N QUEEN

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

using namespace std;
class Solution {
  public:
    bool isSafe1(int row, int col, vector < string > board, int n) {
      // check upper element
      int duprow = row;
      int dupcol = col;

      while (row >= 0 && col >= 0) {
        if (board[row][col] == 'Q')
          return false;
        row--;
        col--;
      }

      col = dupcol;
      row = duprow;
      while (col >= 0) {
        if (board[row][col] == 'Q')
          return false;
        col--;
      }

      row = duprow;
      col = dupcol;
      while (row < n && col >= 0) {
        if (board[row][col] == 'Q')
          return false;
        row++;
        col--;
      }
      return true;
    }

    void solve(int col, vector < string > & board, vector < vector < string >> & ans, int n) {
      if (col == n) {
        ans.push_back(board);
        return;
      }
      for (int row = 0; row < n; row++) {
        if (isSafe1(row, col, board, n)) {
          board[row][col] = 'Q';
          solve(col + 1, board, ans, n);
          board[row][col] = '.';
        }
      }
    }
    vector < vector < string >> solveNQueens(int n) {
      vector < vector < string >> ans;
      vector < string > board(n);
      string s(n, '.');
      for (int i = 0; i < n; i++) {
        board[i] = s;
      }
      solve(0, board, ans, n);
      return ans;
    }
};

int main() {
 int n;
 cin>>n;
  Solution obj;
  vector < vector < string >> ans = obj.solveNQueens(n);
  for (int i = 0; i < ans.size(); i++) {
    cout << "Arrangement " << i + 1 << "\n";
    for (int j = 0; j < ans[0].size(); j++) {
      cout << ans[i][j];
      cout << endl;
    }
    cout << endl;
  }
  return 0;
}
```

JOB SCHEDULING

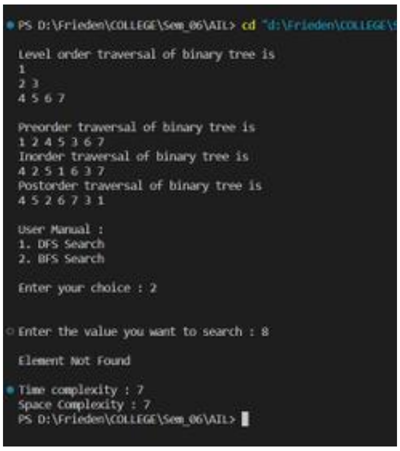
```
#include<bits/stdc++.h>

using namespace std;
// A structure to represent a job
struct Job {
   int id; // Job Id
   int dead; // Deadline of job
   int profit; // Profit if job is over before or on deadline
};
class Solution {
   public:
      bool static comparison(Job a, Job b) {
         return (a.profit > b.profit);
      }
   //Function to find the maximum profit and the number of jobs done
   pair < int, int > JobScheduling(Job arr[], int n) {

      sort(arr, arr + n, comparison);
      int maxi = arr[0].dead;
      for (int i = 1; i < n; i++) {
```

```
      maxi = max(maxi, arr[i].dead);
    }

    int slot[maxi + 1];

    for (int i = 0; i <= maxi; i++)
      slot[i] = -1;

    int countJobs = 0, jobProfit = 0;

    for (int i = 0; i < n; i++) {
      for (int j = arr[i].dead; j > 0; j--) {
        if (slot[j] == -1) {
          slot[j] = i;
          countJobs++;
          jobProfit += arr[i].profit;
          break;
        }
      }
    }

    return make_pair(countJobs, jobProfit);
  }
};
int main() {
  int n = 4;
  Job arr[n] = {{1,4,20},{2,1,10},{3,2,40},{4,2,30}};

  Solution ob;
  //function call
  pair < int, int > ans = ob.JobScheduling(arr, n);
  cout << ans.first << " " << ans.second << endl;

  return 0;
}
```

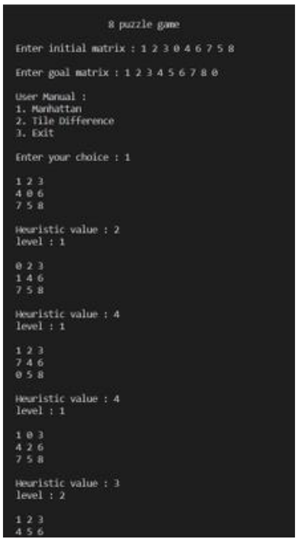
MANHATTEN AND TILE DIFFERENCE

```
#include<bits/stdc++.h>
using namespace std;

int checkheuristic(vector<vector<int>> matrix, vector<vector<int>>
goal_matrix){
    int count = 0;

    for (int i=0; i<3; i++){
        for (int j=0; j<3; j++){
            if (matrix[i][j] != goal_matrix[i][j]){
                count++;
            }
        }
    }

    return count;
}

int checkHeuristic(vector<vector<int>> matrix, vector<vector<int>>
goal_matrix){
    int count = 0;

    for (int i=0; i<3; i++){
        for (int j=0; j<3; j++){
            if (matrix[i][j] != 0){
                int x, y;
                int flag = 0;

                for (int m=0; m<3; m++){
                    for (int n=0; n<3; n++){
                        if (matrix[i][j] == goal_matrix[m][n]){
                            x = m;
                            y = n;
                            flag = 1;
                            break;
                        }
                    }
                    if (flag == 1){
                        break;
                    }
                }

                count = count + abs(i-x) + abs(j - y);
            }


        }
    }

    return count;

}

void display(vector<vector<int>> matrix, vector<vector<int>> goal_matrix, int
level, int type){
    cout<<"\n";
    for(int i=0; i<3; i++){
        for (int j=0; j<3; j++){
            cout<<matrix[i][j]<<" ";
        }
        cout<<endl;
    }

    int h;

    if (type == 1){
        h = checkHeuristic(matrix, goal_matrix);
    }
    else{
        h = checkheuristic(matrix, goal_matrix)-1;
    }

    cout<<"\nHeuristic value : "<<h;
    cout<<"\nlevel : "<<level<<"\n";
}

vector<vector<int>> transform_matrix(vector<vector<int>> matrix,
vector<vector<int>> goal_matrix, set<vector<vector<int>>> &matrix_set, int
level, int type){
    // coordinate of 0;
    int x, y;
    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            if (matrix[i][j] == 0){
                x = i;
                y = j;
            }
        }
    }

    // generation of matrix and checking if alredy exists in set;
    vector<vector<vector<int>>> matrix_collection;

    vector<vector<int>> matrix_a = matrix;
    vector<vector<int>> matrix_b = matrix;

    if (x == 1 and y == 1){
        vector<vector<int>> matrix_c = matrix;
        vector<vector<int>> matrix_d = matrix;

        swap(matrix_a[1][1], matrix_a[0][1]);
        swap(matrix_b[1][1], matrix_b[2][1]);
        swap(matrix_c[1][1], matrix_c[1][0]);
        swap(matrix_d[1][1], matrix_d[1][2]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
        matrix_collection.push_back(matrix_c);
        matrix_collection.push_back(matrix_d);
    }
    else if (x == 0 and y == 0){
        swap(matrix_a[0][0], matrix_a[0][1]);
        swap(matrix_b[0][0], matrix_b[1][0]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
    }
    else if (x == 0 and y == 1){
        vector<vector<int>> matrix_c = matrix;

        swap(matrix_a[0][1], matrix_a[1][1]);
        swap(matrix_b[0][1], matrix_b[0][2]);
        swap(matrix_c[0][1], matrix_c[0][0]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
        matrix_collection.push_back(matrix_c);

    }
    else if (x == 0 and y == 2){
        swap(matrix_a[0][2], matrix_a[1][2]);
        swap(matrix_b[0][2], matrix_b[0][1]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
    }
    else if (x == 1 and y == 0){
        vector<vector<int>> matrix_c = matrix;

        swap(matrix_a[1][0], matrix_a[1][1]);
        swap(matrix_b[1][0], matrix_b[0][0]);
        swap(matrix_c[1][0], matrix_c[2][0]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
        matrix_collection.push_back(matrix_c);
    }
    else if (x == 1 and y == 2){
        vector<vector<int>> matrix_c = matrix;

        swap(matrix_a[1][2], matrix_a[1][1]);
        swap(matrix_b[1][2], matrix_b[0][2]);
        swap(matrix_c[1][2], matrix_c[2][2]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
        matrix_collection.push_back(matrix_c);

    }
    else if (x == 2 and y == 0){
        swap(matrix_a[2][0], matrix_a[2][1]);
        swap(matrix_b[2][0], matrix_b[1][0]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
    }
    else if (x == 2 and y == 1){
        vector<vector<int>> matrix_c = matrix;

        swap(matrix_a[2][1], matrix_a[1][1]);
        swap(matrix_b[2][1], matrix_b[2][0]);
        swap(matrix_c[2][1], matrix_c[2][2]);

        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
        matrix_collection.push_back(matrix_c);
    }
    else if (x == 2 and y == 2){
        swap(matrix_a[2][2], matrix_a[2][1]);
        swap(matrix_b[2][2], matrix_b[1][2]);
        matrix_collection.push_back(matrix_a);
        matrix_collection.push_back(matrix_b);
    }

    // check heuristic of all matrix;
    int min = -1;
    int heu = INT_MAX;
    for (int i = 0; i <matrix_collection.size(); i++){
        auto pos = matrix_set.find(matrix_collection[i]);
        display(matrix_collection[i], goal_matrix, level, type);

        if (pos == matrix_set.end()){
            matrix_set.insert(matrix_collection[i]);
            int temp;

            if (type == 1){
                temp = checkHeuristic(matrix_collection[i], goal_matrix);
            }
            else{
                temp = checkheuristic(matrix_collection[i], goal_matrix);
            }

            if (temp < heu){
                min = i;
                heu = temp;
            }
        }
    }

    // return one with minimum heuristics
    return matrix_collection[min];
}

int main(){
    cout<<"\n\n\t\t 8 puzzle game";

    vector<vector<int>> matrix(3, vector<int>(3, 0));
    vector<vector<int>> goal_matrix(3, vector<int>(3, 0));

    set<vector<vector<int>>> matrix_set;
    matrix_set.insert(matrix);

    cout<<"\n\nEnter initial matrix : ";

    for(int i = 0; i <3; i++){
        for (int j = 0; j <3; j++){
            cin>>matrix[i][j];
        }
    }

    cout<<"\nEnter goal matrix : ";

    for(int i = 0; i <3; i++){
        for (int j = 0; j <3; j++){
            cin>>goal_matrix[i][j];
        }
    }

    int choice;
    cout<<"\nUser Manual : \n1. Manhattan\n2. Tile Difference\n3. Exit";
    cout<<"\n\nEnter your choice : ";
    cin>>choice;
    int loop = 0;

    if (choice == 1){
        int heuristic = checkHeuristic(matrix, goal_matrix);
        int level = 0;

        while (heuristic != 0){
            matrix = transform_matrix(matrix, goal_matrix, matrix_set, 
level+1, 1);
            heuristic = checkHeuristic(matrix, goal_matrix);
            level++;
        }
    }
    else if (choice == 2){
        int heuristic = checkheuristic(matrix, goal_matrix);
        int level = 0;

        while (heuristic != 0){
            matrix = transform_matrix(matrix, goal_matrix, matrix_set, 
level+1, 2);
            heuristic = checkheuristic(matrix, goal_matrix);
            level++;
        }
    }
    else if (choice == 3){
        cout<<"\nTerminated Successfully !!";
    }
    else{
        cout<<"\nInvalid choice so terminated !!";
    }

    return 0;
}


/* g++ file_name.cpp
   ./a.out
*/
```

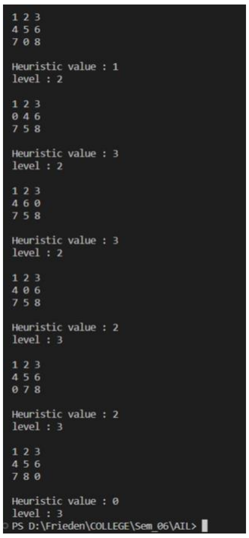
OUTPUT



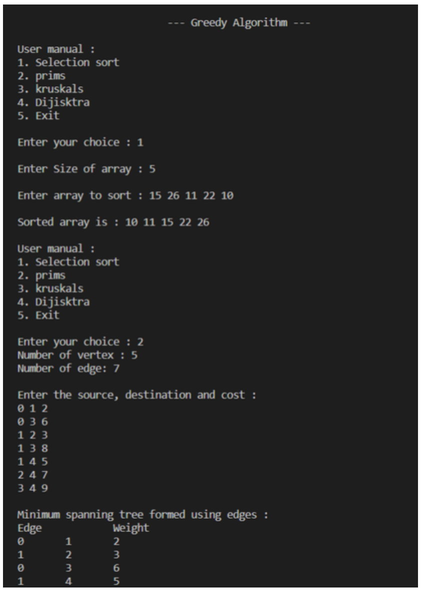
```
PS D:\Frieden\COLLEGE\Sem_06\AIL> cd "d:\Frieden\COLLEGE\S

Level order traversal of binary tree is
1
2 3
4 5 6 7

Preorder traversal of binary tree is
1 2 4 5 3 6 7
Inorder traversal of binary tree is
4 2 5 1 6 3 7
Postorder traversal of binary tree is
4 5 2 6 7 3 1

User Manual :
1. DFS Search
2. BFS Search

Enter your choice : 2


Enter the value you want to search : 8

Element Not Found

Time complexity : 7
Space Complexity : 7
PS D:\Frieden\COLLEGE\Sem_06\AIL>
```



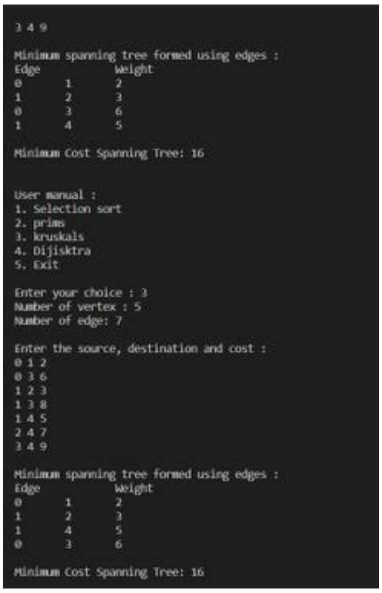
```
                8 puzzle game

Enter initial matrix : 1 2 3 0 4 6 7 5 8

Enter goal matrix : 1 2 3 4 5 6 7 8 0

User Manual :
1. Manhattan
2. Tile Difference
3. Exit

Enter your choice : 1

1 2 3
4 0 6
7 5 8

Heuristic value : 2
level : 1

0 2 3
1 4 6
7 5 8

Heuristic value : 4
level : 1

1 2 3
7 4 6
0 5 8

Heuristic value : 4
level : 1

1 0 3
4 2 6
7 5 8

Heuristic value : 3
level : 2

1 2 3
4 5 6
```



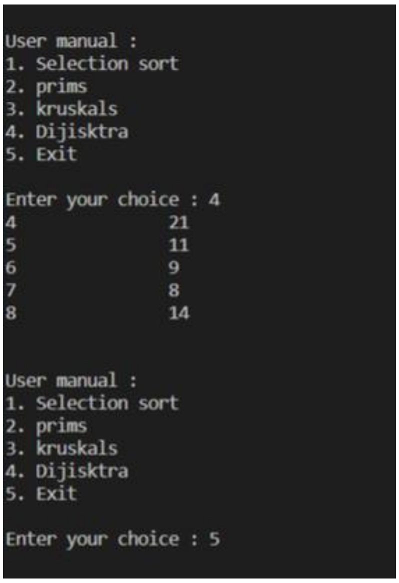
```
1 2 3
4 5 6
7 0 8

Heuristic value : 1
level : 2

1 2 3
0 4 6
7 5 8

Heuristic value : 3
level : 2

1 2 3
4 6 0
7 5 8

Heuristic value : 3
level : 2

1 2 3
4 0 6
7 5 8

Heuristic value : 2
level : 3

1 2 3
4 5 6
0 7 8

Heuristic value : 2
level : 3

1 2 3
4 5 6
7 8 0

Heuristic value : 0
level : 3
PS D:\Frieden\COLLEGE\Sem_06\AIL>
```



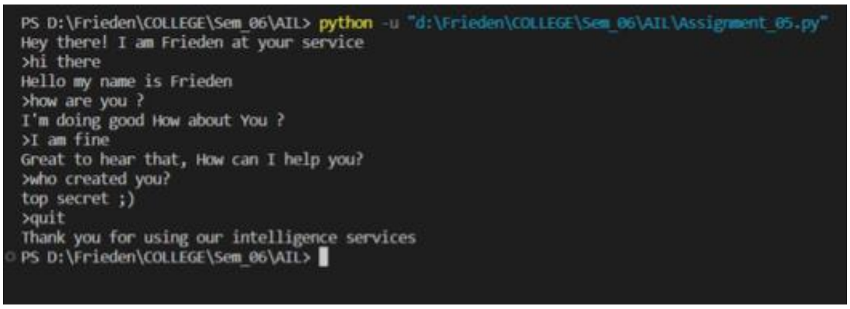
```
                    --- Greedy Algorithm ---

User manual :
1. Selection sort
2. prims
3. kruskals
4. Dijisktra
5. Exit

Enter your choice : 1

Enter Size of array : 5

Enter array to sort : 15 26 11 22 10

Sorted array is : 10 11 15 22 26

User manual :
1. Selection sort
2. prims
3. kruskals
4. Dijisktra
5. Exit

Enter your choice : 2
Number of vertex : 5
Number of edge: 7

Enter the source, destination and cost :
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9

Minimum spanning tree formed using edges :
Edge            Weight
0       1       2
1       2       3
0       3       6
1       4       5
```



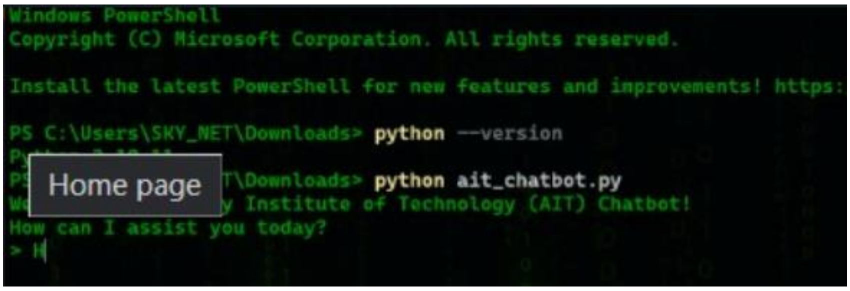
```
3 4 9

Minimum spanning tree formed using edges :
Edge            Weight
0       1       2
1       2       3
0       3       6
1       4       5

Minimum Cost Spanning Tree: 16


User manual :
1. Selection sort
2. prims
3. kruskals
4. Dijisktra
5. Exit

Enter your choice : 3
Number of vertex : 5
Number of edge: 7

Enter the source, destination and cost :
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9

Minimum spanning tree formed using edges :
Edge            Weight
0       1       2
1       2       3
1       4       5
0       3       6

Minimum Cost Spanning Tree: 16
```



```
User manual :
1. Selection sort
2. prims
3. kruskals
4. Dijisktra
5. Exit

Enter your choice : 4
4               21
5               11
6               9
7               8
8               14


User manual :
1. Selection sort
2. prims
3. kruskals
4. Dijisktra
5. Exit

Enter your choice : 5
```



```
PS D:\Frieden\COLLEGE\Sem_06\AIL> python -u "d:\Frieden\COLLEGE\Sem_06\AIL\Assignment_05.py"
Hey there! I am Frieden at your service
>hi there
Hello my name is Frieden
>how are you ?
I'm doing good How about You ?
>I am fine
Great to hear that, How can I help you?
>who created you?
top secret ;)
>quit
Thank you for using our intelligence services
PS D:\Frieden\COLLEGE\Sem_06\AIL> 
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https:

PS C:\Users\SKY_NET\Downloads> python --version
Py
PS                 T\Downloads> python ait_chatbot.py
We                 y Institute of Technology (AIT) Chatbot!
How can I assist you today?
> H
```