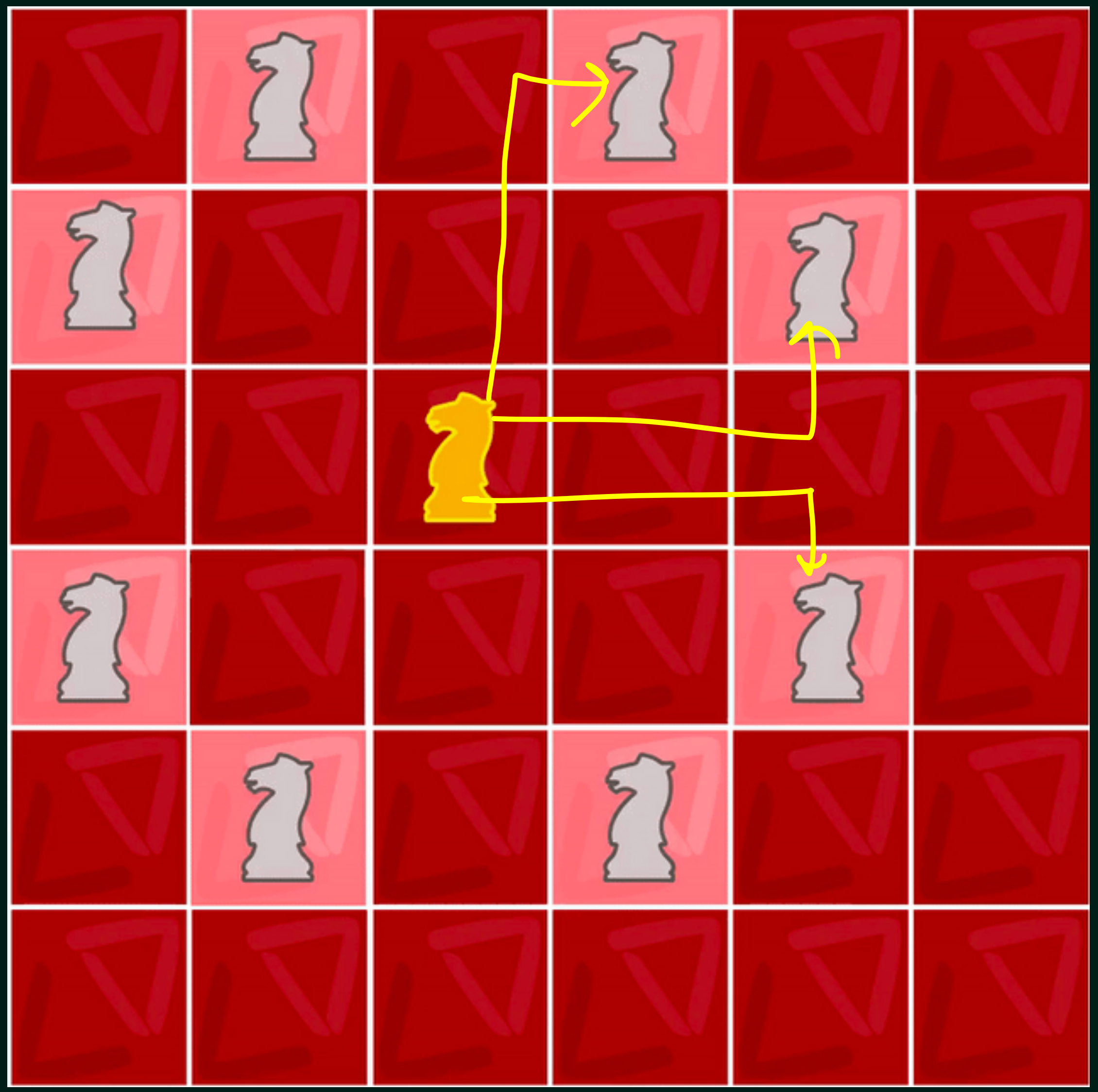
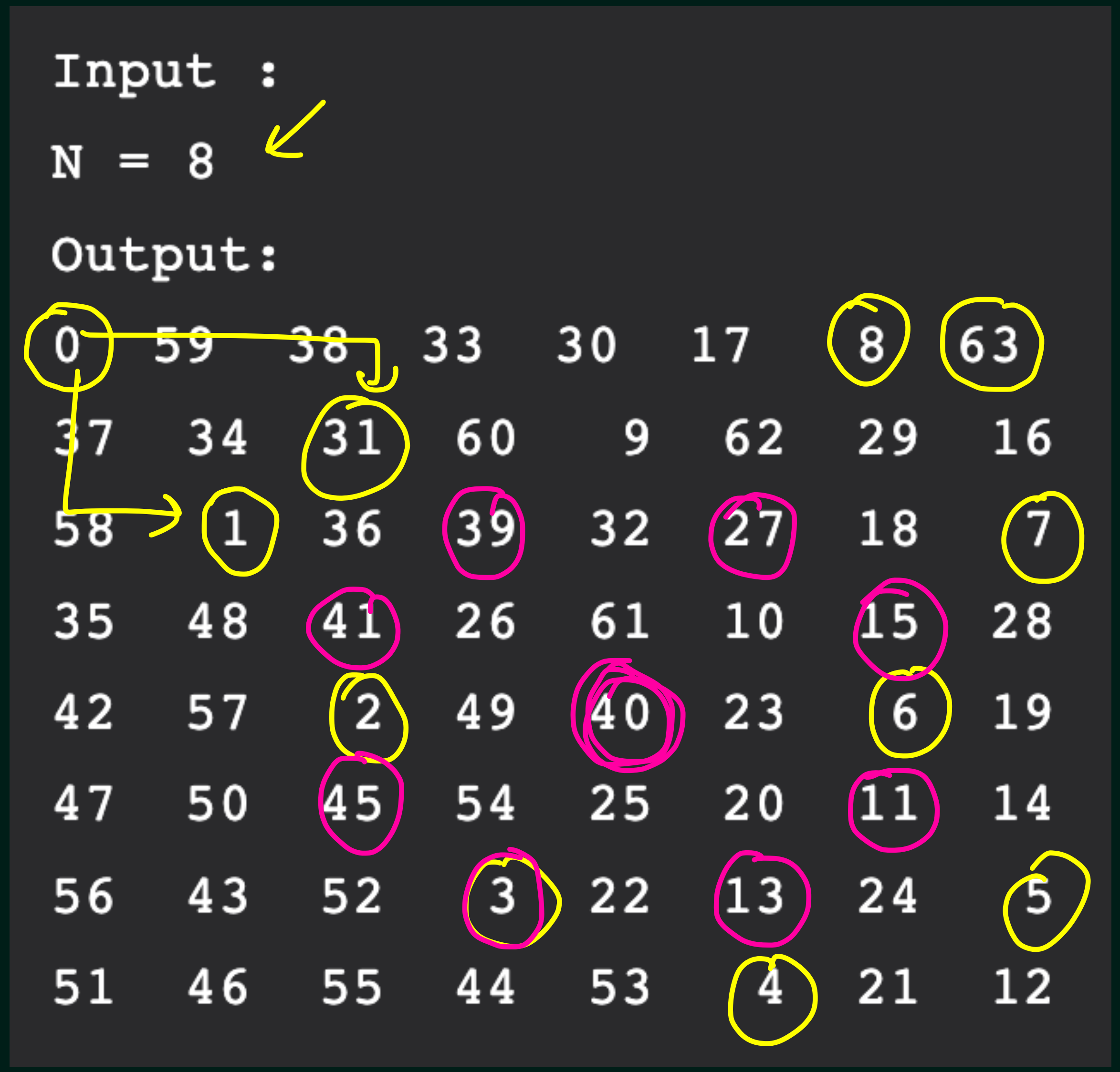
Backtracking - 3

The Knight's Tour Problem

Given a N*N board with the Knight placed on the first block of an empty board. Moving according to the rules of chess knight must visit each square exactly once. Print the order of each cell in which they are visited.



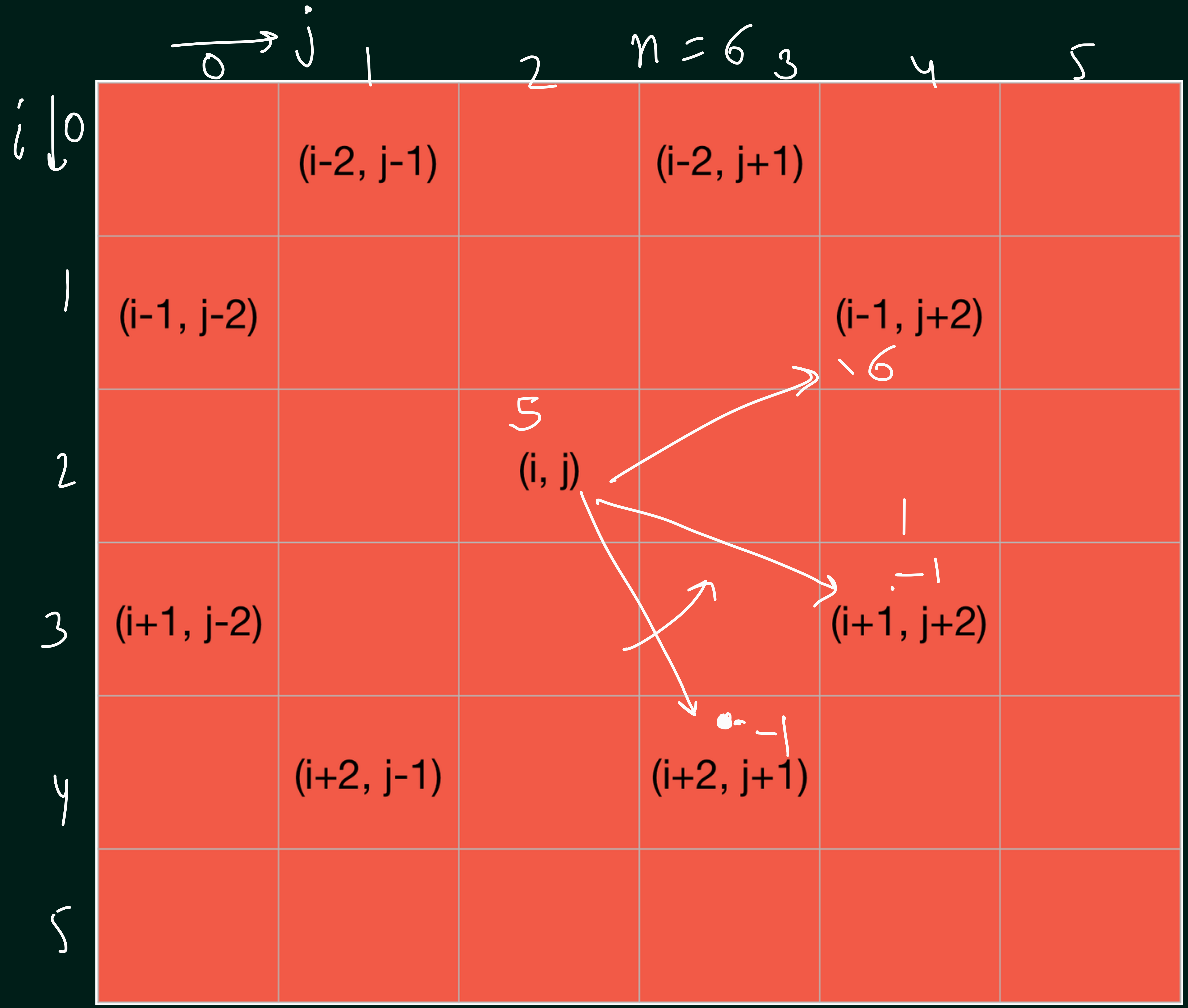


```
Input :
N = 8
Output:
0   59  38  33  30  17  8   63
37  34  31  60  9   62  29  16
58  1   36  39  32  27  18  7
35  48  41  26  61  10  15  28
42  57  2   49  40  23  6   19
47  50  45  54  25  20  11  14
56  43  52  3   22  13  24  5
51  46  55  44  53  4   21  12
```

64

2D Array

The Knight's Tour Problem



i \ j	0	1	2	3	4	5
0		(i-2, j-1)		(i-2, j+1)		
1	(i-1, j-2)				(i-1, j+2) 6	
2			5 (i, j)		1	
3	(i+1, j-2)				-1 (i+1, j+2)	
4		(i+2, j-1)		-1 (i+2, j+1)		
5						

(x, y)

↓ ↓

moveX[] = { 2, 1, -1, -2, -2, -1, 1, 2 }

moveY[] = { 1, 2, 2, 1, -1, -2, -2, -1 }

Use the Common structure of Backtracking Solution

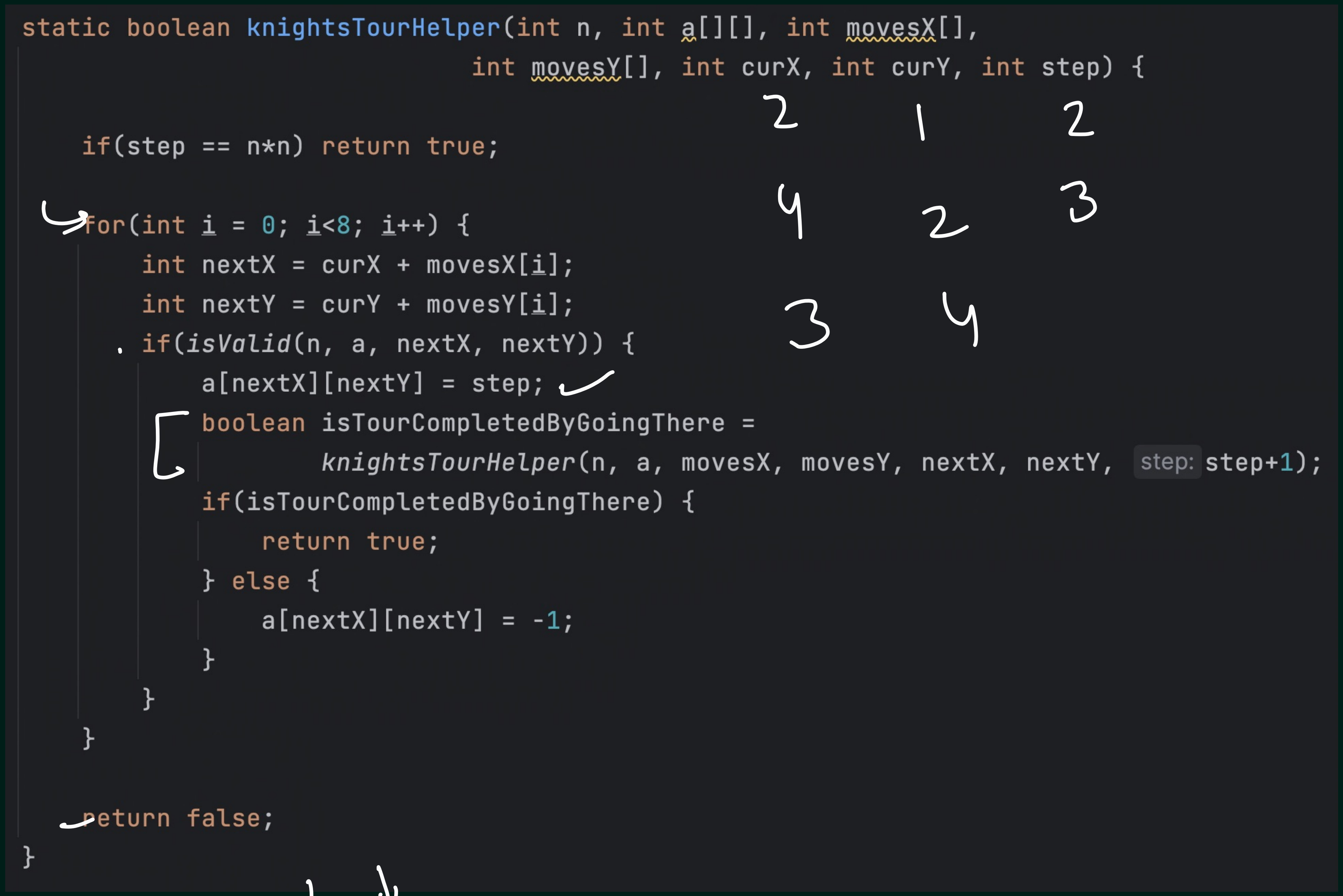
```
boolean findSolutions(n, other params) :
    if (found a solution) :
        displaySolution();
        return true;

    for (val = first to last) : ✓
        if (isValid(val, n)) : ✓
            applyValue(val, n);
            ~~if~~ (findSolutions(n+1, other params))
                return true;
            removeValue(val, n); // Backtrack
        return false;
```

The Knight's Tour Problem

n=5

	0	1	2	3	4
0	0	x 5	16	9	12
1	-1	10	13	x 4	15
2		1	6	11	8
3		-1	-1	14	x 3
4	-1		x 2	7	-1

1

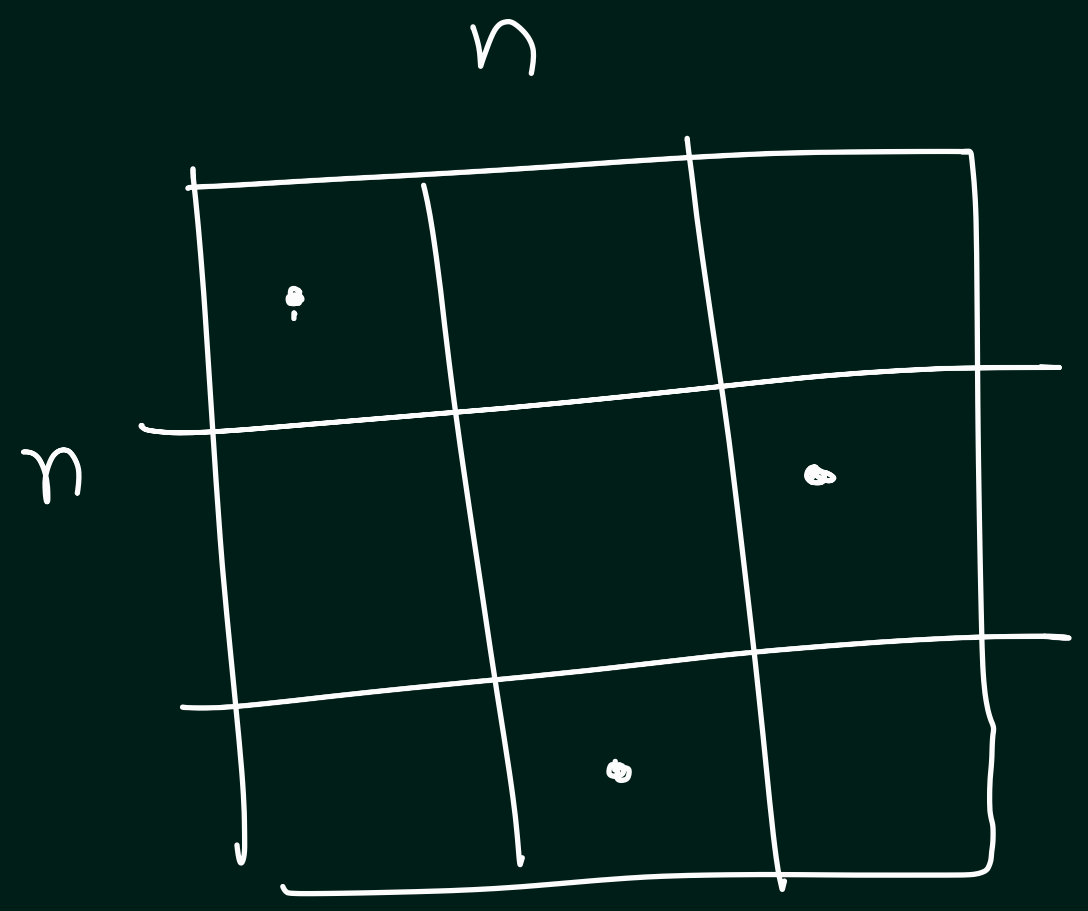


```java
static boolean knightsTourHelper(int n, int a[][], int movesX[],
                                 int movesY[], int curX, int curY, int step) {

    if(step == n*n) return true;

    for(int i = 0; i<8; i++) {
        int nextX = curX + movesX[i];
        int nextY = curY + movesY[i];
        if(isValid(n, a, nextX, nextY)) {
            a[nextX][nextY] = step;
            boolean isTourCompletedByGoingThere =
                    knightsTourHelper(n, a, movesX, movesY, nextX, nextY, step: step+1);
            if(isTourCompletedByGoingThere) {
                return true;
            } else {
                a[nextX][nextY] = -1;
            }
        }
    }

    return false;
}
```

2 1 2
4 2 3
3 4

moveX[ ] = { 2, 1, -1, -2, -2, -1, 1, 2 }

moveY[ ] = { 1, 2, 2, 1, -1, -2, -2, -1 }



$$\boxed{O(\gamma^{n^2})} \rightarrow O(k^{n^2})$$