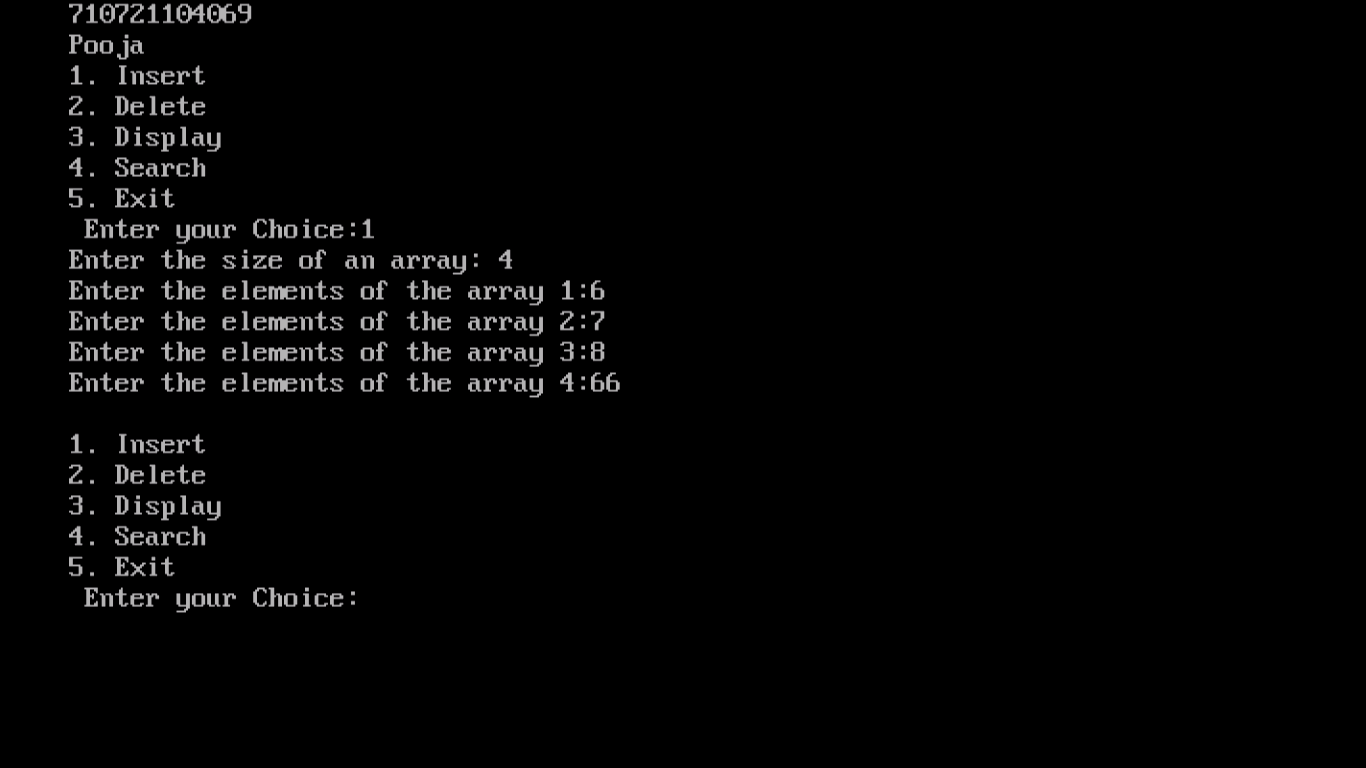
printf("\nInvalid Choice");

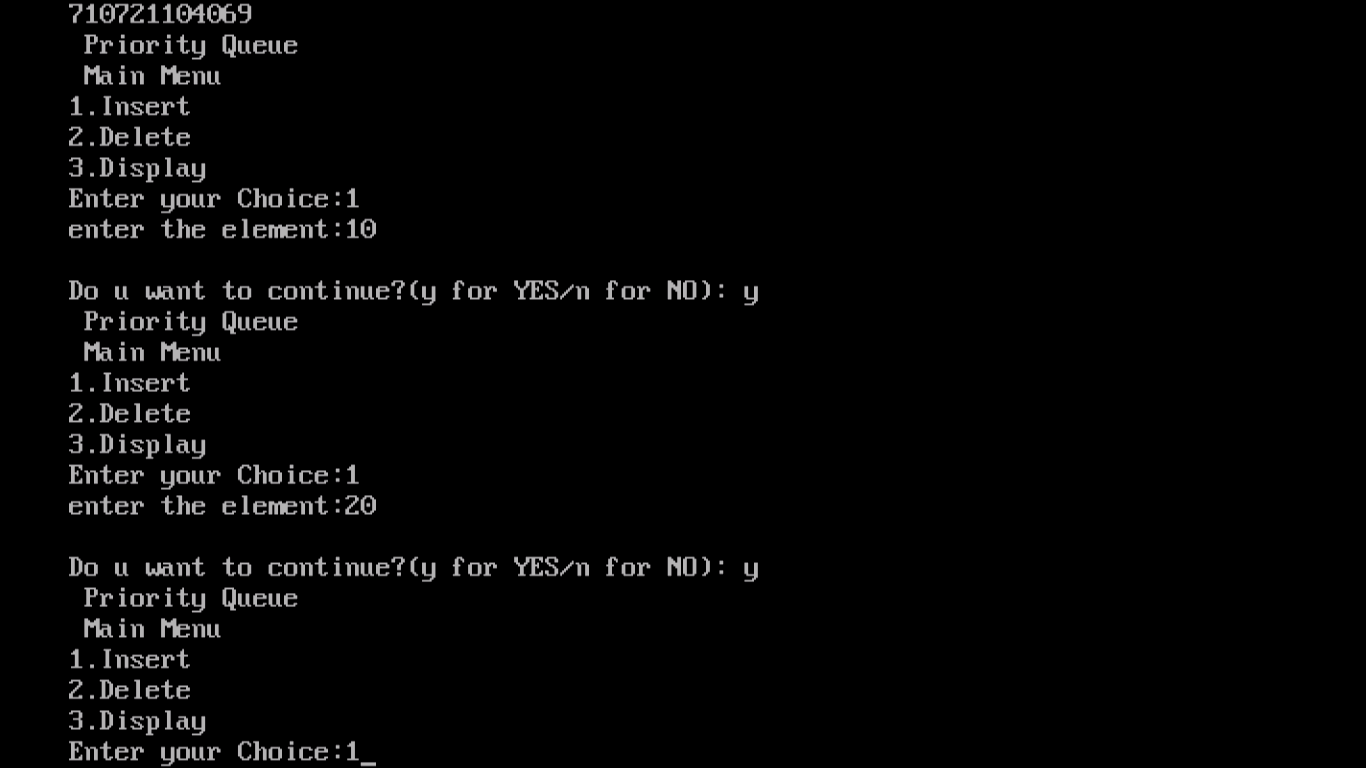
}

}

}

**OUTPUT:**





**RESULT:**

Thus, the C program to implement Linear Search was executed successfully and the output was verified.

**IMPLEMENTATION OF BINARY SEARCH**

**AIM:**

To write a ‘C’ program to implement binary search.

**ALGORITHM:**

Step 1: Start the program

Step 2: Define function for Binary Search as

1. Sort the array in ascending order
2. Let lb=0 and ub=n-1
3. Read the data to be searched ‘X’
4. Find the mid position of the given array

Mid=(lb+ub)/2

1. Compare X with a[mid]

If equal then

Goto step (g)

Else

If X less than a[mid] then ub=mid-1

If X greater than a[mid] then lb=mid+1

1. If lb<=ub

Repeat steps (d) and (e) for the sub array lb to ub

Else

Goto step (g)

1. If(lb>ub)

Print “Search Success”

Else

Print “Search Failed”

1. Return

Step3: Stop the program.

**PROGRAM**

#include<stdio.h>

int main()

{

int a[10],i,n,m,c=0,l,u,mid;

printf("\nEnter the size of an array: ");

scanf("%d",&n);

printf("\nEnter the elements in ascending order");

for(i=0;i<n;i++){

printf("\nEnter the %d elements:", i+1);

scanf("%d",&a[i]);

}

printf("\nEnter the number to be search: ");

scanf("%d",&m);

l=0,u=n-1;

while(l<=u){

mid=(l+u)/2;

if(m==a[mid]){

c=1;

break;

}

else if(m<a[mid]){

u=mid-1;

}

else

l=mid+1;

}

if(c==0)

printf("\nThe number is not found.");

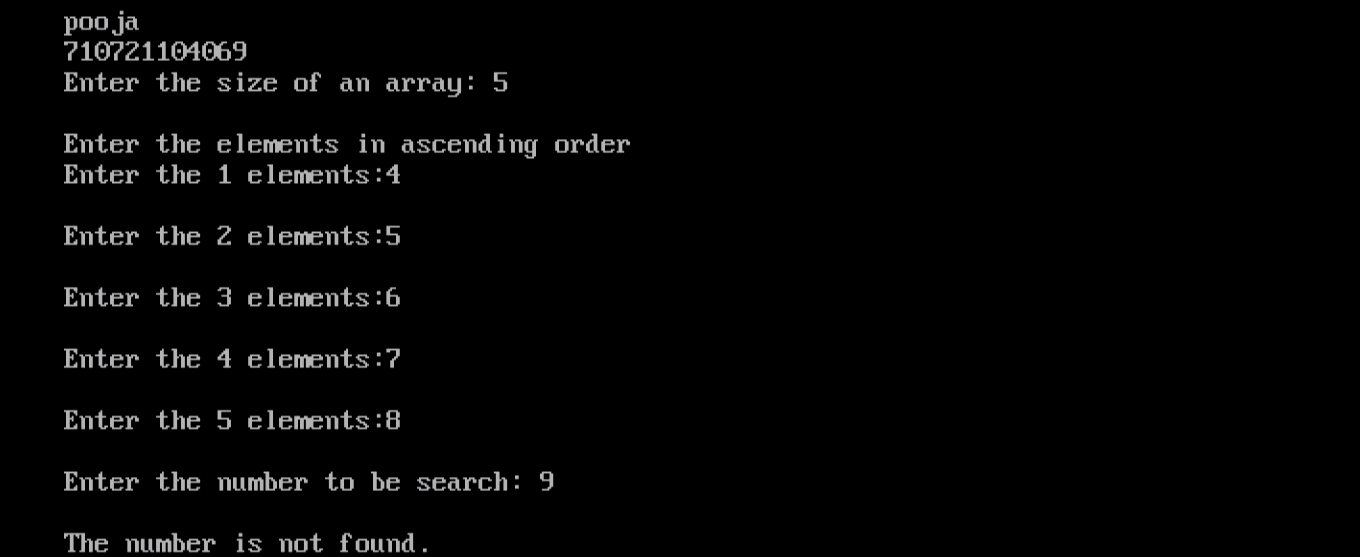
else

printf("\nThe number is found.");

return 0;

}

**OUTPUT:**

****

**RESULT:**

Thus, the C program to implement Binary Search was executed successfully and the output was verified.

**INSERTION SORT**

**AIM:**

To write a ‘C’ program to implement insertion sort.

**ALGORITHM:**

Step 1: Read the elements into the array

Step 2: Take the second element. Compare it with the first element. If the second element less than the first element interchange them

Step 3: Take the third element compare it first and second element and insert it in the correct position by shifting the elements in the array. So that the first, second and third elements are in sorted array

Step 4: In general, take the ith element and compare it with the all the elements before it and place it in the proper position by shifting the elements one position right.

Step 5: When the ith element is placed, the elements in the array from the 0th to the ith position will be in the sorted order

Step 6: The above process is continued for all the elements in the array.

Step 7: Stop.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void insert(int[],int);

void main(){

int a[20],i,n;

clrscr();

printf("\nEnter the size of an array:");

scanf("%d",&n);

for(i=0;i<n;i++){

printf("\nEnter the %d element in the array:",i+1);

scanf("%d",&a[i]);

}

insert(a,n);

getch();

}

void insert(int a[],int n){

int i,j,temp;

for(i=1;i<n;i++){

temp=a[i];

for(j=i-1;j>=0;j--){

if(a[j]>temp){

a[j+1]=a[j];

}

else

break;

}

a[j+1]=temp;

}

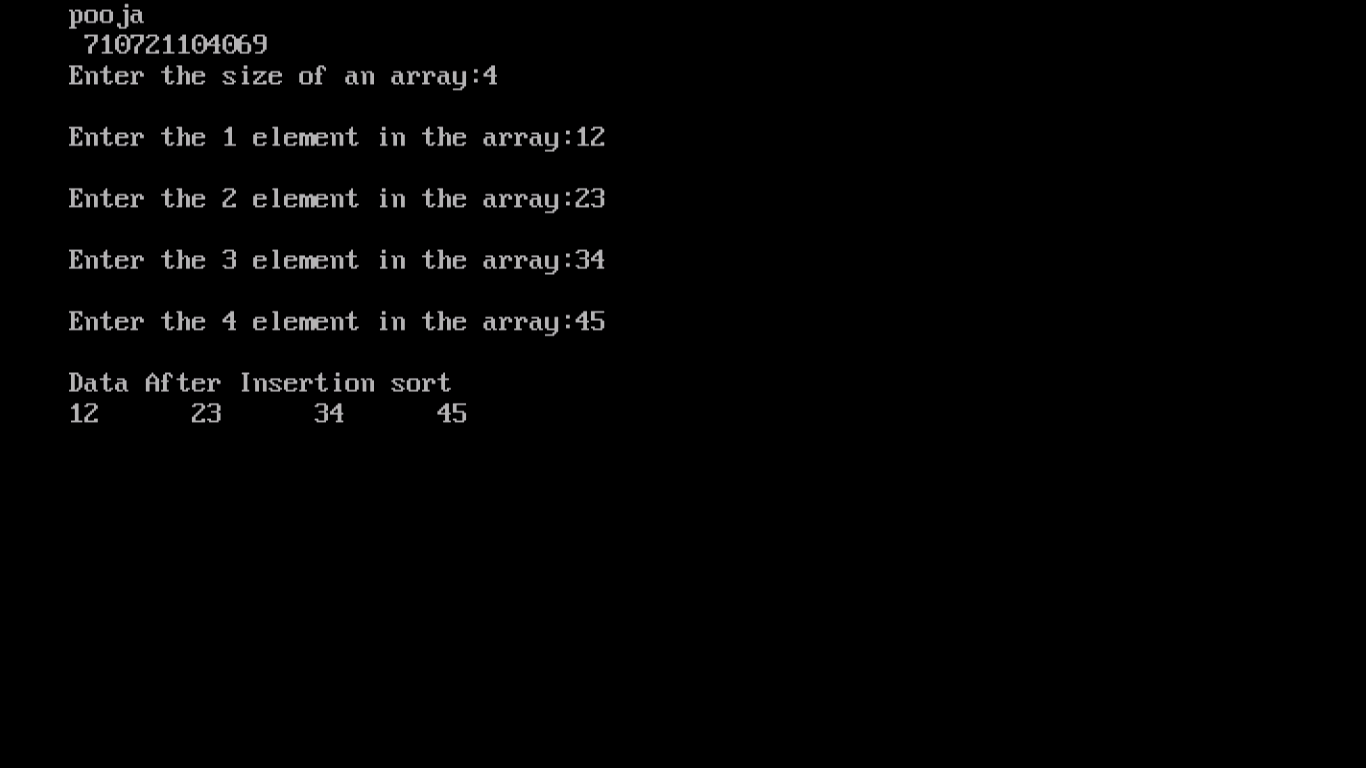
printf("\nData After Insertion sort\n");

for(i=0;i<n;i++)

printf("%d\t", a[i]);

}

**OUTPUT:**



**RESULT:**

Thus, the C program to implement Insertion Sort was executed successfully and the output was verified.

**SELECTION SORT**

**AIM:**

To write a “C++‟ program to implement Selection Sort.

**ALGORITHM:**

Step1 : Start the program.

Step2 : Set MIN to location 0

Step3 : Search the minimum element in the list

Step4 : Swap with value at location MIN

Step5 : Increment MIN to point to next element

Step6 : Repeat until list is sorted

Step7 : Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void SelectionSort (int arr[], int n)

{

int i, j;

for (i = 0; i < n; ++i)

{

for (j = i+1; j < n; ++j)

{

if (arr[i] > arr[j])

{

arr[i] = arr[i]+arr[j];

arr[j] = arr[i]-arr[j];

arr[i] = arr[i]-arr[j];

}

}

}

}

int main()

{

clrscr();

int n, i;

printf("\nEnter the number of data element to be sorted: ");

scanf("%d",&n);

int arr[10];

for(i = 0; i <n ; i++)

{

printf("Enter %d element:",i+1);

scanf("%d",&arr[i]);

}

SelectionSort(arr, n);

printf("\nSorted Data:\n");

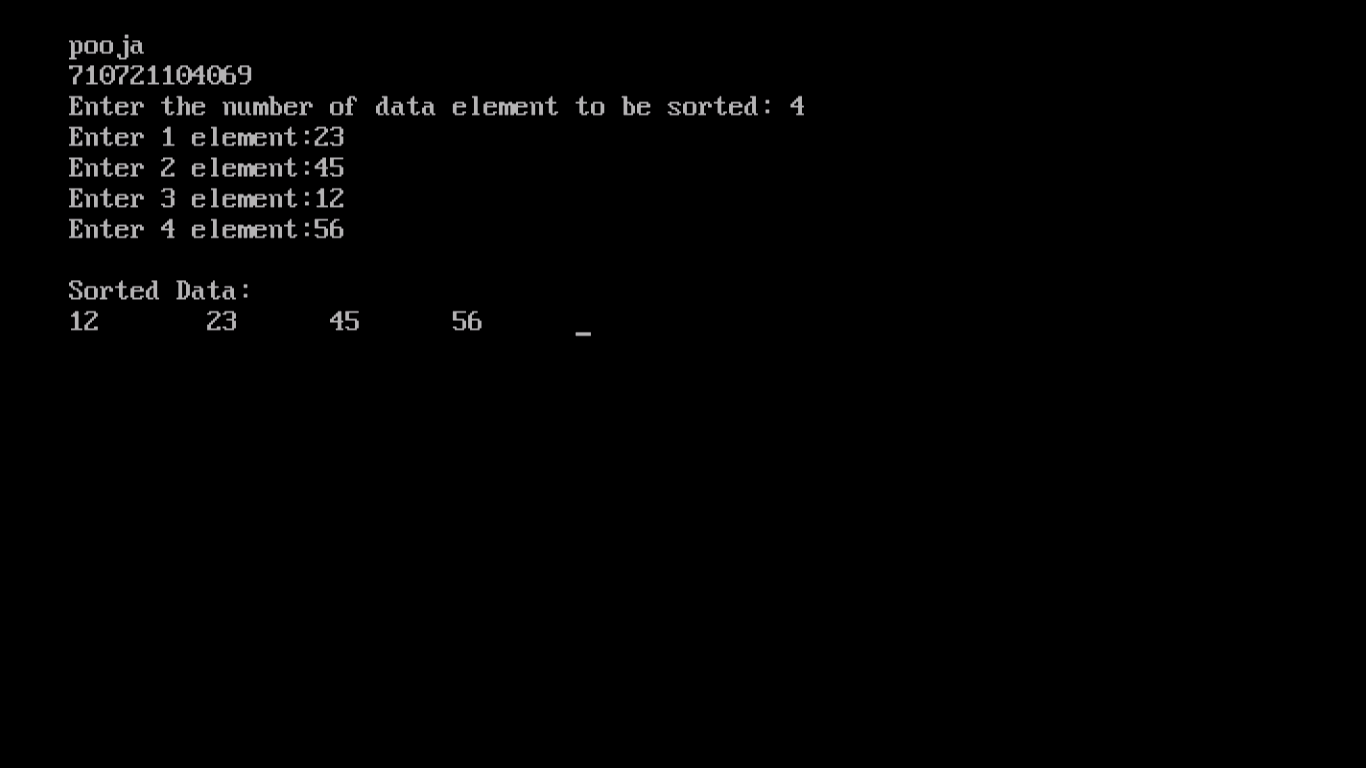
for (i = 0; i < n; i++)

printf("%d \t ",arr[i]);

getch();

}

**OUTPUT:**

****

**RESULT:**

Thus, the C++ program to implement Selection Sort was executed successfully and the output was verified.

**MERGE SORT**

**AIM:**

To write a ‘C’ program to implement merge sort.

**ALGORITHM:**

Step 1: Read the elements into the array

Step 2: Take the second element. Compare it with the first element. If the second element less than the first element interchange them

Step 3: Take the third element compare it first and second element and insert it in the correct position by shifting the elements in the array. So that the first, second and third elements are in sorted array

Step 4: In general take the ith element and compare it with the all the elements before it and place it in the proper position by shifting the elements one position right.

Step 5: When the ith element is placed, the elements in the array from the 0th to the ith position will be in the sorted order

Step 6: The above process is continued for all the elements in the array.

Step 7: Stop.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void merge(int [],int ,int ,int );

void part(int [],int ,int );

int main()

{

int arr[30];

int i,size;

printf("Enter total no. of elements : ");

scanf("%d",&size);

for(i=0; i<size; i++)

{

printf("Enter %d element: ",i+1);

scanf("%d",&arr[i]);

}

part(arr,0,size-1);

printf("\n\tSorted elements\n");

for(i=0; i<size; i++)

printf("%d\t",arr[i]);

getch();

return 0;

}

void part(int arr[],int min,int max)

{

int mid;

if(min<max)

{

mid=(min+max)/2;

part(arr,min,mid);

part(arr,mid+1,max);

merge(arr,min,mid,max);

}

}

void merge(int arr[],int min,int mid,int max)

{

int tmp[30];

int i,j,k,m;

j=min;

m=mid+1;

for(i=min; j<=mid && m<=max ; i++)

{

if(arr[j]<=arr[m])

{

tmp[i]=arr[j];

j++;

}

else

{

tmp[i]=arr[m];

m++;

}

}

if(j>mid)

{

for(k=m; k<=max; k++)

{

tmp[i]=arr[k];

i++;

}

}

else

{

for(k=j; k<=mid; k++)

{

tmp[i]=arr[k];

i++;

}

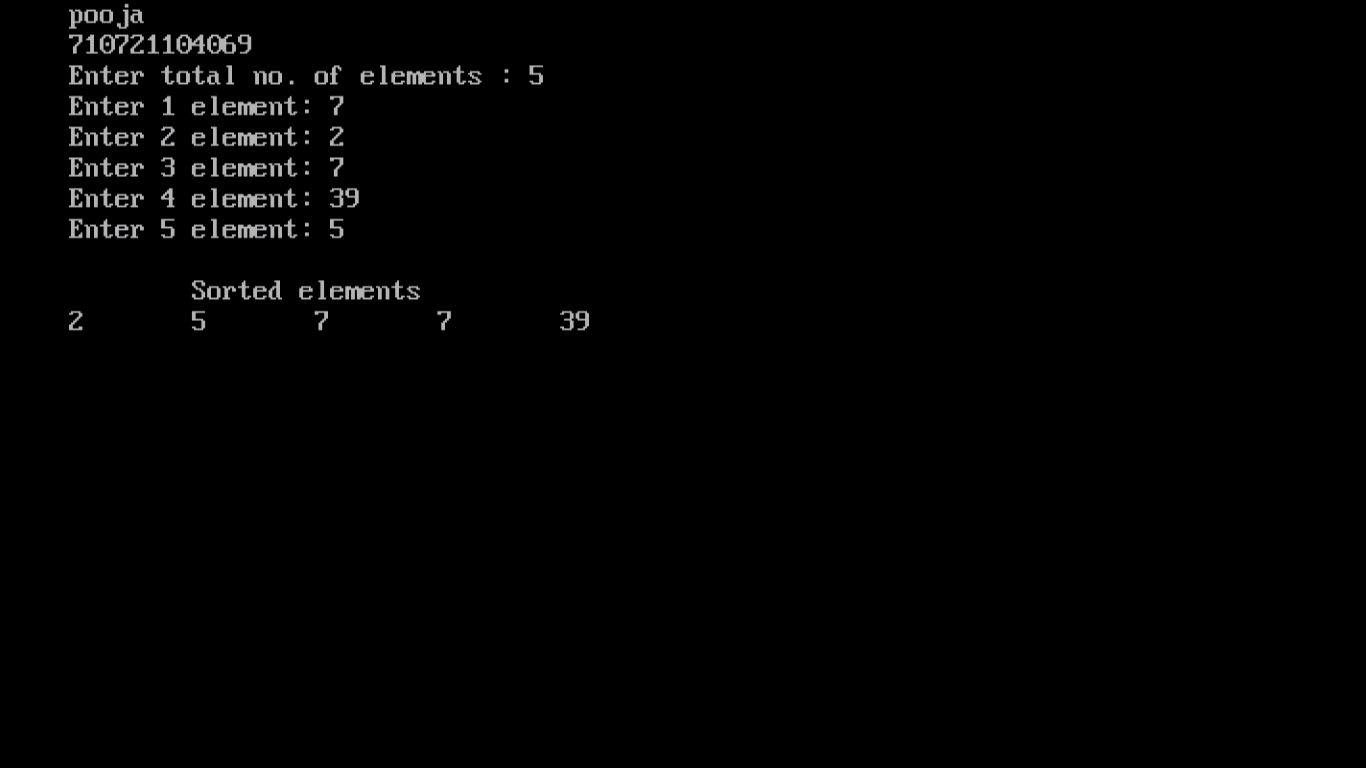
}

for(k=min; k<=max; k++)

arr[k]=tmp[k];

}

**OUTPUT:**

****

**RESULT:**

Thus, the C program to implement Merge Sort was executed successfully and the output was verified.

**OPEN ADDRESSING LINEAR PROBING**

**AIM:**

To write a ‘C’ PROGRAM to implement the linear probing hashing technique.

**ALGORTHIM:**

* 1. start the program
  2. Declare the variables

3. Check the condition for loop for linear probing and set FLAG as 0 and COUNT as 0

4. By increment of count we can insert the element in the hash table.

* 1. check the condition for full and display the elements in the hash table

if(count==MAX), by help of for loop we can display the elements.

* 1. stop the program

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define MAX 10

int create(int num)

{

int key;

key=num%10;

return key;

}

void display(int a[MAX])

{

int i;

printf("\nThe Hash Table is ....\n");

for(i=0;i<MAX;i++)

{

printf("\n %d %d",i,a[i]);

}

}

void linearprob(int a[MAX],int key,int num)

{

int flag=0,i,count=0;

if(a[key]= = -1)

{

a[key]=num;

}

else

{

i=0;

while(i<MAX)

{

if(a[i]!=-1)

{

count++;

}

i++;

}

if(count= =MAX)

{

printf("\nHash Table is Full");

display(a);

getch();

exit(1);

}

for(i=key+1;i<MAX;i++)

{

if(a[i]= = -1)

{

a[i]=num;

flag=1;

break;

}

}

for(i=0;i<key&&flag= = 0;i++)

{

if(a[i]= = -1)

{

a[i]=num;

flag=1;

break;

}

}

}

}

void main()

{

int a[max],num,key,i;

char ans;

clrscr();

printf("\nCollision Handling by Linear Probing");

for(i=0;i<MAX;i++)

{

a[i]=-1;

}

do

{

printf("\nEnter the Number ");

scanf("%d",&num);

key=create(num);

linearprob(a,key,num);

printf("\nDo u want to continue?(y for YES/n for NO): ");

ans=getche();

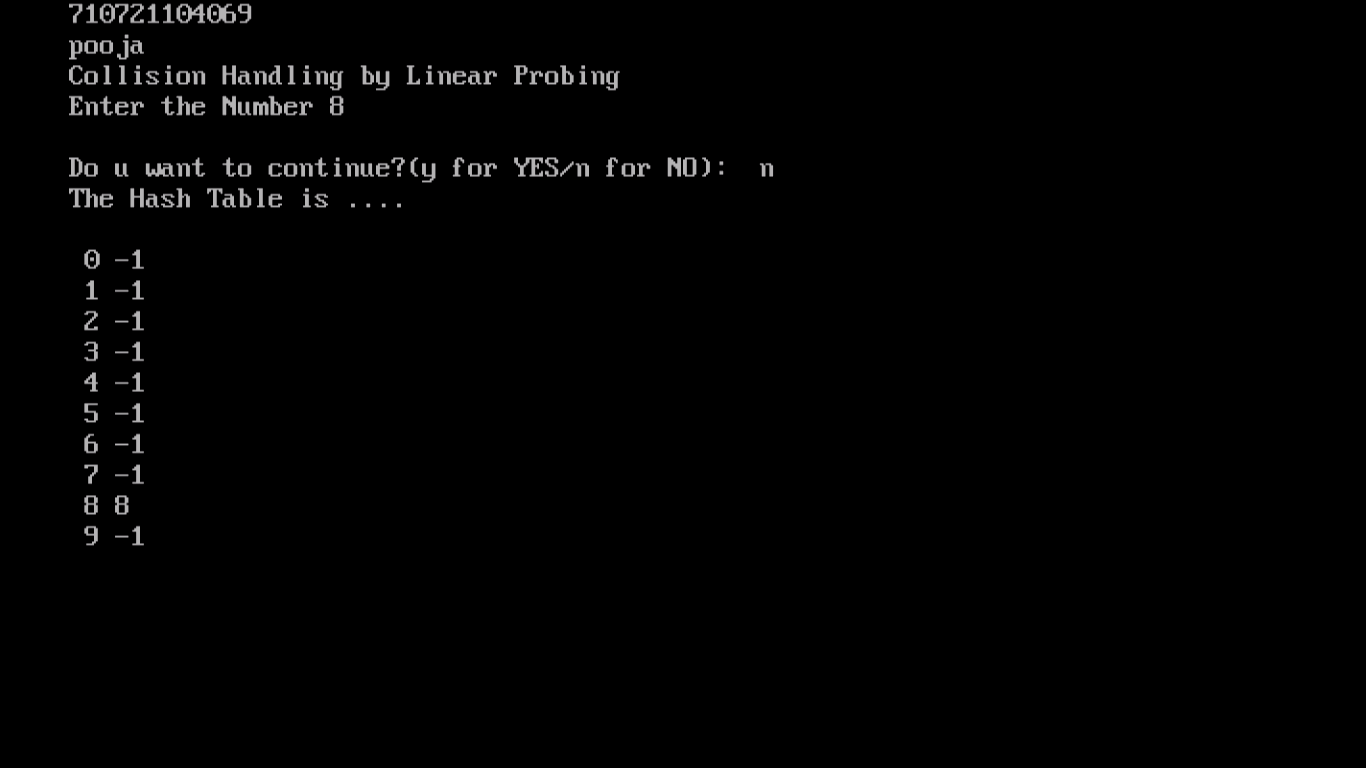
}while(ans= = 'y');

display(a);

getch();

}

**OUTPUT:**

****

**RESULT:**

Thus, the Program for implementation of linear probing hashing technique is executed and its output is verified.

**OPEN ADDRESSING QUADRATIC PROBING**

**AIM:**

To write a ‘C’ PROGRAM to implement the quadratic probing hashing technique.

**ALGORTHIM:**

* 1. start the program
  2. Create an array of structure (i.e a hash table).
  3. Take a key and a value to be stored in hash table as input.
  4. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.
  5. Using the generated index, access the data located in that array index.
  6. In case of absence of data, create one and insert the data item (key and value) into it and increment the size of hash table.
  7. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.

1. Note: This probing will continue until we reach the same element again (from where we began probing)
2. Note: Here, unlike Linear Probing, probing will be done according to the following formula –
3. (currentPosition + h) % arraySize => Linear Probing
4. (currentPosition + (h \* h)) % arraySize => Quadratic Probing
5. where h = 1, 2, 3, 4 and so on.
   1. To display all the elements of hash table, element at each index is accessed (via for loop).
   2. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).
   3. Stop

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#define MAX 10

void printArray(int arr[], int n)

{

printf("The Quadratic Probing Hash table");

for (int i = 0; i < n; i++)

{

printf("\n[%d] %d",i,arr[i]);

}

}

void hashing(int table[], int tsize,

int arr[], int N)

{

for (int i = 0; i < N; i++)

{

int hv = arr[i] % tsize;

if (table[hv] == -1)

table[hv] = arr[i];

else

{

for (int j = 0; j < tsize; j++)

{

int t = (hv + j \* j) % tsize;

if (table[t] == -1)

{

table[t] = arr[i];

break;

}

}

}

}

printArray(table, N);

}

void main()

{

int arr[MAX],i,n,s=10;

clrscr();

printf("\nEnter the size of n:");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\nEnter the Number: ");

scanf("%d",&arr[i]);

}

int hash\_table[MAX];

for (int i = 0; i < s; i++)

{

hash\_table[i] = -1;

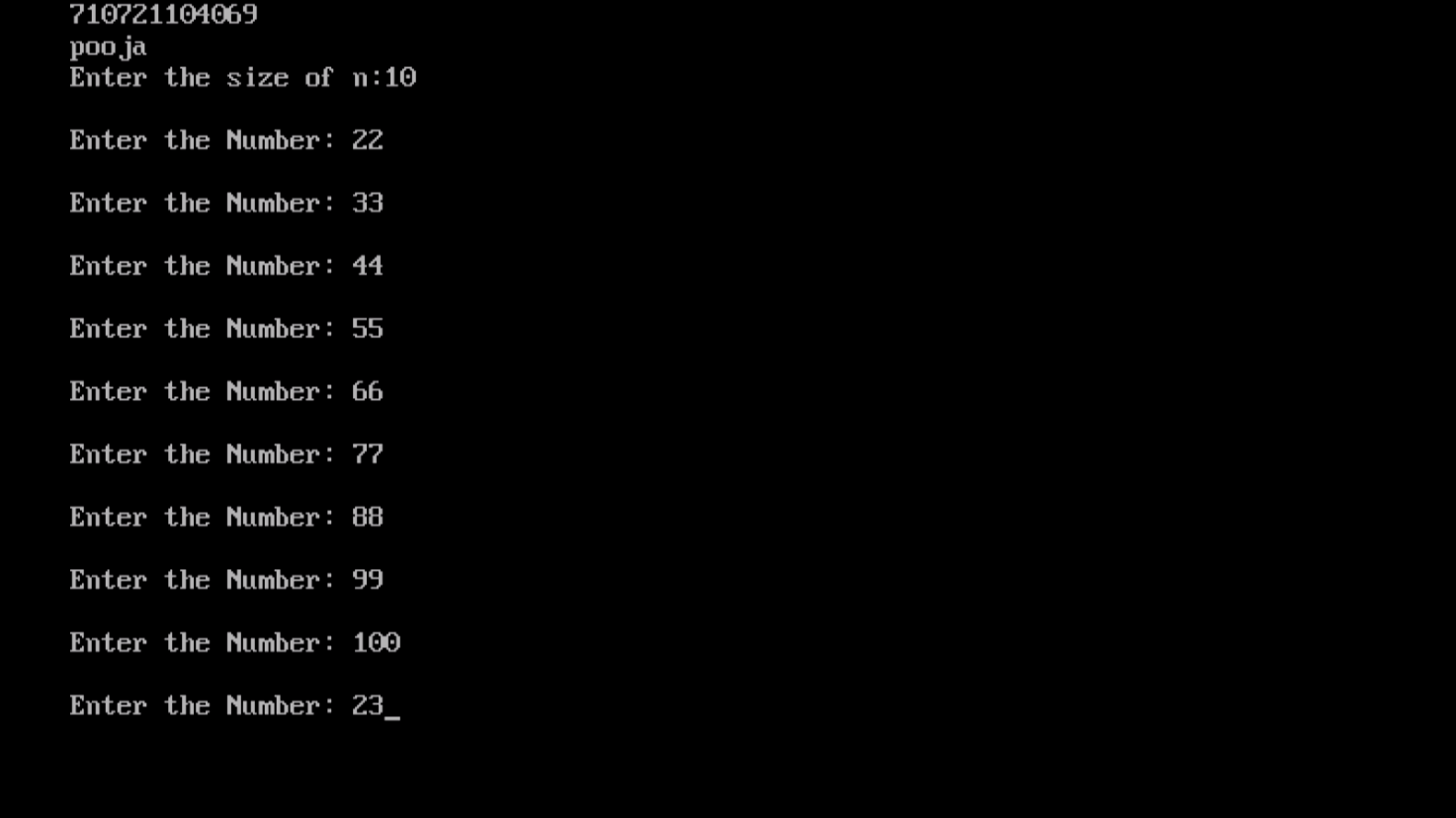
}

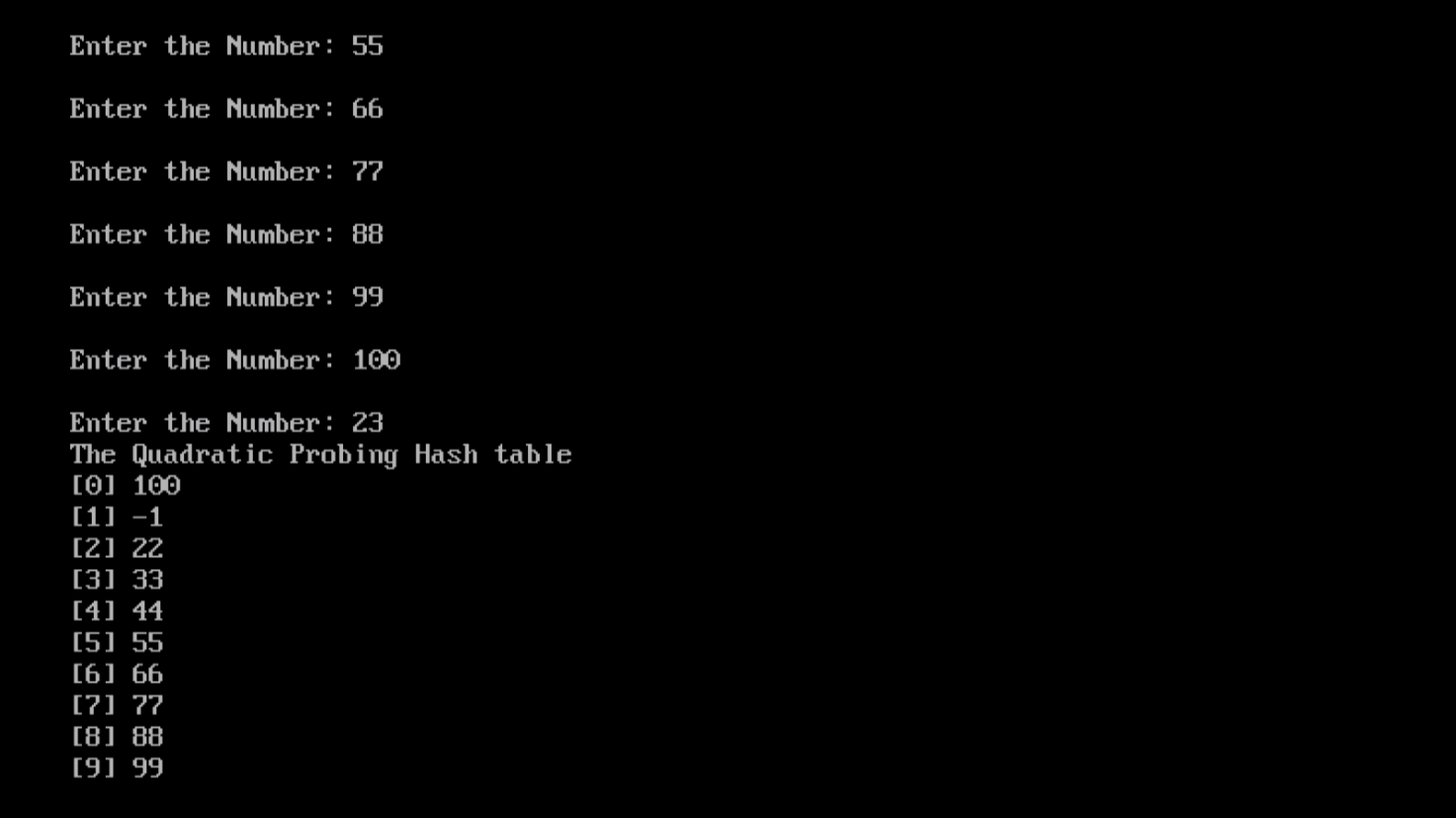
hashing(hash\_table, s, arr, n);

getch();

}

**OUTPUT:**

****

****

**RESULT:**

Thus, the Program for implementation of quadratic probing hashing technique is executed and its output is verified.