Laboratory work # 2

Student: HU Riqian
Student ID: 20321114
Timus Name: hduads2022_20321114
Mail: jhlxhrq@163.com

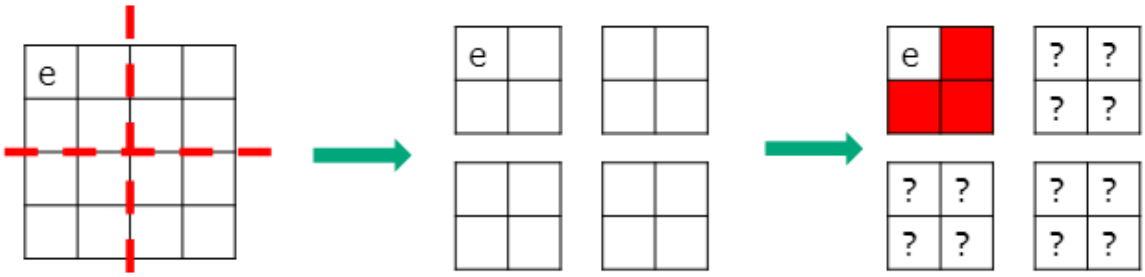
Problem # 1401. Gamers

Screenshot from Timus:

9803053	16:57:34 1 Apr 2022	hduads2022_20321114	1401	Java 1.8	Accepted		0.14	6 564 KB

Explanation of algorithm:

1. We can transfer the whole chess board into 4 average pieces.



2. Then we break the problem down into four sub-questions.
3. We use the `paveBoard` method, which is used recursively until the board is reduced to a 1*1 board, and the recursion ends.

Computational complexity of algorithm:

$$O(4^n)$$

Source code:

```
import java.io.*;

public class Gamers {
    public static void main(String[] args) throws IOException{
        new Gamers().run();
    }

    StreamTokenizer in;
    PrintWriter out;
    static int WIDTH;
    static int[][] chessBoard;
```

```
    static int index = 0;

    int nextInt() throws IOException {
        in.nextToken();
        return (int)in.nval;
    }

    void run() throws IOException
    {
        in = new StreamTokenizer(new BufferedReader(new
InputStreamReader(System.in)));
        out = new PrintWriter(System.out);
        initChessBoard(nextInt());
        paveBoard(0, 0, nextInt() - 1, nextInt() - 1, WIDTH);
        printBoard(chessBoard);
        out.flush();
    }

    void initChessBoard(int n) {
        WIDTH = (int)Math.pow(2, n);
        chessBoard = new int[WIDTH][WIDTH];
    }

    // This method is used recursively until the board is reduced
    // to a 1*1 board, and the recursion ends.
    void paveBoard(int tr, int tc, int pr, int pc, int width) {
        if (width == 1) {
            return;
        }
        int count = ++index;
        int h = width / 2;

        if (pr < (tr + h) && pc < (tc + h)) {
            paveBoard(tr, tc, pr, pc, h);
        } else {
            chessBoard[tr + h - 1][tc + h - 1] = count;
            paveBoard(tr, tc, tr + h - 1, tc + h - 1, h);
        }

        if (pr < (tr + h) && pc >= (tc + h)) {
            paveBoard(tr, tc + h, pr, pc, h);
        } else {
            chessBoard[tr + h - 1][tc + h] = count;
            paveBoard(tr, tc + h, tr + h - 1, tc + h, h);
        }

        if (pr >= (tr + h) && pc < (tc + h)) {
            paveBoard(tr + h, tc, pr, pc, h);
        } else {
            chessBoard[tr + h][tc + h - 1] = count;
            paveBoard(tr + h, tc, tr + h, tc + h - 1, h);
        }

        if (pr >= (tr + h) && pc >= (tc + h)) {
            paveBoard(tr + h, tc + h, pr, pc, h);
```

```
        } else {
            chessBoard[tr + h][tc + h] = count;
            paveBoard(tr + h, tc + h, tr + h, tc + h, h);
        }
    }

    void printBoard(int[][] board) {
        for (int i = 0; i < WIDTH; i++) {
            for (int j = 0; j < WIDTH; j++) {
                out.print(board[i][j] + " ");
            }
            out.println();
        }
    }
}
```