Name: - Atish Kumar

Roll No: - 120CS0173

Lab Sheet: - 05

Q1. Write a program to convert a sparse matrix from a 2D array into

A. An array representation of nx3 size, where n is the number of non-zero elements in the sparse matrix and each element is represented by row, column, and data.

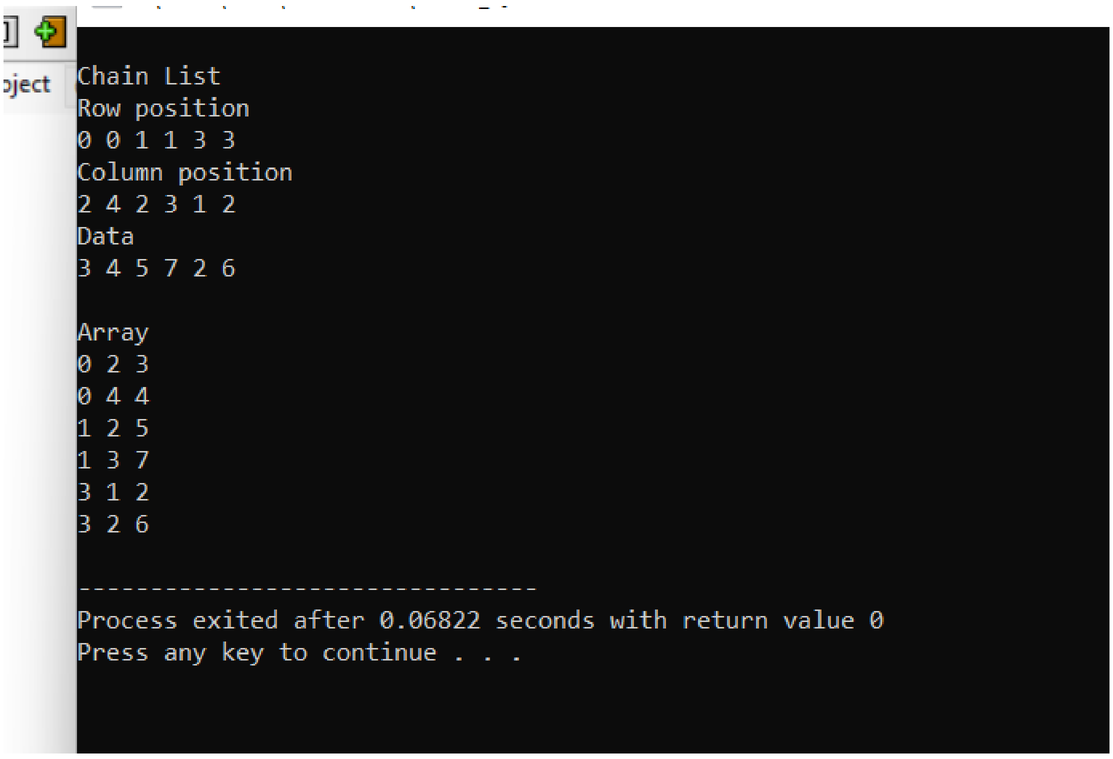
B. A chain list (Linked list representation of the sparse matrix where each non-zero element is represented by a node containing the data, row, column, and a pointer)

Program:-

```
#include <iostream>

using namespace std;


class Node{

   public:

   int data, row, col;

   Node *next;

   Node();

   Node(int, int, int);

};

Node::Node(){

   next = NULL;

}


Node::Node(int data, int row, int col){

   this->data = data;

   this->row = row;

   this->col = col;

   next = NULL;
```

```
}

class ChainList{
    static Node *head;
    int size;



    public:
    ChainList();
    void append(int, int, int);
    int len();
    void show();
};

Node* ChainList::head = NULL;
ChainList::ChainList(){
    size = 0;
}

void ChainList::append(int data, int row, int col){
    if(head == NULL){
        head = new Node(data, row, col);
        size++;
        return;
    }
    Node *curr = head;
    while(curr->next != NULL){
        curr = curr->next;
    }
    curr->next = new Node(data, row, col);
    size++;
}

int ChainList::len(){
    return size;
}

void ChainList::show(){
    Node* curr = head;
    cout<<"Row position\n";
    while(curr != NULL){
        cout<<curr->row<<" ";
        curr = curr->next;
    }
    curr = head;
    cout<<"\nColumn position\n";
    while(curr != NULL){
        cout<<curr->col<<" ";
        curr = curr->next;
    }
    curr = head;
    cout<<"\nData\n";
    while(curr != NULL){
        cout<<curr->data<<" ";
        curr = curr->next;
    }
    cout<<endl;
}

int main(){
    ChainList list;
    int size = 0, sparseMatrix[4][5] = {{0, 0, 3, 0, 4},
                                        {0, 0, 5, 7, 0},
                                        {0, 0, 0, 0, 0},
                                        {0, 2, 6, 0, 0}};
    for(int i = 0; i < 4; i++)
    for(int j = 0; j < 5; j++){
        if(sparseMatrix[i][j] != 0){
            list.append(sparseMatrix[i][j], i, j);
            size++;
        }
    }



    int (*array)[3] = new int[size][3];
    for(int i = 0, k = 0; i < 4; i++)
    for(int j = 0; j < 5; j++){
        if(sparseMatrix[i][j] != 0){
            array[k][0] = i;
            array[k][1] = j;
            array[k][2] = sparseMatrix[i][j];
            k++;
        }
    }
    cout<<"\nChain List\n";
    list.show();
    cout<<"\nArray\n";
    for(int i = 0; i < size; i++){
        cout<<array[i][0]<<" "<<array[i][1]<<" "<<array[i][2]<<endl;
    }



    return 0;
}
```

Output:-



```
Chain List
Row position
0 0 1 1 3 3
Column position
2 4 2 3 1 2
Data
3 4 5 7 2 6

Array
0 2 3
0 4 4
1 2 5
1 3 7
3 1 2
3 2 6

--------------------------------
Process exited after 0.06822 seconds with return value 0
Press any key to continue . . .
```

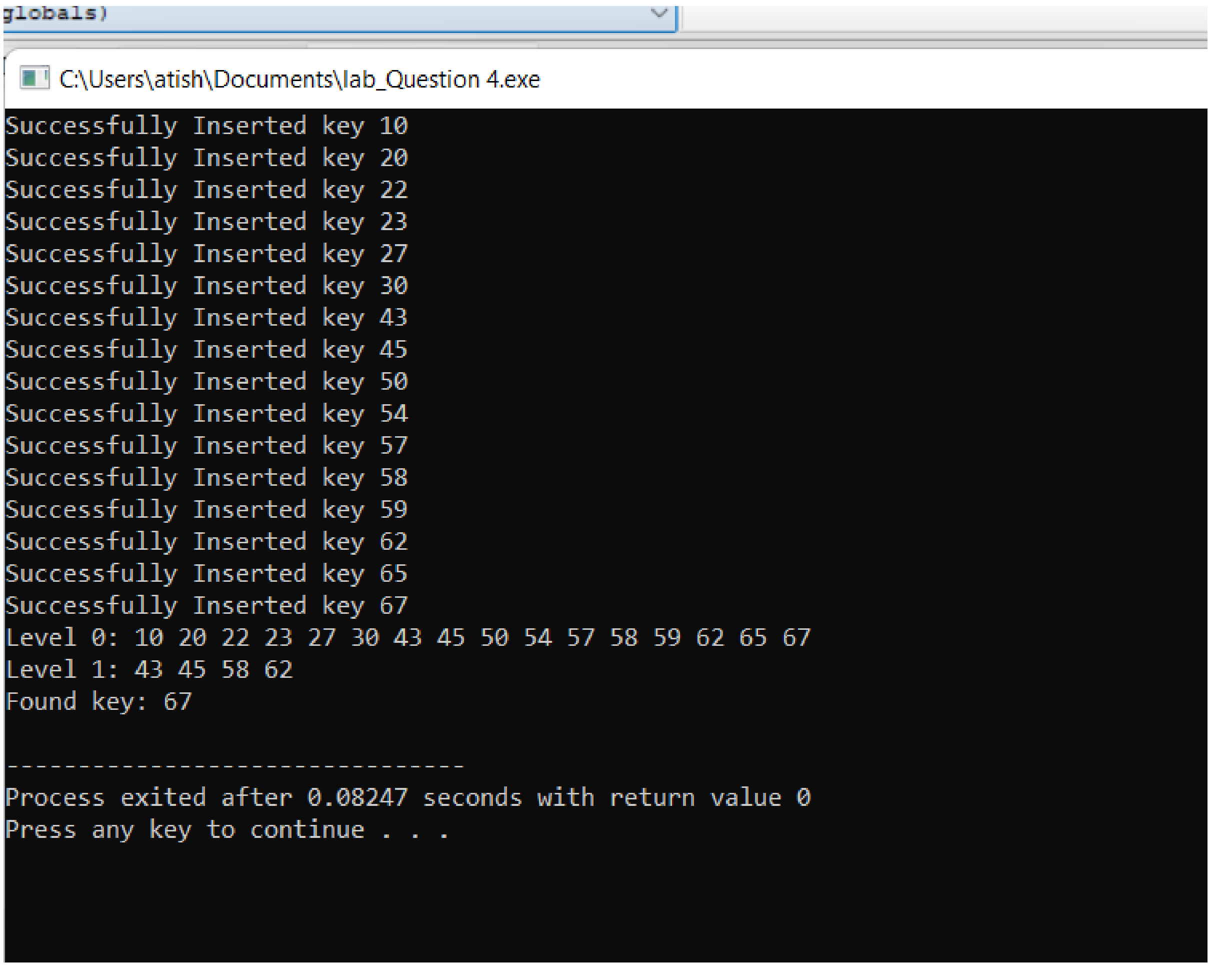
4.

Program:-

```
#include <bits/stdc++.h>

using namespace std;

class Node

{

 public:

 int k;

 Node **forw;

 Node(int, int);

};

Node::Node(int k, int level)

{

 this->k = k;
 forw = new Node*[level+1];
 memset(forw, 0, sizeof(Node*)*(level+1));
};
class SkipList
{

 int MAX;
 float P;
 int level;
 Node *header;
public:
 SkipList(int, float);
 int randomLevel();
 Node* createNode(int, int);
 void insertElement(int);
 void searchElement(int);
 void displayList();
};
SkipList::SkipList(int MAX, float P)
{
 this->MAX = MAX;
 this->P = P;
 level = 0;
 header = new Node(-1, MAX);
};
int SkipList::randomLevel()
```

```
{
 float r = (float)rand()/RAND_MAX;
 int lvl = 0;
 while(r < P && lvl < MAX)
 {
 lvl++;
 r = (float)rand()/RAND_MAX;
 }
 return lvl;
};
Node* SkipList::createNode(int k, int level)
{
 Node *n = new Node(k, level);
 return n;
};
void SkipList::insertElement(int k)
{
 Node *cur = header;
 Node *update[MAX+1];
 memset(update, 0, sizeof(Node*)*(MAX+1));
 for(int i = level; i >= 0; i--)
 {
 while(cur->forw[i] != NULL &&
 cur->forw[i]->k < k)
 cur = cur->forw[i];
 update[i] = cur;
 }
 cur = cur->forw[0];
 if (cur == NULL || cur->k != k)
 {
 int rlevel = randomLevel();
 if(rlevel > level)
 {
 for(int i=level+1;i<rlevel+1;i++)
 update[i] = header;
 level = rlevel;
 }
 Node* n = createNode(k, rlevel);
 for(int i=0;i<=rlevel;i++)
 {
 n->forw[i] = update[i]->forw[i];
 update[i]->forw[i] = n;
 }
 cout<<"Successfully Inserted key "<<k<<"\n";
 }
};
void SkipList::displayList()
{
 for(int i=0;i<=level;i++)
 {
 Node *node = header->forw[i];
 cout<<"Level "<<i<<": ";
 while(node != NULL)
 {
 cout<<node->k<<" ";
 node = node->forw[i];
 }
 cout<<"\n";
 }
};
void SkipList::searchElement(int k)
{
 Node *cur = header;
 for(int i = level; i >= 0; i--)
 {
 while(cur->forw[i] &&
 cur->forw[i]->k < k)
 cur = cur->forw[i];
 }
 cur = cur->forw[0];
 if(cur and cur->k == k)
 cout<<"Found key: "<<k<<"\n";
};
int main()
{
 srand((unsigned)time(0));
 SkipList l(1, 0.5);
 l.insertElement(10);
 l.insertElement(20);
 l.insertElement(22);
 l.insertElement(23);
 l.insertElement(27);
 l.insertElement(30);
 l.insertElement(43);
 l.insertElement(45);
 l.insertElement(50);
 l.insertElement(54);
 l.insertElement(57);
 l.insertElement(58);
 l.insertElement(59);
 l.insertElement(62);
 l.insertElement(65);
 l.insertElement(67);
 l.displayList();
 l.searchElement(67);
}
```

Output:-



```
Successfully Inserted key 10
Successfully Inserted key 20
Successfully Inserted key 22
Successfully Inserted key 23
Successfully Inserted key 27
Successfully Inserted key 30
Successfully Inserted key 43
Successfully Inserted key 45
Successfully Inserted key 50
Successfully Inserted key 54
Successfully Inserted key 57
Successfully Inserted key 58
Successfully Inserted key 59
Successfully Inserted key 62
Successfully Inserted key 65
Successfully Inserted key 67
Level 0: 10 20 22 23 27 30 43 45 50 54 57 58 59 62 65 67
Level 1: 43 45 58 62
Found key: 67

--------------------------------
Process exited after 0.08247 seconds with return value 0
Press any key to continue . . .
```