Homework 2 Fenchao Du

2	3	4	4	4	4	4	4	…	…	4	4	4	4	4	4	3	2
3	4	5	5	5	5	5	5	…	…	5	5	5	5	5	5	4	3
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
…	…							…	…							…	…
…	…							…	…							…	…
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6	🟥	🟥	6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
4	5	6	6	6	6	6	6			6	6	6	6	6	6	5	4
3	4	5	5	5	5	5	5	…	…	5	5	5	5	5	5	4	3
2	3	4	4	4	4	4	4	…	…	4	4	4	4	4	4	3	2

For a ship with 3-unit length.
Assume a ship is randomly placed on a set of coordinates the grid and the probability of each set is same.
The numbers represent how many unique sets of coordinates that contain this block.
e.g., Block "2" is in the corner. If the ship contains this block, there are only 2 unique ways to place the ship.
Every block will have 6 ways except the borders. In large scale of grid size, we can ignore the optimization of border computation.
We want to find the minimum blocks to try until we hit one part of the ship in worst case scenario.
The "yellow" blocks above would not be part of a same 3-unit ship. Hence, if we try hitting all "yellow" blocks, we will definitely hit the ship.
After the ship is hit, we only need constant computation to verify surrounding blocks.
e.g., Blue block is hit. Then we only need at most 8 more checks of red blocks for a 3-unit ship.

Assume hit(int x, int y) is O(1), grid size = n and ship len = m.
In worst case, it takes n^2/m to find the blue block.
Then a constant computation of 4*(m-1)
Time complexity is O(n^2/m + 4m – 4). If we know m = 3, O(n^2/3)

We don't allocate additional memory for temporary computation.
All hit checks can share the O(1) memory.
We use double vector for output. Return output uses O(2m) memory.

Homework 2 Fenchao Du

```
vector<vector<int>> findShip(int gridSize, int shipLen = 3) {
    vector<vector<int>> ship;
    for (int row = 0; row < gridSize; ++row) {
        for (int col = 0; col < gridSize; ++col) {
            if (row%shipLen == col%shipLen && hit(row, col)) {
                for (int offset = shipLen-1; offset > 0; --offset) {
                    if (col-offset >= 0 && hit(row, col-offset)) {
                        ship.push_back({row, col-offset});
                    }
                    if (row-offset >= 0 && hit(row-offset, col)) {
                        ship.push_back({row-offset, col});
                    }
                }
                ship.push_back({row, col});
                for (int offset = 1; offset < shipLen; ++offset) {
                    if (col+offset < gridSize && hit(row, col+offset)) {
                        ship.push_back({row, col+offset});
                    }
                    if (row+offset < gridSize && hit(row+offset, col)) {
                        ship.push_back({row+offset, col});
                    }
                }
                return ship;
            }
        }
    }
    return ship;
}
```