# LAB MANUAL

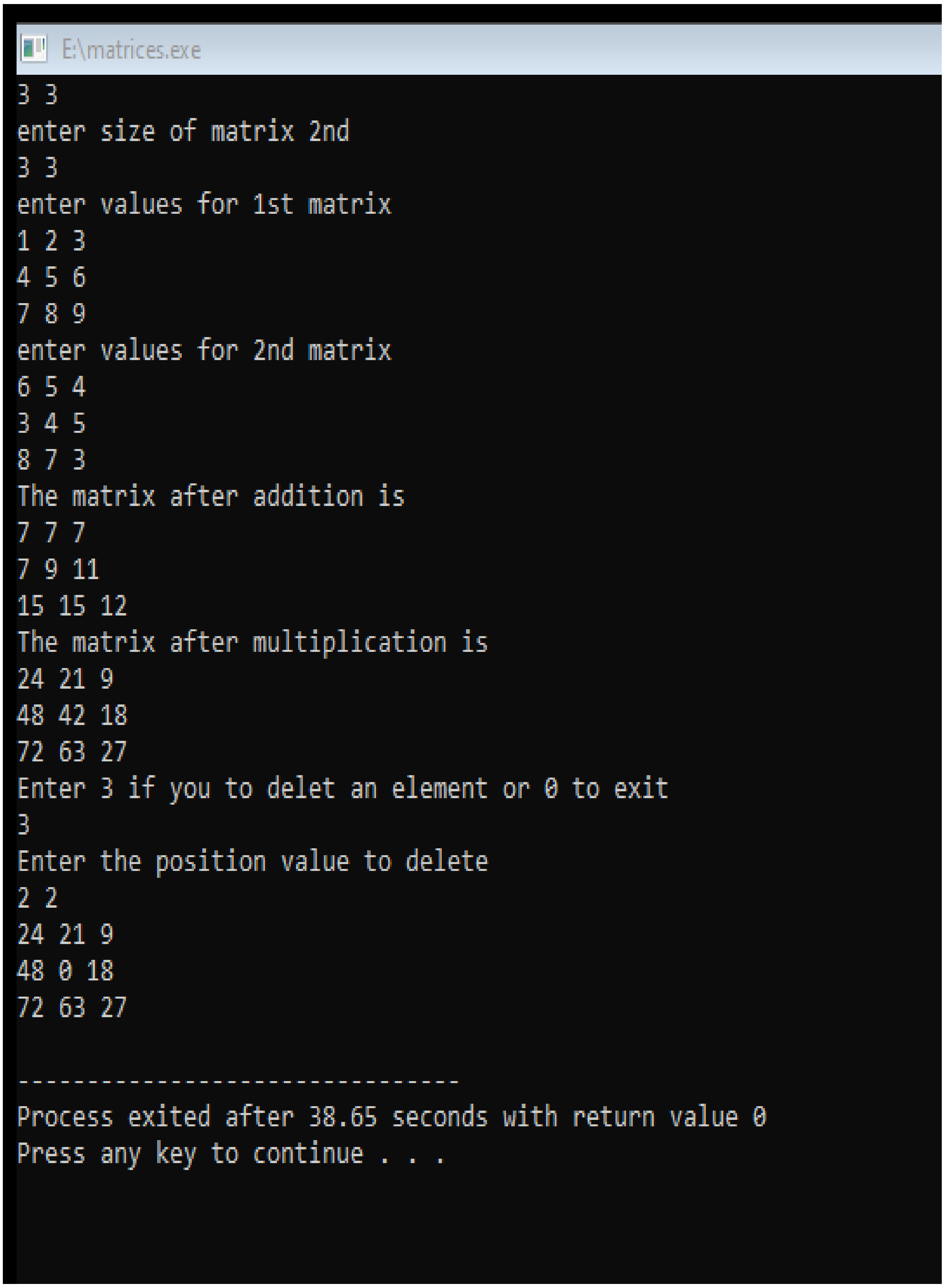
# DATA STRUCTURE & ALGORITHMS USING C/C++

| Sr. no. | Program | Date | Signature |
|---|---|---|---|
| 1 | To create and manipulating of matrices<br>a) Addition<br>b) Multiplication<br>c) Deletion | | |
| 2 | To Insert and Delete elements in Stack | | |
| 3 | To Insert and Delete elements in Queue | | |
| 4 | To Traverse a Binary Tree in Pre order , In order and Post order manner | | |
| 5 | To Implement<br>a) Linear Search<br>b) Binary search | | |
| 6 | To Implement<br>a) Bubble Sort<br>b) Selection Sort<br>c) Insertion Sort<br>d) Quick Sort | | |
| 7 | To Insert and Delete elements in a Binary Search Tree | | |
| 8 | To Implement<br>a) Breadth First Traversal (BSF)<br>b) Dept First Traversal ( DSF) | | |
| 9 | To Create a Link List , Insert and Delete elements in Link List | | |
| 10 | To Implement Heap Sort | | |

# Program 1

**Program to create and manipulating of matrices:-**

```
#include <iostream>
using namespace std;
int a[10][10],b[10][10],c[10][10],i,j,k,m,n,p,q;
int del2(int posr,int posc)
{c[posr][posc]=0;
 }
 void display()
  {  for(int i=0;i<m;i++)
      {for(int j=0;j<n;j++)
       cout<<c[i][j]<<" ";
       cout<<endl;  } }
int main()
{int l,ch,m1,n1;
 cout<<"enter size of matrix 1st "<<endl;
 cin>>m>>n;
 cout<<"enter size of matrix 2nd "<<endl;
 cin>>p>>q;
   cout<<"enter values for 1st matrix "<<endl;
   for(i=0;i<m;i++)
    for(j=0;j<n;j++)
     cin>>a[i][j];
   cout<<"enter values for 2nd matrix "<<endl;
   for(i=0;i<m;i++)
    for(j=0;j<n;j++)
     cin>>b[i][j];
    for(i=0;i<m||i<p;i++)
     {for(j=0;j<n||j<q;j++)
   c[i][j]=a[i][j]+b[i][j]; }
  cout<<"The matrix after addition is"<<endl;
   display();
   for(i=0;i<m||i<p;i++)
   for(j=0;j<n||j<q;j++)
   for(k=0;k<n||k<q;k++)
   { c[i][j]=0;
   c[i][j]+=a[i][k]*b[k][j]; }
    cout<<"The matrix after multiplication is"<<endl;
    display();
    co  tyut<<"Enter 3 if you to delet an element or 0 to exit"<<endl;
    cin>>l;
    if(l=3)
   {cout<<"Enter the position value to delete"<<endl;
    cin>>m1>>n1;
    del2(m1-1,n1-1; }
display();}
```

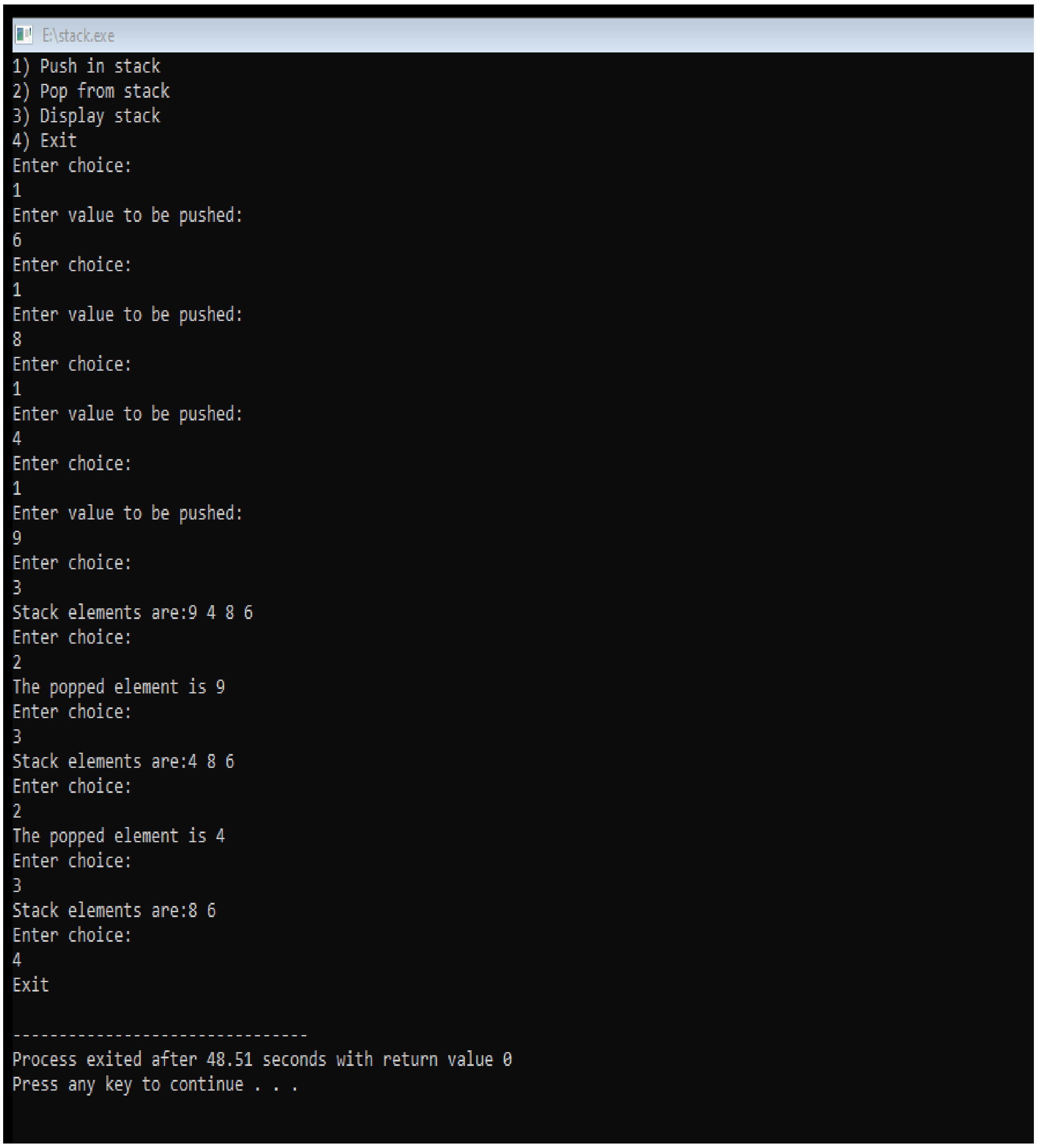


```
3 3
enter size of matrix 2nd
3 3
enter values for 1st matrix
1 2 3
4 5 6
7 8 9
enter values for 2nd matrix
6 5 4
3 4 5
8 7 3
The matrix after addition is
7 7 7
7 9 11
15 15 12
The matrix after multiplication is
24 21 9
48 42 18
72 63 27
Enter 3 if you to delet an element or 0 to exit
3
Enter the position value to delete
2 2
24 21 9
48 0 18
72 63 27


--------------------------------
Process exited after 38.65 seconds with return value 0
Press any key to continue . . .
```

# Program 2

**Program to Insert and Delete elements in Stack**

```
#include <iostream>
using namespace std;
int stack[100], n=100, top=-1;
void push(int val)
 { if(top>=n-1)
   cout<<"Stack Overflow"<<endl;
    else
  {top++;
       stack[top]=val; }}
 void pop()
 {if(top<=-1)
   cout<<"Stack Underflow"<<endl;
   else
 {cout<<"The popped element is "<< stack[top] <<endl;
      top--; }}
void display()
 {if(top>=0)
 {cout<<"Stack elements are:";
   for(int i=top; i>=0; i--)
   cout<<stack[i]<<" ";
   cout<<endl;  }
else
   cout<<"Stack is empty";}
int main()
 {int ch, val;
   cout<<"1) Push in stack"<<endl;
   cout<<"2) Pop from stack"<<endl;
  cout<<"3) Display stack"<<endl;
  cout<<"4) Exit"<<endl;
  do {
     cout<<"Enter choice: "<<endl;
     cin>>ch;
     switch(ch)
    {case 1:
       {cout<<"Enter value to be pushed:"<<endl;
         cin>>val;
         push(val);
         break;}
        case 2:
{ pop();
           break;   }
        case 3:
      {display();
           break;  }
        case 4:
       {cout<<"Exit"<<endl;
           break;  }
        default:
       {cout<<"Invalid Choice"<<endl;    } }
  }while(ch!=4);
  return 0;}
```



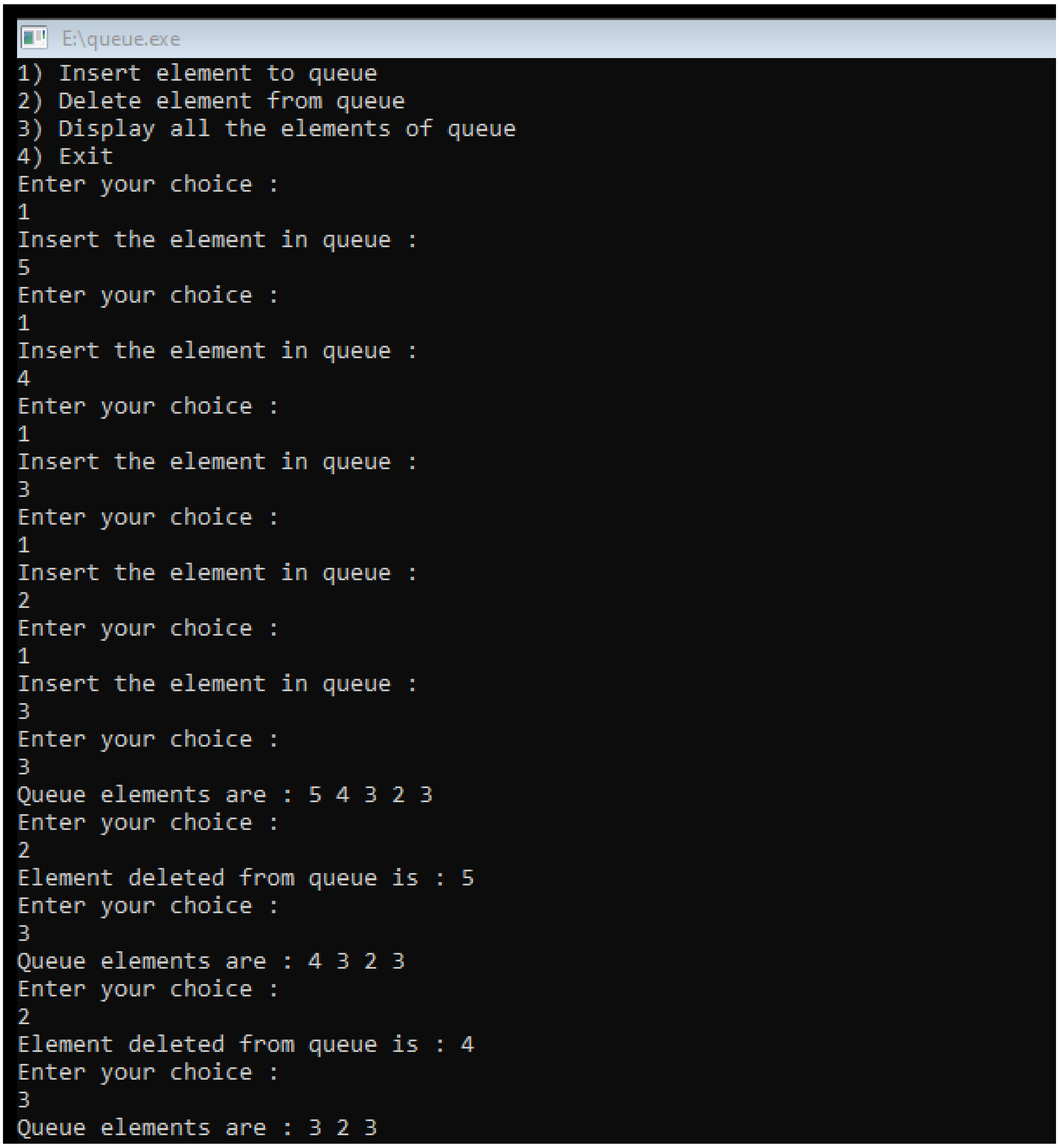
```
1) Push in stack
2) Pop from stack
3) Display stack
4) Exit
Enter choice:
1
Enter value to be pushed:
6
Enter choice:
1
Enter value to be pushed:
8
Enter choice:
1
Enter value to be pushed:
4
Enter choice:
1
Enter value to be pushed:
9
Enter choice:
3
Stack elements are:9 4 8 6
Enter choice:
2
The popped element is 9
Enter choice:
3
Stack elements are:4 8 6
Enter choice:
2
The popped element is 4
Enter choice:
3
Stack elements are:8 6
Enter choice:
4
Exit

--------------------------------
Process exited after 48.51 seconds with return value 0
Press any key to continue . . .
```

# Program 3

**Program to Insert and Delete elements in Queue**

```
#include <iostream>
using namespace std;
int queue[6], n =6 , front = - 1, rear = - 1;
void Insert()
 { int val;
   if (rear == n - 1)
   cout<<"Queue Overflow"<<endl;
   else
    {if (front == - 1)
      front = 0;
      cout<<"Insert the element in queue : "<<endl;
      cin>>val;
      rear++;
      queue[rear] = val;  }}
void Delete()
 {if (front == - 1 || front > rear)
    {cout<<"Queue Underflow "; }
   else
    {cout<<"Element deleted from queue is : "<< queue[front] <<endl;
      front++; }}
void Display()
{if (front == - 1)
   cout<<"Queue is empty"<<endl;
   else
   {cout<<"Queue elements are : ";
      for (int i = front; i <= rear; i++)
      cout<<queue[i]<<" ";
```

```
        cout<<endl; }}
int main()
 {int ch;
   cout<<"1) Insert element to queue"<<endl;
   cout<<"2) Delete element from queue"<<endl;
   cout<<"3) Display all the elements of queue"<<endl;
   cout<<"4) Exit"<<endl;
   do {
      cout<<"Enter your choice : "<<endl;
      cin>>ch;
      switch (ch)
     { case 1: Insert();
        break;
        case 2: Delete();
        break;
        case 3: Display();
        break;
        case 4: cout<<"Exit"<<endl;
        break;
        default: cout<<"Invalid choice"<<endl;  }
   } while(ch!=4);
   return 0;}
```



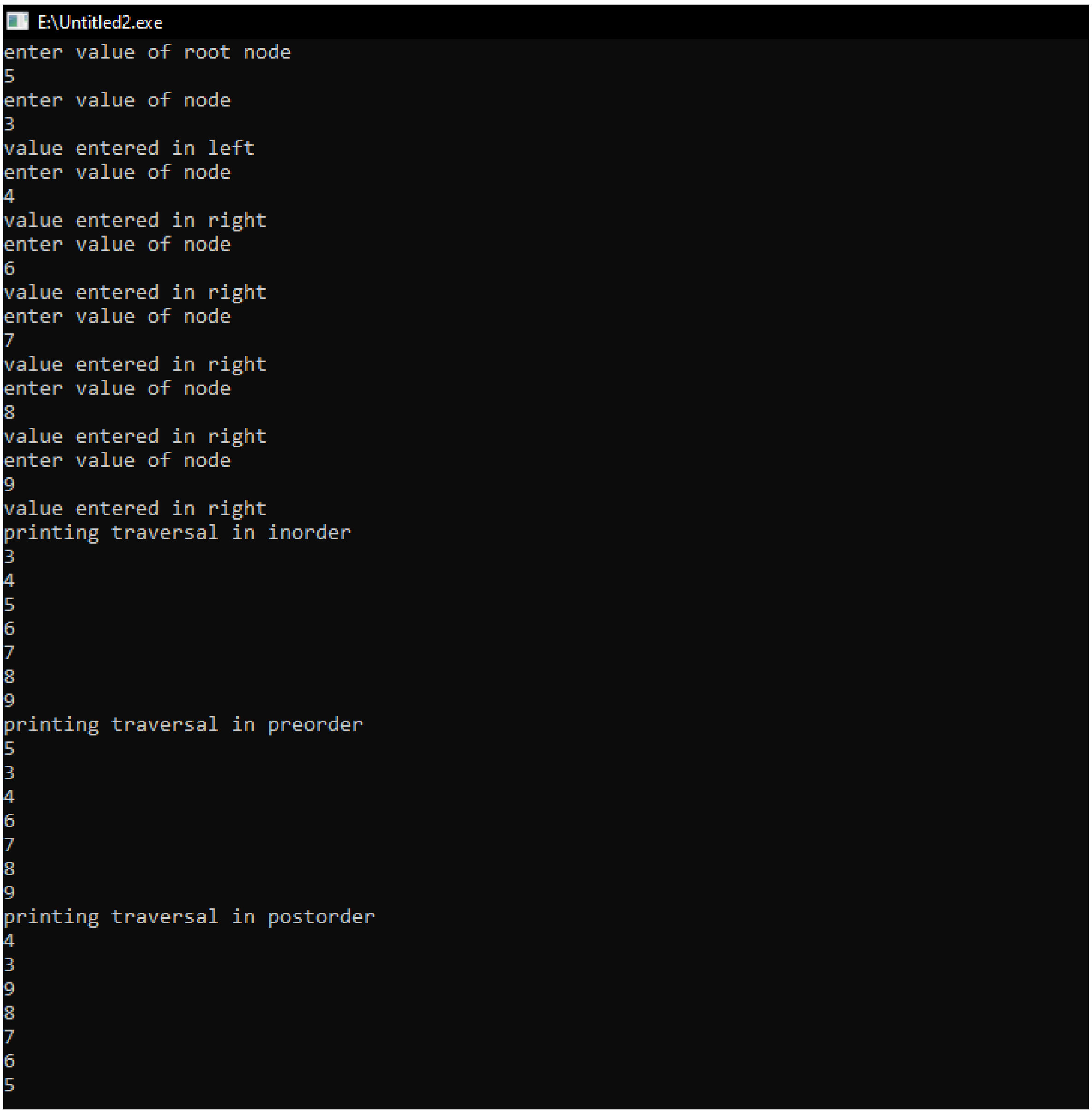
```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice :
1
Insert the element in queue :
5
Enter your choice :
1
Insert the element in queue :
4
Enter your choice :
1
Insert the element in queue :
3
Enter your choice :
1
Insert the element in queue :
2
Enter your choice :
1
Insert the element in queue :
3
Enter your choice :
3
Queue elements are : 5 4 3 2 3
Enter your choice :
2
Element deleted from queue is : 5
Enter your choice :
3
Queue elements are : 4 3 2 3
Enter your choice :
2
Element deleted from queue is : 4
Enter your choice :
3
Queue elements are : 3 2 3
```

# Program 4

## To Traverse a Binary Tree in Pre order , In order and  Post order manner

```
#include <iostream>
using namespace std;
#include <conio.h>
struct tree
{tree *l, *r;
 int data;
}*root = NULL, *p = NULL, *np = NULL, *q;
void create()
{int value,c = 0;
 while (c < 10)
    {if (root == NULL)
        {root = new tree;
         cout<<"enter value of root node\n";
         cin>>root->data;
         root->r=NULL;
         root->l=NULL;  }
        else
        {p = root;
         cout<<"enter value of node\n";
         cin>>value;
            while(true)
            {if (value < p->data)
                {if (p->l == NULL)
                    {   p->l = new tree;
                        p = p->l;
                        p->data = value;
                        p->l = NULL;
                        p->r = NULL;

                        cout<<"value entered in left\n";

                        break;}

                    else if (p->l != NULL)

                    p = p->l;       }

                else if (value > p->data)

                {if (p->r == NULL)

                    { p->r = new tree;

                        p = p->r;

                        p->data = value;

                        p->l = NULL;

                        p->r = NULL;

                        cout<<"value entered in right\n";

                        break;}

                    else if (p->r != NULL)

                    p = p->r

                  }}}

                c++;}}

void inorder(tree *p)

{if (p != NULL)

    {   inorder(p->l);

        cout<<p->data<<endl;

        inorder(p->r)    }}

void preorder(tree *p)

{if (p != NULL)

    {   cout<<p->data<<endl;

        preorder(p->l);

        preorder(p->r);}}

void postorder(tree *p)
```

```
{if (p != NULL)
   {  postorder(p->l);
      postorder(p->r);
      cout<<p->data<<endl;}}
int main()
{   create();
   cout<<"printing traversal in inorder\n";
   inorder(root);
   cout<<"printing traversal in preorder\n";
   preorder(root);
   cout<<"printing traversal in postorder\n";
   postorder(root);
   getch();}
```

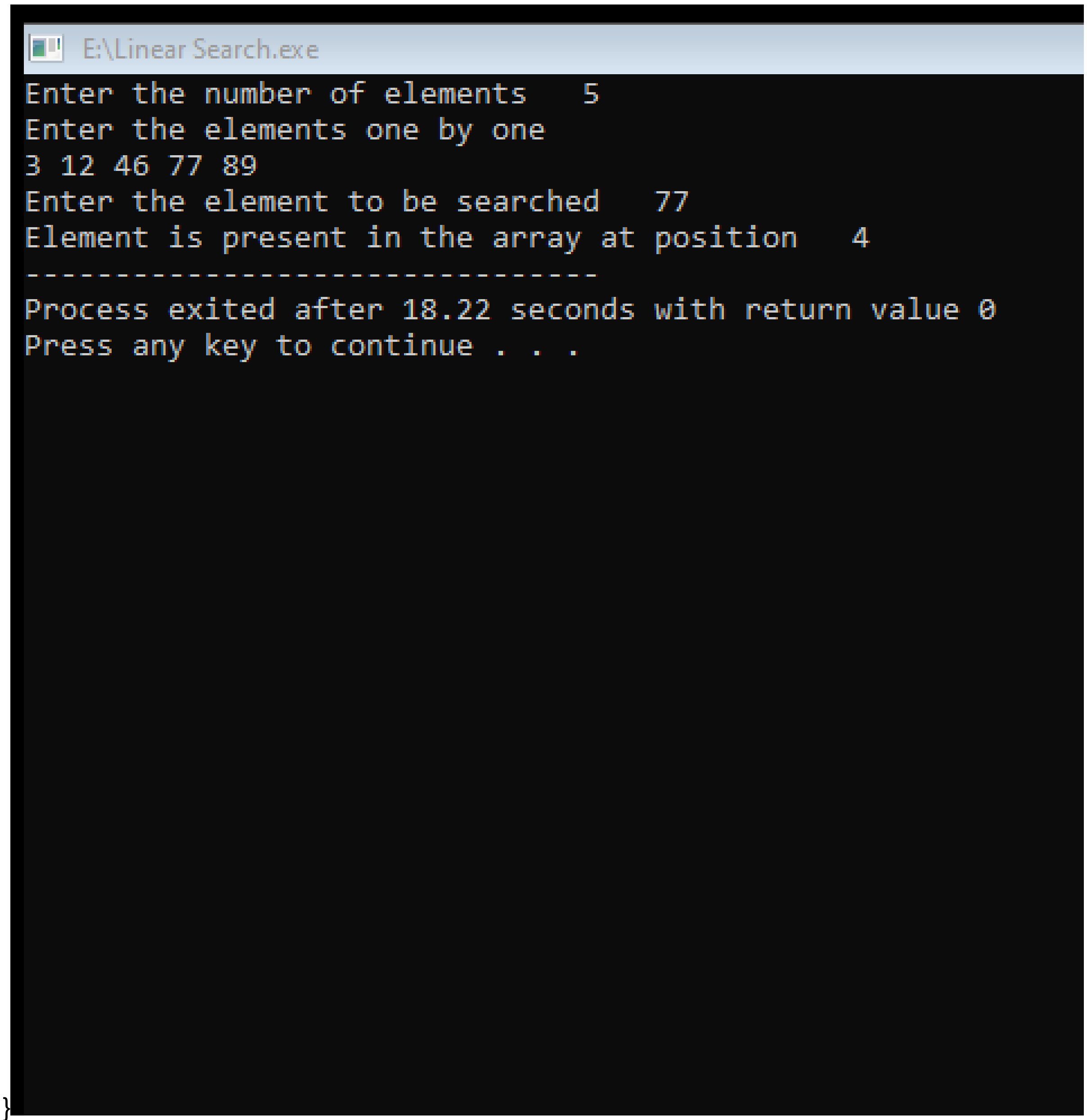

E:\Untitled2.exe

```
enter value of root node
5
enter value of node
3
value entered in left
enter value of node
4
value entered in right
enter value of node
6
value entered in right
enter value of node
7
value entered in right
enter value of node
8
value entered in right
enter value of node
9
value entered in right
printing traversal in inorder
3
4
5
6
7
8
9
printing traversal in preorder
5
3
4
6
7
8
9
printing traversal in postorder
4
3
9
8
7
6
5
```

## Program 5(a)

## To Implement Linear Search

```
#include <iostream>
using namespace std;
class LS
{public:
 void LinearSearch(int arr[], int value, int i, int n)
 {int found = 0;
 for (i = 0; i < n ; i++)
 {if (value == arr[i] )
  {found = 1;
    break;    }   }
    if (found == 1)
    cout<<"Element is present in the array at position   "<<i+1;
    else
    cout<<"Element is not present in the array.";    }};
int  main()
{  int num;
    int i,  keynum, found = 0;
    cout<<"Enter the number of elements   ";
    cin>>num;
    int array[num];
    cout<<"Enter the elements one by one \n";
    for (i = 0; i < num; i++)
    cin>> array[i];
    cout<<"Enter the element to be searched   ";
    cin>>keynum; LS l1;
    l1.LinearSearch(array,keynum,i,num);
    return 0;
```


E:\Linear Search.exe

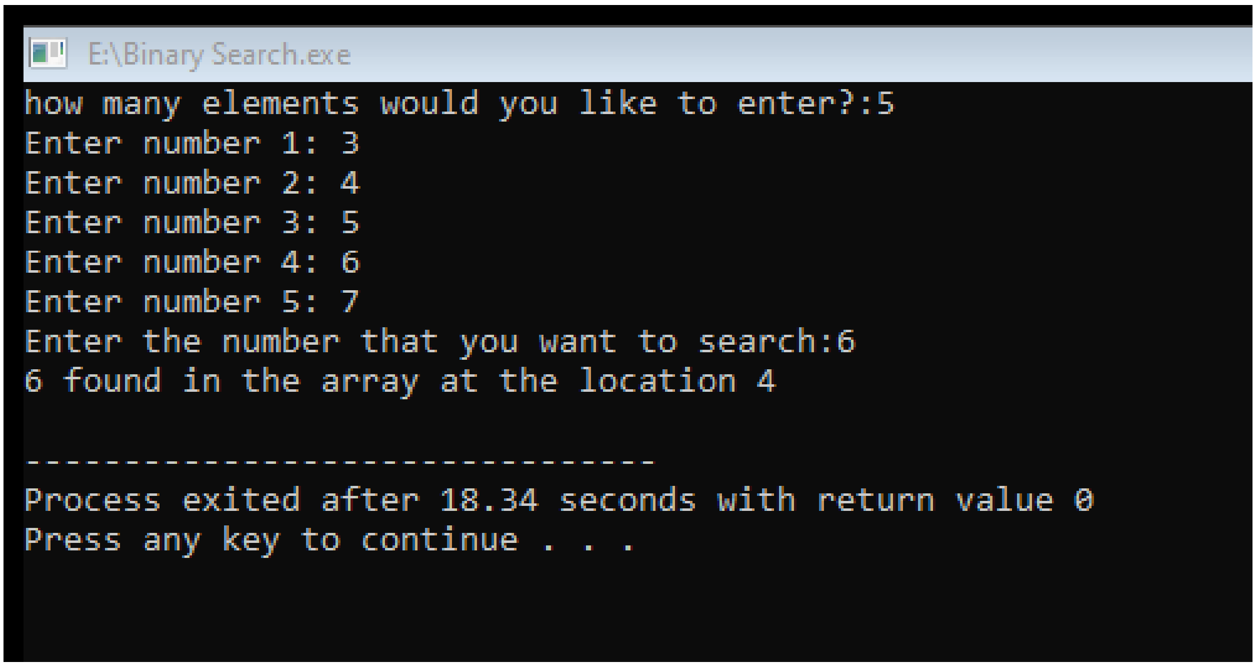
```
Enter the number of elements    5
Enter the elements one by one
3 12 46 77 89
Enter the element to be searched    77
Element is present in the array at position    4
--------------------------------
Process exited after 18.22 seconds with return value 0
Press any key to continue . . .
```

}

## Program 5(b)

### To Implement Binary Search

```
#include <iostream>
using namespace std;
int main()
{int count, i, arr[30], num, first, last, middle;
 cout<<"how many elements would you like to enter?:";
 cin>>count;
 for (i=0; i<count; i++)
 {cout<<"Enter number "<<(i+1)<<": ";
  cin>>arr[i];}
    cout<<"Enter the number that you want to search:";
    cin>>num;
            first = 0;
            last = count-1;
            middle = (first+last)/2;
            while (first <= last)
            {if(arr[middle] < num)
              first = middle + 1;
               else if(arr[middle] == num)
               { cout<<num<<" found in the array at the location "<<middle+1<<"\n";
        break;  }
           else
          last = middle - 1;
           middle = (first + last)/2; }
        if(first > last)
                cout<<num<<" not found in the array";
            return 0;}
```

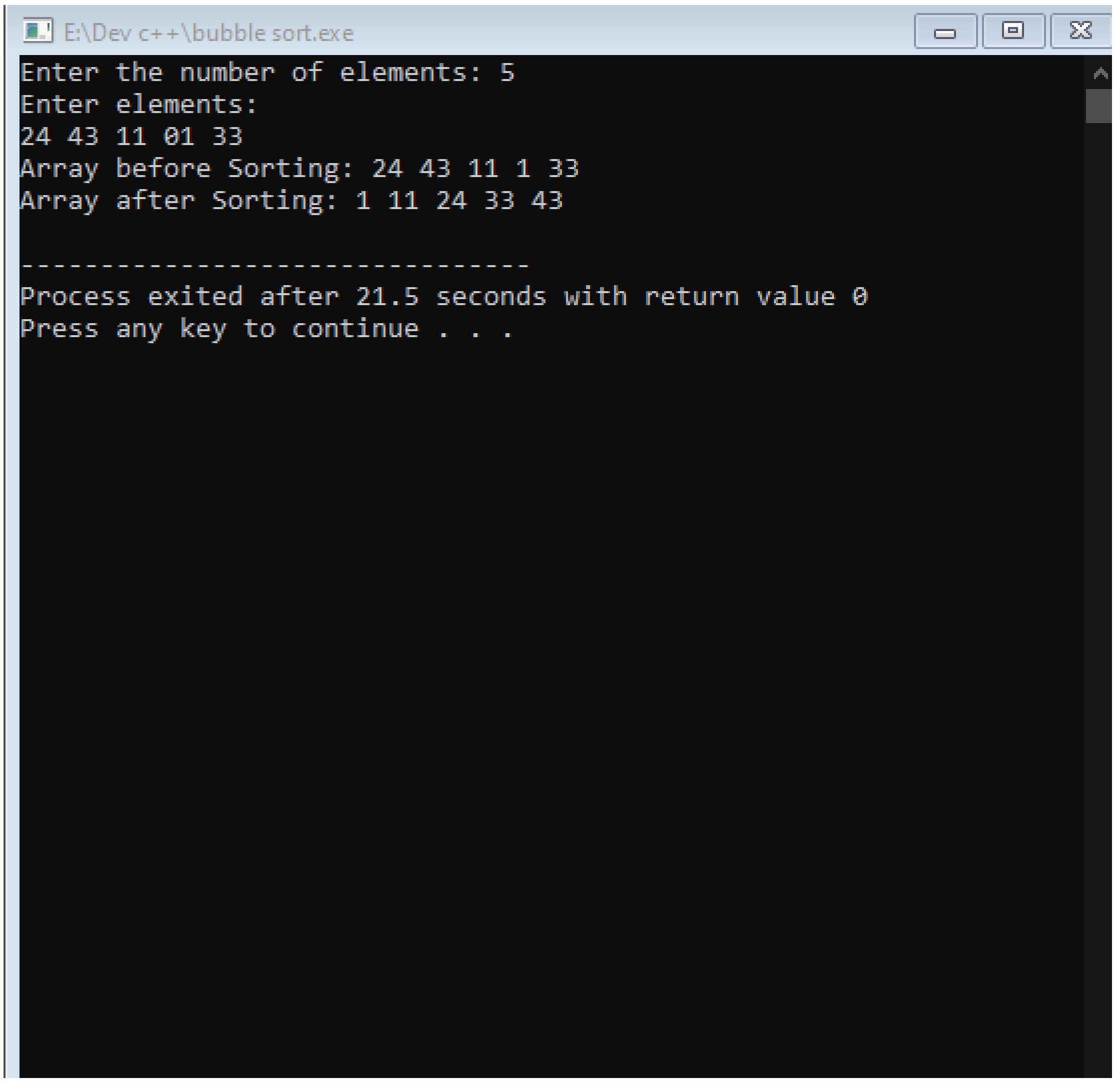


```
how many elements would you like to enter?:5
Enter number 1: 3
Enter number 2: 4
Enter number 3: 5
Enter number 4: 6
Enter number 5: 7
Enter the number that you want to search:6
6 found in the array at the location 4

--------------------------------
Process exited after 18.34 seconds with return value 0
Press any key to continue . . .
```

## Program 6 (a)

## To implement Bubble Sort

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b) {
  int temp;
  temp = a;
  a = b;
  b = temp;}
void display(int *array, int size) {
  for(int i = 0; i<size; i++)
    cout << array[i] << " ";
  cout << endl;}
void bubbleSort(int *array, int size) {
  for(int i = 0; i<size; i++) {
    int swaps = 0;
    for(int j = 0; j<size-i-1; j++) {
      if(array[j] > array[j+1]) {
        swapping(array[j], array[j+1]);
        swaps = 1;        }
    if(!swaps)
      break  }}
int main() {
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  int arr[n];
  cout << "Enter elements:" << endl;
  for(int i = 0; i<n; i++) {
   cin >> arr[i];}
cout << "Array before Sorting: ";
display(arr, n);
bubbleSort(arr, n);
cout << "Array after Sorting: ";
display(arr, n);}
```

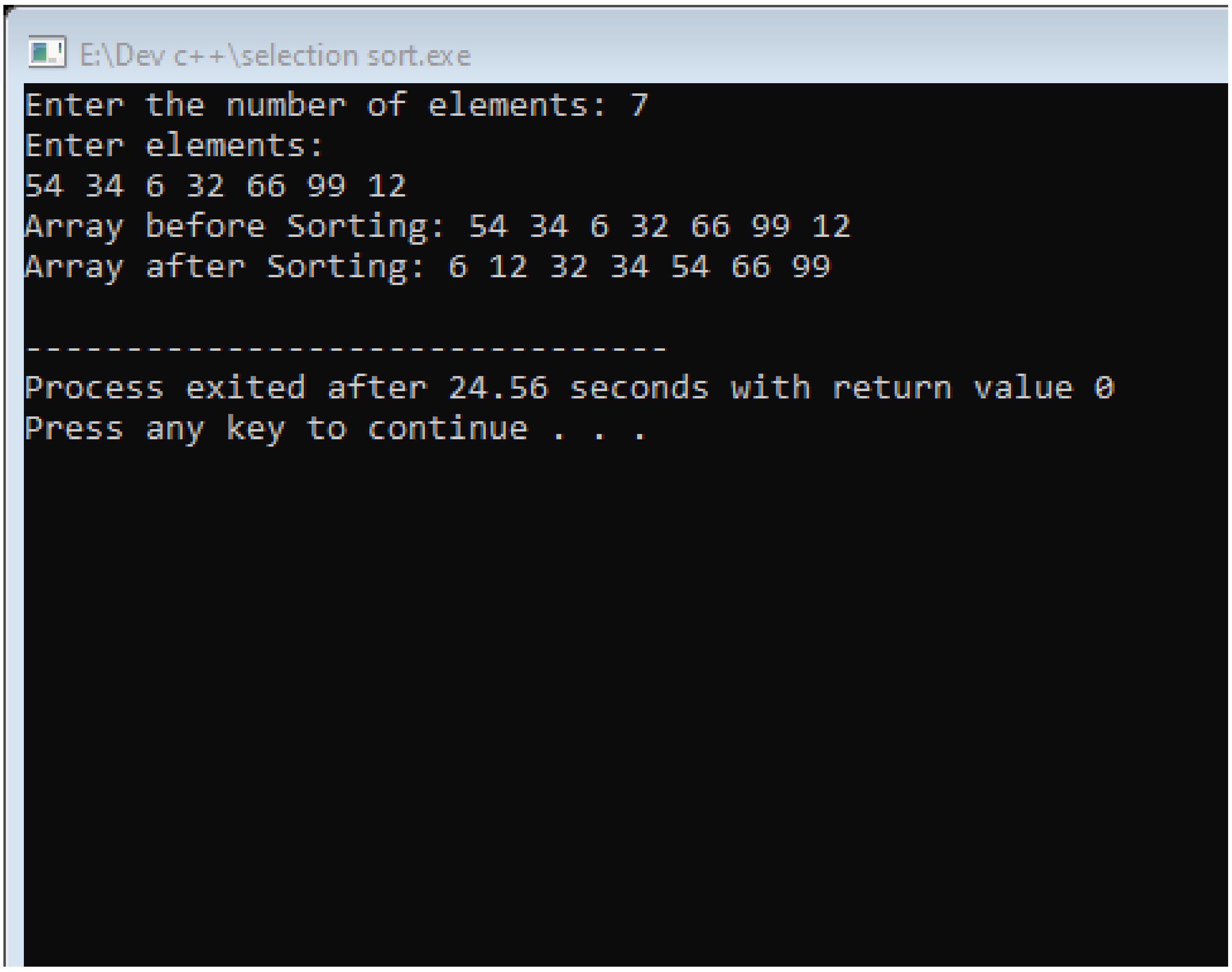


```
Enter the number of elements: 5
Enter elements:
24 43 11 01 33
Array before Sorting: 24 43 11 1 33
Array after Sorting: 1 11 24 33 43

--------------------------------
Process exited after 21.5 seconds with return value 0
Press any key to continue . . .
```

## Program 6(b)

## To implement Selection Sort

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b) {
   int temp;
   temp = a;
   a = b;
   b = temp;}
void display(int *array, int size) {
   for(int i = 0; i<size; i++)
      cout << array[i] << " ";
   cout << endl;}
void selectionSort(int *array, int size) {
   int i, j, imin;
   for(i = 0; i<size-1; i++) {
      imin = i;
      for(j = i+1; j<size; j++)
         if(array[j] < array[imin])
            imin = j;
         swap(array[i], array[imin]);}   }
int main() {
   int n;
   cout << "Enter the number of elements: ";
   cin >> n;
   int arr[n];
   cout << "Enter elements:" << endl;
   for(int i = 0; i<n; i++) {
      cin >> arr[i];  }
cout << "Array before Sorting: ";

display(arr, n);

selectionSort(arr, n);

cout << "Array after Sorting: ";

display(arr, n);}
```

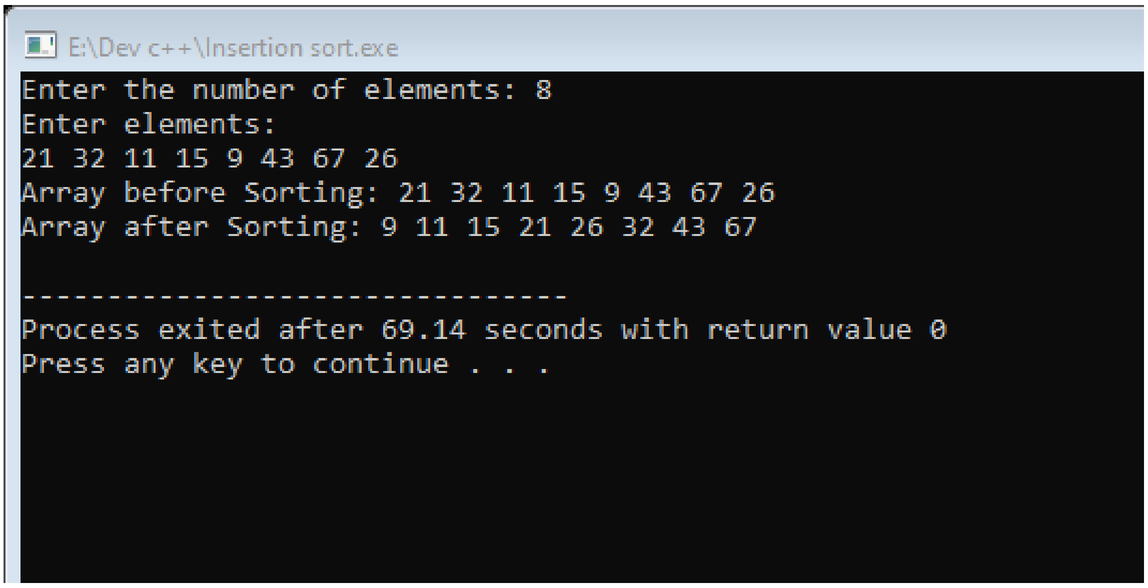


```
Enter the number of elements: 7
Enter elements:
54 34 6 32 66 99 12
Array before Sorting: 54 34 6 32 66 99 12
Array after Sorting: 6 12 32 34 54 66 99

--------------------------------
Process exited after 24.56 seconds with return value 0
Press any key to continue . . .
```

## Program 6(c)

## To Implement Insertion sort

```
#include<iostream>
using namespace std;
void display(int *array, int size) {
  for(int i = 0; i<size; i++)
    cout << array[i] << " ";
  cout << endl;
}
void insertionSort(int *array, int size) {
  int key, j;
  for(int i = 1; i<size; i++) {
    key = array[i];
    j = i;
    while(j > 0 && array[j-1]>key) {
      array[j] = array[j-1];
      j--;
    }
    array[j] = key;
  }
}
int main() {
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  int arr[n];
  cout << "Enter elements:" << endl;
  for(int i = 0; i<n; i++) {
    cin >> arr[i];
    }
    cout << "Array before Sorting: ";
    display(arr, n);
    insertionSort(arr, n);
    cout << "Array after Sorting: ";
    display(arr, n);
}
```

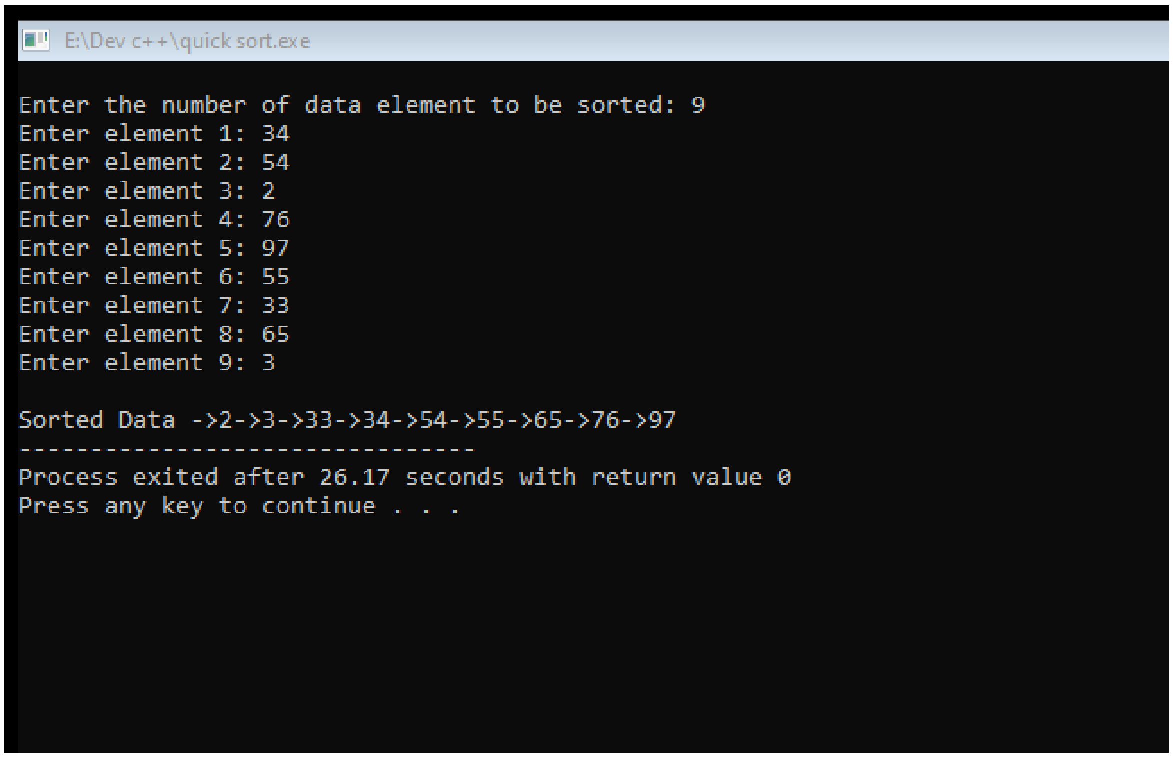


```
Enter the number of elements: 8
Enter elements:
21 32 11 15 9 43 67 26
Array before Sorting: 21 32 11 15 9 43 67 26
Array after Sorting: 9 11 15 21 26 32 43 67

--------------------------------
Process exited after 69.14 seconds with return value 0
Press any key to continue . . .
```

## Program 6(d)

## To Implement Quick Sort

```
#include<iostream>

#include<cstdlib>

using namespace std;

void swap(int *a, int *b) {

  int temp;

  temp = *a;

  *a = *b;

  *b = temp;}

int Partition(int a[], int l, int h) {

  int pivot, index, i;

  index = l;

  pivot = h;

  for(i = l; i < h; i++) {

    if(a[i] < a[pivot]) {

      swap(&a[i], &a[index]);

      index++;  }}

  swap(&a[pivot], &a[index]);

  return index;}

int RandomPivotPartition(int a[], int l, int h) {

  int pvt, n, temp;

  n = rand();

  pvt = l + n%(h-l+1);

  swap(&a[h], &a[pvt]);

  return Partition(a, l, h);}

int QuickSort(int a[], int l, int h) {

  int pindex;

  if(l < h) {
      pindex = RandomPivotPartition(a, l, h);
      QuickSort(a, l, pindex-1);
      QuickSort(a, pindex+1, h);}
   return 0;}
int main() {
   int n, i;
   cout<<"\nEnter the number of data element to be sorted: ";
   cin>>n;
   int arr[n];
   for(i = 0; i < n; i++) {
      cout<<"Enter element "<<i+1<<": ";
      cin>>arr[i];}
   QuickSort(arr, 0, n-1);
   cout<<"\nSorted Data ";
   for (i = 0; i < n; i++)
      cout<<"->"<<arr[i];
   return 0;
}
```



E:\Dev c++\quick sort.exe

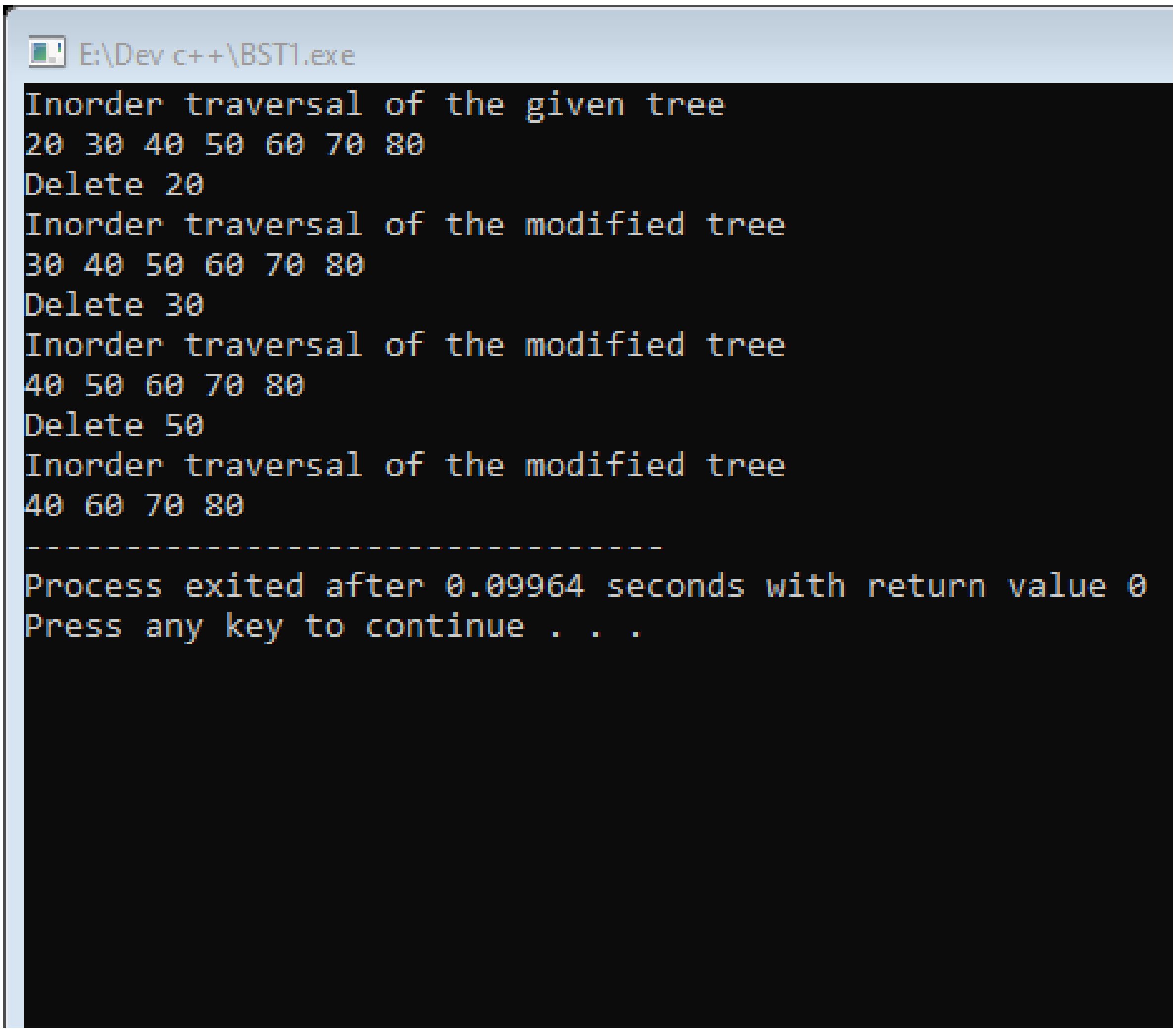
```
Enter the number of data element to be sorted: 9
Enter element 1: 34
Enter element 2: 54
Enter element 3: 2
Enter element 4: 76
Enter element 5: 97
Enter element 6: 55
Enter element 7: 33
Enter element 8: 65
Enter element 9: 3

Sorted Data ->2->3->33->34->54->55->65->76->97
--------------------------------
Process exited after 26.17 seconds with return value 0
Press any key to continue . . .
```

# Program 7

## To Insert and Delete elements in a Binary Search Tree

```
#include<bits/stdc++.h>
using namespace std;
struct node
{  int key;
   struct node *left, *right; };
// A utility function to create a new BST node
struct node *newNode(int item)
{   struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp; }
// A utility function to do inorder traversal of BST
void inorder(struct node *root)
{   if (root != NULL)
    { inorder(root->left);
        cout<<  root->key;
        inorder(root->right);  } }
/* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key)
{ /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);
    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left = insert(node->left, key);
    else
        node->right = insert(node->right, key);
/* return the (unchanged) node pointer */
```

```
    return node;
}
/* Given a non-empty binary search tree, return the node with minimum
key value found in that tree. Note that the entire tree does not
need to be searched. */
struct node * minValueNode(struct node* node)
{    struct node* current = node;
  /* loop down to find the leftmost leaf */
    while (current && current->left != NULL)
 current = current->left;
    return current; }
/* Given a binary search tree and a key, this function deletes the key
and returns the new root */
struct node* deleteNode(struct node* root, int key)
{  // base case
    if (root == NULL) return root;
    // If the key to be deleted is smaller than the root's key,
    // then it lies in left subtree
    if (key < root->key)
        root->left = deleteNode(root->left, key);
  // If the key to be deleted is greater than the root's key,
    // then it lies in right subtree
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    // if key is same as root's key, then This is the node
    // to be deleted
    else
    {  // node with only one child or no child
        if (root->left == NULL)
        {     struct node *temp = root->right;
           free(root);
           return temp;  }
        else if (root->right == NULL)
        {      struct node *temp = root->left;
           free(root);
           return temp;     }
        // node with two children: Get the inorder successor (smallest
        // in the right subtree)
        struct node* temp = minValueNode(root->right);
        // Copy the inorder successor's content to this node
        root->key = temp->key;
        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->key);  }
    return root; }
  // Driver Program to test above functions
int main()
{  /* Let us create following BST
         50
        /    \
        30    70
        / \ / \
    20 40 60 80 */
    struct node *root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
    cout << "Inorder traversal of the given tree \n";
    inorder(root);
    cout<<"\nDelete 20\n";
    root = deleteNode(root, 20);
    cout<<"Inorder traversal of the modified tree \n";
    inorder(root);
    cout<<"\nDelete 30\n";
    root = deleteNode(root, 30);
    cout<<"Inorder traversal of the modified tree \n";
    inorder(root);
    cout<<"\nDelete 50\n";
    root = deleteNode(root, 50);
    cout<<"Inorder traversal of the modified tree \n";
    inorder(root);
    return 0;
}
```



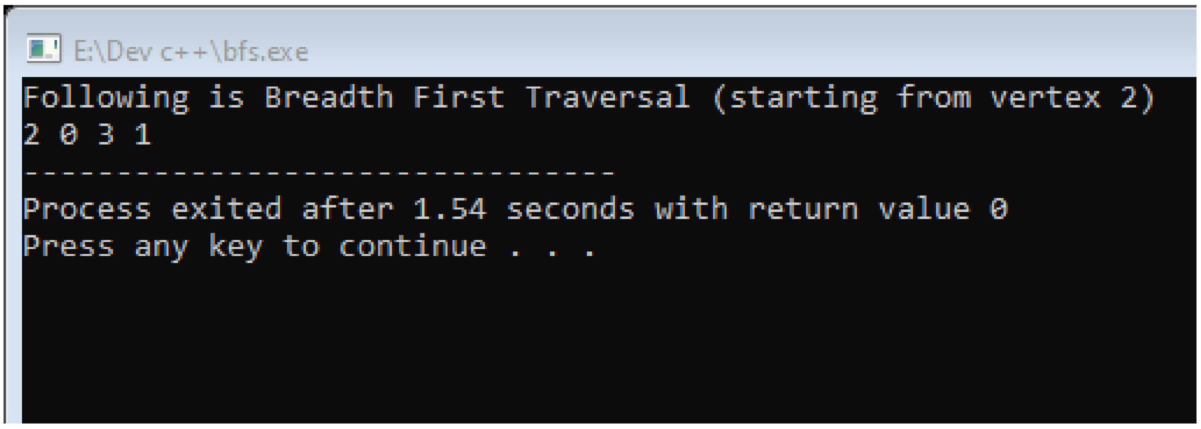
E:\Dev c++\BST1.exe

```
Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
Delete 50
Inorder traversal of the modified tree
40 60 70 80
--------------------------------
Process exited after 0.09964 seconds with return value 0
Press any key to continue . . .
```

## Program 8(a)

## To implement Depth First Traversal

```
#include<iostream>
#include <list>
using namespace std;
// This class represents a directed graph using
// adjacency list representation
class Graph
{   int V;    // No. of vertices
    // Pointer to an array containing adjacency
    // lists
    list<int> *adj;
public:
    Graph(int V);  // Constructor
    // function to add an edge to graph
    void addEdge(int v, int w);
      // prints BFS traversal from a given source s
    void BFS(int s);   };
Graph::Graph(int V)
{     this->V = V;
    adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v’s list.
}
void Graph::BFS(int s)
{   // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;
    // Create a queue for BFS
    list<int> queue;
    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);
    // 'i' will be used to get all adjacent
    // vertices of a vertex
    list<int>::iterator i;
    while(!queue.empty())
    { // Dequeue a vertex from queue and print it
        s = queue.front();
        cout << s << " ";
        queue.pop_front();
        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i); }  } } }
// Driver program to test methods of graph class
int main()
{   // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
    cout << "Following is Breadth First Traversal "
        << "(starting from vertex 2) \n";
    g.BFS(2);
    return 0;
}
```



```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
--------------------------------
Process exited after 1.54 seconds with return value 0
Press any key to continue . . .
```

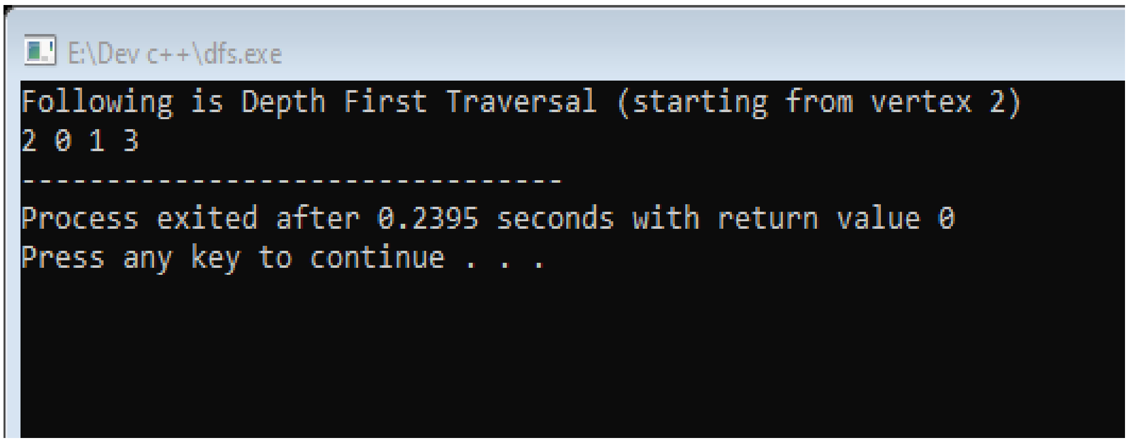
## Program 8(b)

## To implement Depth First Traversal

```
#include<bits/stdc++.h>
using namespace std;
// Graph class represents a directed graph
// using adjacency list representation
class Graph
{           int V; // No. of vertices
            // Pointer to an array containing
            // adjacency lists
            list<int> *adj;
            // A recursive function used by DFS
            void DFSUtil(int v, bool visited[]);
public:
            Graph(int V); // Constructor
            // function to add an edge to graph
            void addEdge(int v, int w);
            // DFS traversal of the vertices
            // reachable from v
            void DFS(int v);
};
Graph::Graph(int V)
{           this->V = V;
            adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{           adj[v].push_back(w); // Add w to v’s list.
}
```

```
void Graph::DFSUtil(int v, bool visited[])
{		// Mark the current node as visited and
		// print it
		visited[v] = true;
		cout << v << " ";

		// Recur for all the vertices adjacent
		// to this vertex
		list<int>::iterator i;
		for (i = adj[v].begin(); i != adj[v].end(); ++i)
		  if (!visited[*i])
			DFSUtil(*i, visited);
}

// DFS traversal of the vertices reachable from v.
// It uses recursive DFSUtil()
void Graph::DFS(int v)
{
		// Mark all the vertices as not visited
		bool *visited = new bool[V];
		for (int i = 0; i < V; i++)
		  visited[i] = false;

		// Call the recursive helper function
		// to print DFS traversal
		DFSUtil(v, visited);
}

// Driver code
```

```
int main()
{
		// Create a graph given in the above diagram
		Graph g(4);
		g.addEdge(0, 1);
		g.addEdge(0, 2);
		g.addEdge(1, 2);
		g.addEdge(2, 0);
		g.addEdge(2, 3);
		g.addEdge(3, 3);

		cout << "Following is Depth First Traversal"
			" (starting from vertex 2) \n";
		g.DFS(2);

		return 0;
}
```



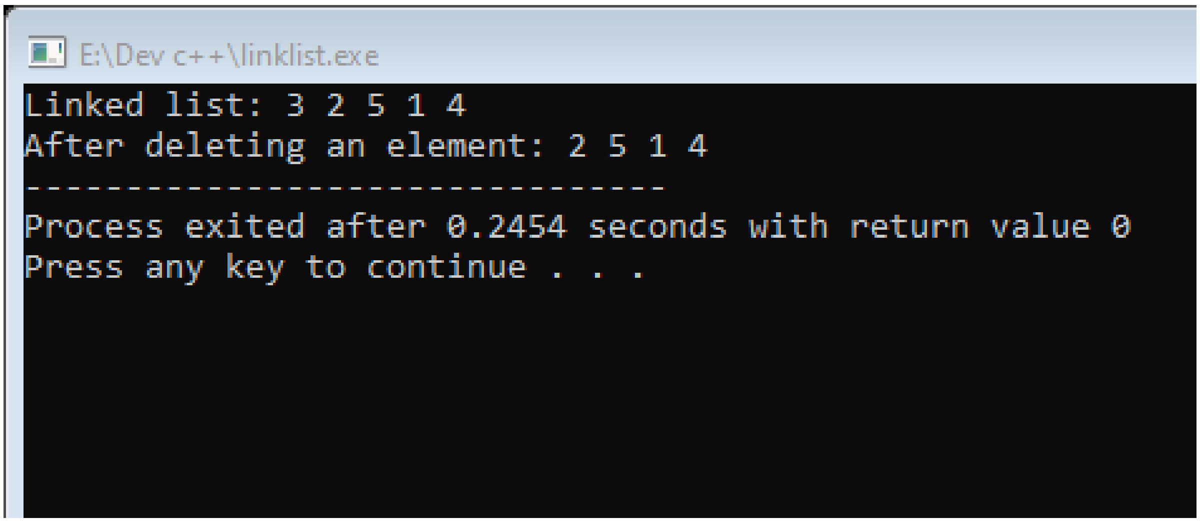
E:\Dev c++\dfs.exe

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
--------------------------------
Process exited after 0.2395 seconds with return value 0
Press any key to continue . . .
```

## Program 9

## To Create a Link List , Insert and Delete elements in Link List

```
// Linked list operations in C++
#include <stdlib.h>
#include <iostream>
using namespace std;
// Create a node
struct Node {
  int item;
  struct Node* next;
};
void insertAtBeginning(struct Node** ref, int data) {
  // Allocate memory to a node
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  // insert the item
  new_node->item = data;
  new_node->next = (*ref);
  // Move head to new node
  (*ref) = new_node;
}
// Insert a node after a node
void insertAfter(struct Node* prev_node, int data) {
  if (prev_node == NULL) {
    cout << "the given previous node cannot be NULL";
    return;
  }
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node->item = data;
  new_node->next = prev_node->next;
  prev_node->next = new_node;
}
void insertAtEnd(struct Node** ref, int data) {
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  struct Node* last = *ref;
 new_node->item = data;
  new_node->next = NULL;
  if (*ref == NULL) {
    *ref = new_node;
    return;
  }
  while (last->next != NULL)
    last = last->next;
  last->next = new_node;
  return;
}
void deleteNode(struct Node** ref, int key) {
  struct Node *temp = *ref, *prev;
  if (temp != NULL && temp->item == key) {
    *ref = temp->next;
    free(temp);
    return;
  }
  // Find the key to be deleted
  while (temp != NULL && temp->item != key) {
    prev = temp;
    temp = temp->next;
  }
  // If the key is not present
  if (temp == NULL) return;
  // Remove the node
  prev->next = temp->next
  free(temp);
}
// Print the linked list
void printList(struct Node* node) {
  while (node != NULL) {
    cout << node->item << " ";
    node = node->next;
  }}
// Driver program
int main() {
  struct Node* head = NULL;
  insertAtEnd(&head, 1);
  insertAtBeginning(&head, 2);
  insertAtBeginning(&head, 3);
  insertAtEnd(&head, 4);
  insertAfter(head->next, 5);
  cout << "Linked list: ";
  printList(head);
  cout << "\nAfter deleting in element: ";
  deleteNode(&head, 3);
  printList(head);
}
```



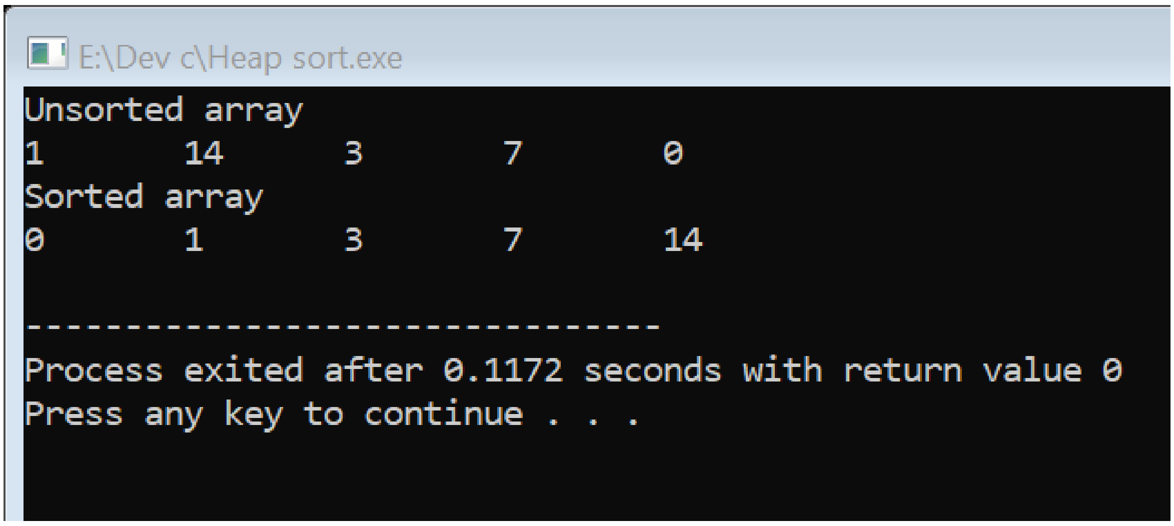
E:\Dev c++\linklist.exe

```
Linked list: 3 2 5 1 4
After deleting an element: 2 5 1 4
--------------------------------
Process exited after 0.2454 seconds with return value 0
Press any key to continue . . .
```

# Program 10

## To Implement Heap Sort

```
#include <iostream>
using namespace std;
void heapify(int arr[], int n, int i)
{   int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    //If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;
    //If right child largest
    if (r < n && arr[r] > arr[largest])
        largest = r;
    //If root is nor largest
    if (largest != i)
    {       swap(arr[i], arr[largest]);

        //Recursively heapifying the sub-tree
        heapify(arr, n, largest);
    }}
void heapSort(int arr[], int n)
{   for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    //One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        //Moving current root to end
        swap(arr[0], arr[i]);
        //Calling max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}
//Function to print array
void display(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << "\t";
    }
    cout << "\n";
}
int main()
{
    int arr[] = {1, 14, 3, 7, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Unsorted array  \n";
    display(arr, n);
    heapSort(arr, n);
    cout << "Sorted array  \n";
    display(arr, n);
}
```



```
Unsorted array
1       14      3       7       0
Sorted array
0       1       3       7       14

--------------------------------
Process exited after 0.1172 seconds with return value 0
Press any key to continue . . .
```