```cpp
#include<bits/stdc++.h>
#include<chrono>
// #include <sys/resource.h>
using namespace std;

unordered_set<string> visited;//duplicacy check and no. of expanded nodes count needed
globally

struct State//keep track of puzzle's state
{
    vector<vector<int>> board{4,vector<int> (4,0)};
    int emptyr,emptyc;//dont search for empty cell location directly use this
    State(){}//empty constructor required

    State(vector<vector<int>> b,int r,int c)
    {
        board=b;emptyr=r;emptyc=c;
    }
};

struct Node
{
    Node *parent;
    State state;//state of this node
    int cost=1;
    string path;// steps to reach here
    Node(State st, Node* n, string s)//constructor
    {
        state=st;cost=1;path=s;parent=n;
    }
};

int isGoal(State state)//return 0 if not goal else 1
{
    if(state.emptyc==3&&state.emptyr==3)//Check for 0 at last position ie. [3,3]
    {
        int c=1;
        for(int i=0;i<4;i++)
            for(int j=0;j<4;j++)
            {
                if(i==3&&j==3)return 1;
                if(c!=state.board[i][j])return 0;
                c++;
            }
            return 1;
    }
    else return 0;//not goal
}

string bfsSearch(Node *initialNode) // give solution directly either path or not
possible
{
    queue<Node*> q;//to keep track of nodes in the frontier
    q.push(initialNode);
    visited.insert(initialNode->path);
    int dir[5]={-1,0,1,0,-1};
    string label="URDL"; // move anticlockwise through 4 directions acc to dir array
    while(!q.empty())
    {
        Node *curr=q.front();q.pop();//take topmost state from queue
        visited.insert(curr->path);//this path will always bring us here, so marking it
visited
        if(!isGoal(curr->state))//if not goal state add all children to frontier queue
        {

            int cr=curr->state.emptyr,cc=curr->state.emptyc;//the current empty cell
location

            for(int i=0;i<4;i++)     //checking in all 4 directions
            {
                int nr=cr+dir[i],nc=cc+dir[i+1]; //new row and col values
                if(nr>=0&&nc>=0&&nr<4&&nc<4&&(visited.find(curr-
>path+label[i])==visited.end())) //Checking for repeated states and skipping them
                {
                    State ns=curr->state;
                    swap(ns.board[cr][cc],ns.board[nr][nc]);
                    string np=curr->path+label[i];//new path to reach this child node
                    ns.emptyr=nr;ns.emptyc=nc;
                    Node *newNode=new Node(ns,curr,np);
                    visited.insert(np);
                    q.push(newNode);

                }
            }
        }
        else return curr->path;
    }

    return "NO";
}


//**************************   To pass arguments from standard I/O
*******************************

// int main()
// {
//     ios_base::sync_with_stdio(false);
//     auto start = chrono::high_resolution_clock::now();

//     vector<vector<int>> initialArr(4,vector<int>(4,0));
//     int x,r,c;

//     for(int i=0;i<4;i++)
//     {
//         for(int j=0;j<4;j++)
//         {
//             cin>>x;if(!x)r=i,c=j;
//             initialArr[i][j]=x;
//         }
//     }


//     State initS(initialArr,r,c);
//     Node* initialNode=new Node(initS,nullptr,"");
//     string sol="At Goal";
//     if(!isGoal(initS)) sol=bfsSearch(initialNode);

//     auto end = chrono::high_resolution_clock::now();

//     if(sol=="No") cout<<"No solution possible.\n";
//     else
//     {
//         cout<<"Moves:"<<sol<<endl;
//         cout<<"Number of Nodes expanded:"<<visited.size()<<endl;


//         chrono::duration<double> tExec = end-start;
//         cout<<"Time Taken:"<<tExec.count()<<"sec"<<endl;

//         size_t
totSize=(visited.size()*sizeof(visited))+(sizeof(initialNode))+(16*sizeof(initialArr));
//approx size
//         // size_t memoryUsage = sizeof(State) + sizeof(Node) + sizeof(visited);
//         cout<<"Memory Taken:"<<totSize/1024.0<<"kb"<<endl;
//     }
//     return 0;
// }




//**************************   To pass arguments in command line
*******************************

int main(int argc, char *argv[])
{

    ios_base::sync_with_stdio(false);
    auto start = chrono::high_resolution_clock::now();


   // cout<<argc;
      if(argc!=17) {cout<<"Wrong Input size\n";return 1;}// 16 elements + the command
to invoke file at argv[0]

    vector<vector<int>> initialArr(4,vector<int>(4,0)); //array of zeroes
    int x,r,c;

    for(int i=1;i<=16;i++) //converting input array stream to 2d array of 4*4 size
    {
        stringstream(argv[i])>>x;//get the array int by int
        initialArr[(i-1)/4][(i-1)%4]=x;
        if(!x)r=(i-1)/4,c=(i-1)%4;//find empty cell location
    }
    State initS(initialArr,r,c);//initial state
    Node* initialNode=new Node(initS,nullptr,"");
    string sol=bfsSearch(initialNode);

    auto end = chrono::high_resolution_clock::now();

    if(sol=="No") cout<<"No solution possible.\n";
    else
    {
        cout<<"Moves:"<<sol<<endl;
        cout<<"Number of Nodes expanded:"<<visited.size()<<endl;
        //Estimated execution time
        chrono::duration<double> tExec = end-start;
        cout<<"Time Taken:"<<tExec.count()<<"sec"<<endl;
        //Estimated memory usage
        size_t
totSize=(visited.size()*sizeof(visited))+(sizeof(initialNode))+(16*sizeof(initialArr));
//approx size
        cout<<"Memory Taken:"<<totSize/1024.0<<"kb"<<endl;//sizeof returns sizes in
byte, so conversion to kb is required
    }
    return 0;

}
```