Naval Battle Editorial

By D, we denote the maximal coordinate of all ships.

Subtask 1

We can do this on a case by case basis or we can make a useful *observation:* If two ships are going to collide, it will happen exactly after $t_{1,2} = \frac{|x_1-x_2|+|y_1-y_2|}{2}$ steps. (Their distance must decrease by 2 in each step.)

Therefore we can move each ship by t steps and check if their positions are the same.

Subtask 2

First we notice that for any ship, collision is no longer possible when one coordinate becomes smaller than 0 or greater than D. Any collision after that would mean that the other colliding ship would have its initial coordinate outside this range, leading to a contradiction.

We can do a simulation in $\mathcal{O}(ND^3)$ as described in the task statement. For each step, we simply allocate a $D \times D$ sea and check collisions within it.

Subtask 3

As D^2 is too much to allocate, we need to check collisions in a better way. Using an `unordered_map` storing all occupied squares will suffice. This runs in $\mathcal{O}(ND)$.

Subtask 4

We cannot simulate all steps, so we skip the unimportant ones. If we compute the time of the next collision $t_{\min}$ as the minimum of possible collision times for every pair of ships (see Subtask 1), we can jump to it by moving each ship $t_{\min}$ times.

Computing the next collision will take $\mathcal{O}(N^2)$ time and there can be at most $\mathcal{O}(N)$ collisions. This is $\mathcal{O}(N^3)$ total.

Subtask 5

We can notice that the time of the next collision can change only when one of the ships collides. We compute all possible $\mathcal{O}(N^2)$ collisions, sort them by ascending time and go through them, checking each one if it really happened.

However, there is a catch, as more than two ships can collide at once. We can solve this by keeping a death time for each ship and if one ship has death time in the past, then the collision won't happen.

This will take $\mathcal{O}(N^2 \log N)$ because of sorting.

Subtask 6

First we need to notice that only ships with different directions on the same southeast diagonal will collide. Furthermore, a ship traveling south has to lie to the right of the ship traveling east. Also, there cannot be any ship on the diagonal between these two as it would collide with one of them first.

We can imagine ships on a diagonal from south to east as parentheses. East-traveling ships as opening ones and south as closing ones. Then matching parentheses represent colliding ships. (For matching parenthesis that are next to each other, this follows from previous paragraph. Then we can cross them out and continue with the other matching parentheses.)

As we need to sort ships on a diagonal, this will take $\mathcal{O}(N \log N)$.

Subtask 7

We will combine approaches from subtasks 5 and 6. We will do an event-based simulation but only have at most $\mathcal{O}(N)$ events in our queue.

For finding the next collision, we will again separate ships into diagonals and now also horizontal and vertical lines. We will call these *bidirections*. We will store ships in a bidirection as a linked list with an external lookup for each ship. Imagining ships in bidirection as parentheses, the next collision will correspond to two matching parentheses next to each other. We put all these collisions into a priority queue.

Now, we need to handle an event. First we need to check if it really happened (see Subtask 5). If it did, we remove both ships from this bidirection. Otherwise, we remove the already sunk ship from this bidirection. Either way we can now have a new matching parenthesis next to each other, so we construct new collision if necessary.

In the priority queue, we store only 3 possible collisions — one for each other direction. There can be at most $\mathcal{O}(N)$ collisions and each priority queue operation takes $\mathcal{O}(\log N)$. The final solution has time complexity $\mathcal{O}(N \log N)$.