IDAA Assignment 5

Finding k-nearest and k-farthest neighbours of a diagonally moving point of a matrix

By - Mohd Mohtashim Nawaz
- Harsh Aryan
- Mayank Sunil Taksande
- Sneha Mishra

The Problem Statement:

There is given a randomly generated matrix.

We are given with the moving point which moves diagonally. We need to find k- nearest and k-farthest neighbours of that moving point at a current instance where k is provided by the user along with the starting position of the mobile point.

Introduction

First we generate a 100X100 2-D array filled with random numbers. Now according to the problem statement we traverse diagonally in the matrix and find k-farthest and k-nearest neighbours of the mobile point.

Notice that k-nearest points mean the nearest points assuming one step is k units. Similarly, k-farthest points are the farthest points assuming one step is k units.

Now in upcoming slides we will see the algorithm to do so.

Algorithm:

1. Take k and coordinates of the starting point as input. Create a 100 X 100 matrix. Fill it with random numbers.
2. Traverse to the next diagonally located cell, starting from the start point as specified by the user. After every step movement, give a 2 seconds pause, so that the movement can be shown.
3. Find the k-nearest points using k which has been taken as input from the user. The points are to be considered in every direction, that is 8 such neighbours are to be considered for all cases except the corners or edge cells.
4. Find the k-farthest points using k as given already. Again the points are to be found along all possible directions. Then, move to the next iteration.

Algorithm to find nearest neighbours:

```
int n ← 100 (as given in question)

def nearest_neighbours (int:matrix[][n],int:n,int:k,int:i,int:j)
        If (j+k) <= (n-1) then
                Print matrix[i][j+k]
        If (j-k)>=0 then
                Print matrix[i][j-k]
        If (i+k)<=(n-1) then
                Print matrix[i+k][j]
        If (i+k)<=(n-1)  and  (j+k)<=(n-1) then
                Print matrix[i+k][j+k]
        If (i+k)<=(n-1)  and (j-k)>=0 then
                Print matrix[i+k][j-k]
        If (i-k)>=0 then
                Print matrix[i-k][j]
        If (i-k)>=0 and (j-k)>=0 then
                Print matrix[i-k][j-k]
        If (i-k)>=0 and (j+k)<=(n-1)
                Print matrix[i-k][j+k]
end
```

Time Analysis:

Best case: Ω(26)

Worst case: O(52)

Algorithm to find farthest neighbours:

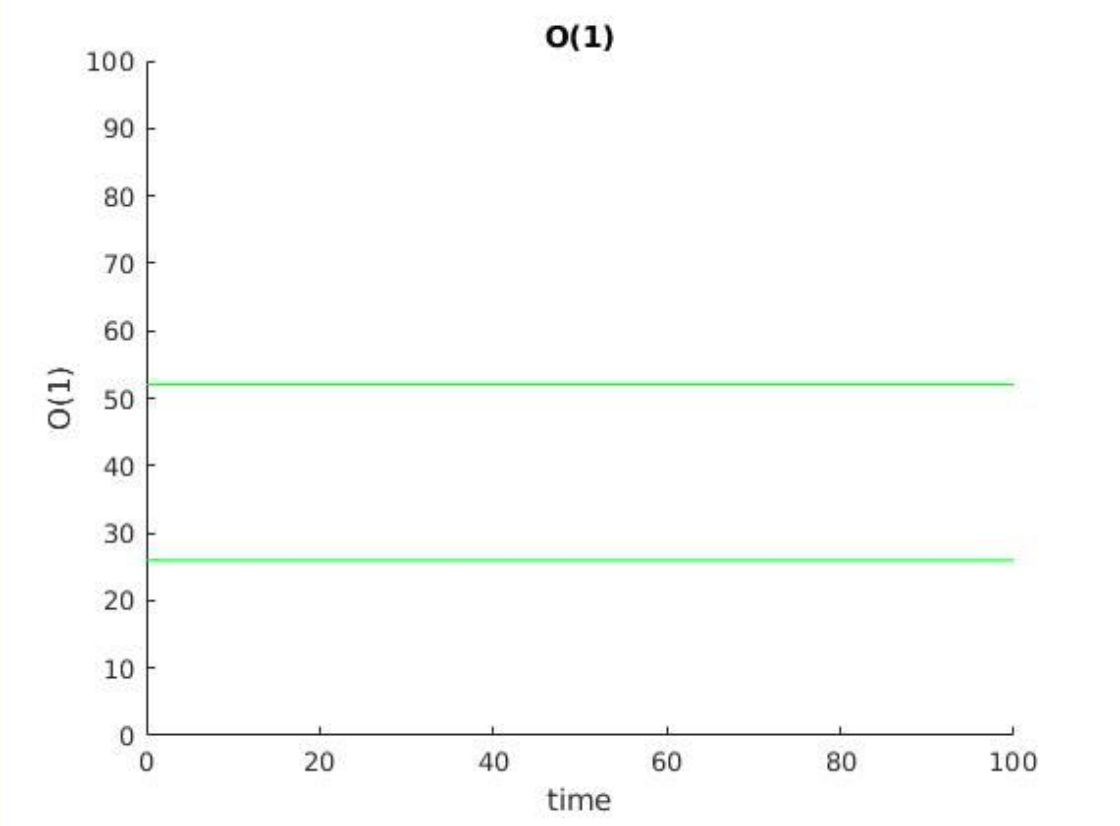
```
int n ← 100 (as given in question)

def farthest_neighbours (int:matrix[][n],int:n,int:k,int:i,int:j)
        If (j+k)<=(n-1) then
                Print matrix[i][n-k]
        If (j-k)>=0 then
                Print matrix[i][k-1]
        If (i+k)<=(n-1) then
                Print matrix[n-k][j]
        If (i+k)<=(n-1) and (j+k)<=(n-1) then
                Print matrix[n-k][n-k]
        If (i+k)<=(n-1) and (j-k)>=0 then
                Print matrix[n-k][k-1]
        If (i-k)>=0 then
                Print matrix[k-1][j]
        If (i-k)>=0 and (j-k)>=0 then
                Print matrix[k-1][k-1]
        If (i-k)>=0 and (j+k)<=(n-1) then
                Print matrix[k-1][n-k]
end
```

Time Analysis:

Best case: Ω(26)

Worst case: O(52)

Graph of nearest neighbours algorithm and farthest neighbours algorithm:



Algorithm to generate random matrix:

int n ← 100 (as given in question)

```
def generatematrix (int:matrix[][n],int:n)
        srand (time(NULL))
        //standard library function
        int:i,j
        For i=0 to n-1 do
                For j=0 to n-1 do
                        Matrix[i][j] ← rand()%10
                 //generating random integers

end
```

Time Analysis:

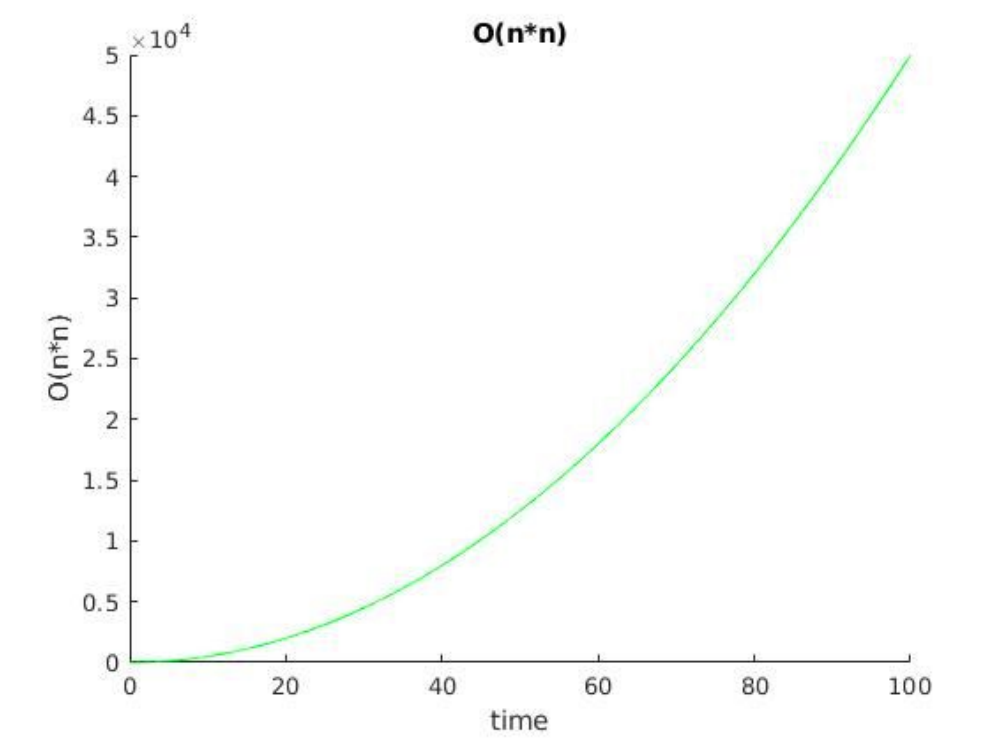
Best case: Ω(5*n*n+2)

taking n=100, it gives 50002

Worst case: O(5*n*n+2)

taking n=100, it gives 50002

Graph of algorithm to generate random matrix:



Main Algorithm:

```
def main()
        int:matrix[n][n]
        int:i,j
        generatematrix(matrix,n)
        For i=0 to n-1 do
                For j=0 to n-1
                        Print matrix[i][j]

  int:x,y,k
                Input → k,x,y
                If x=0 and y=0
                        For i=x to n-1 do
                                For j=y to n-1 do
                                    If i=j then
                                        Print i,j
                                        //position of point
   Print matrix[i][j]
                                        //value at that
                                point
                                nearest_neighbours
                                        (matrix,n,k,i,j)
```

```
farthest_neighbours
                    (matrix,n,k,i,j)
                    sleep(2)
          else if x=n-1 and y=n-1 then
               For i=x to 0 do
               For j=y to 0 do
                    If i=j
                     Print i,j
                     Print matrix[i][j]
          nearest_neighbours
                         (matrix,n,k,i,j)
                         farthest_neighbours
                         (matrix,n,k,i,j)

                         sleep(2)
          else if x=0 and y=n-1 then
                    For i=x to n-1 do
                         For j=y to 0 do
                                   If i+j=n-1 then
                                       Print i,j
                          //current point
                            Print matrix[i][j]
                           //value at current point
                           nearest_neighbours
                           (matrix,n,k,i,j)
```

```
                farthest_neighbours
                        (matrix,n,k,i,j)

                        sleep(2)

        else if x=(n-1) and y=0 then
            For i=x to 0 do
            For j=y to n-1 do
                If i+j = n-1 then
                    Print i,j
                    //current point
                    Print matrix[i][j]
                    //value at point
                    nearest_neighbours
                   (matrix,n,k,i,j)
                    farthest_neighbours
                  (matrix,n,k,i,j)
                    sleep(2)
          Else
                Print “Invalid Point”
                return 0
```

Time Analysis:

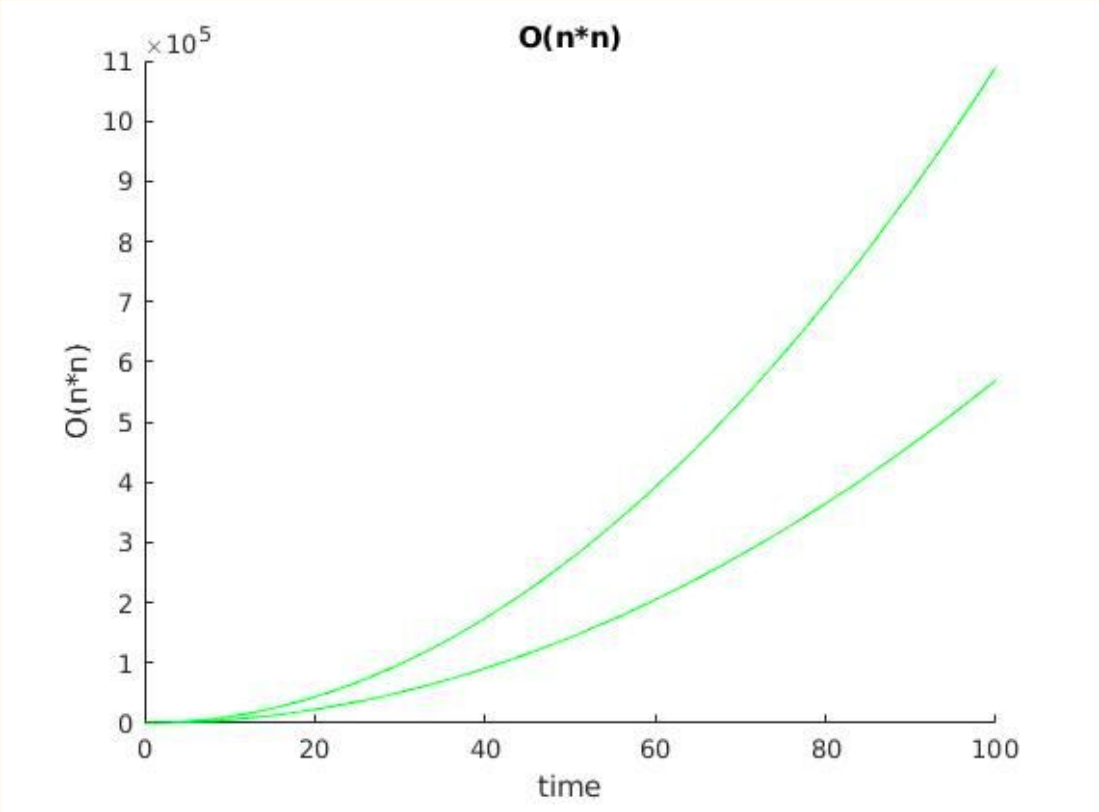
Best case: 6 + n*n*(5+26+26) = 57n*n + 6

Ω(n^2)

Worst case: 6 + n*n*(5+52+52) = 109n*n+6

O(n^2)

Graph of main algorithm:



Result

Overall Time Complexity

Best case - Ω(n^2)

Worst case-O(n^2)

Overall Space Complexity

Best case -Ω(100*100)

Worst case-O(100*100)

Alternate Approaches:

The above approach involves a number of conditional statements to ensure the boundary condition of the mobile point.
An alternate approach can be considered to avoid such comparisons by removing the need to check for boundary conditions for every point on the diagonal.
We can do so by constructing a matrix of 3*3 around the mobile point with it being in the middle of the matrix for all those cases where we do not have any element from original matrix we will put value 0 in those places

Thus a matrix of form :
3 1 4
1 5 9
2 6 5
can be visualised as :
0 0 0 0 0
0 3 1 4 0
0 1 5 9 0
0 2 6 5 0
0 0 0 0 0

Alternate Approaches:

Another Alternate Approach can be implemented by the following pseudo code:

```
row_limit = count(array);
if(row_limit > 0){
  column_limit = count(array[0]);
  for(x = max(0, i-1); x <= min(i+1, row_limit); x++){
    for(y = max(0, j-1); y <= min(j+1, column_limit); y++){
      if(x != i || y != j){
        print array[x][y];}
    }
  }
}
```

Conclusion

We have developed an algorithm to solve the given problem, that is to find the k-farthest and k-nearest neighbours of a mobile point in a 100 X 100 matrix.

There could have been some other approaches however the one used is simple and both time and space efficient. Thus we have come up with the best possible approach of the given problem.

Thank You