香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

# T253 – Peaceful Pirate Pairs

Isaac Chan {snowysecret}

2025-05-10

# Background

Problem idea by `snowysecret`

Preparation by `QwertyPi` (Thanks!)

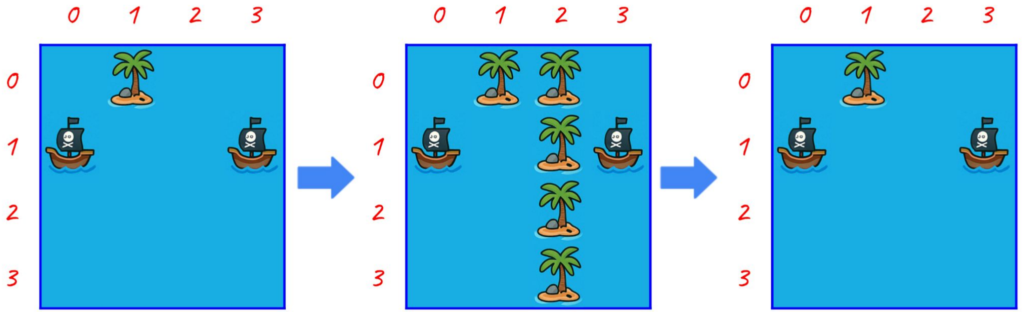
Figures by `snowysecret`

Presented by `snowysecret`

(Easiest problem in the problem set!)

# The Problem

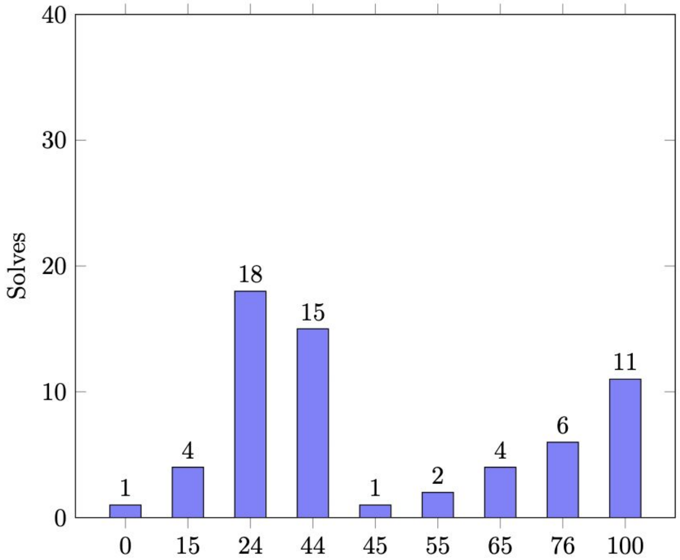
- The ocean is N x N in size; some cells are pirate ships and some cells are islands.
- Two ships are **in conflict** if they are on the same row + no island is strictly between them. The ocean is peaceful if no pair of ships is in conflict.
- Q updates: Apply or remove barriers on columns. A barrier turns all cells on one column into islands.
- Determine if the ocean is peaceful after each update.



# Subtasks

| Subtask | Score | Constraints |
|---|---|---|
| 1 | 15 | $N, M \leq 30$, $Q \leq 10$, **only type 1 updates** |
| 2 | 9 | $N \leq 2000$, $Q \leq 10$, **only type 1 updates** |
| 3 | 20 | $Q \leq 10$, **only type 1 updates** |
| 4 | 11 | **Only type 1 updates** |
| 5 | 21 | All pirate ships on row 0 |
| 6 | 24 | No additional constraints |

# Statistics




First solved by **dbsjkjk** at **0:47:38**

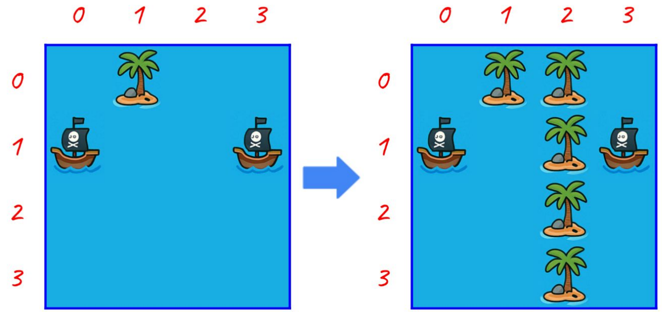
Second solved by **s20251** at **0:47:49**

# Subtask 1 (15%): N, M ≤ 30, Q ≤ 10, only type 1 updates

The constraints are pretty small here. It suffices to naively implement what you are told to do!

(1) Maintain type 1 updates, i.e. turn an entire column into islands.

(2) Search for a pair of pirate ships that are **in conflict**; or report that there are none and the ocean is peaceful.



# Subtask 1 (15%): N, M ≤ 30, Q ≤ 10, only type 1 updates

(1) Maintain type 1 updates, i.e. turn an entire column into islands.

We can simply use a 2D array **`arr[N][N]`** to store the state of the ocean. For example, **`arr[i][j] = 0`** if the cell (i, j) is open sea or a pirate ship; and **`arr[i][j] = 1`** if the cell (i, j) is an island.

When we have to do a type 1 update on column c, simply replace **`arr[i][c] = 1`** for $0 \le i < N$.

Time complexity: O(N) per update → O(NQ) over all updates

*(Remember to check the indexing! In some problems we have indices ranging from 1 to N, in others indices range from 0 to N – 1.)*

# Subtask 1 (15%): N, M ≤ 30, Q ≤ 10, only type 1 updates

(2) Search for a pair of pirate ships that are **in conflict**; or report that there are none and the ocean is peaceful.

- Brute force over all pairs of pirate ships.
- If a pair of pirate ships lies on $(r, c_1)$ and $(r, c_2)$ respectively, where $c_1 < c_2$, brute force over all cells between them (i.e. cells $(r, c_1 + 1)$, $(r, c_1 + 2)$, ..., $(r, c_2 - 1)$) and check if there is an island between them.
- Time complexity: $O(M^2N)$ after each update → $O(M^2NQ)$ overall

# Subtask 1 (15%): N, M ≤ 30, Q ≤ 10, only type 1 updates

Overall time complexity: $O(M^2NQ)$

Fortunately, N, M and Q are quite small so this algorithm can pass easily!

Expected score: 15

# Subtask 2 (9%): $N \le 2000$, $Q \le 10$, only type 1 updates

Overall time complexity: ~~$O(M^2NQ)$~~ **TLE :( Time to optimize!**

Step (1) runs in O(NQ) time, so we don't have to change it.

Revisit Step (2): Search for a pair of pirate ships that are **in conflict**; or report that there are none and the ocean is peaceful.

Can we do this more efficiently? (Hint: Look at the constraints. What time complexity does this hint?)

# Subtask 2 (9%): $N \leq 2000$, $Q \leq 10$, only type 1 updates

Overall time complexity: ~~$O(M^2NQ)$~~ **TLE :( Time to optimize!**

Step (1) runs in O(NQ) time, so we don’t have to change it.

Revisit Step (2): Search for a pair of pirate ships that are **in conflict**; or report that there are none and the ocean is peaceful.

Can we do this more efficiently? (Hint: Look at the constraints. What time complexity does this hint?)

**... Maybe $O(N^2Q)$?**

# Subtask 2 (9%): N ≤ 2000, Q ≤ 10, only type 1 updates

The $N^2$ factor hints sweeping the grid once, instead of brute forcing over all pairs of pirate ships. Let's iterate over each row from left to right.

- For every row, we keep track of the last thing we saw: is it a pirate ship, or is it an island?
- If we see two pirate ships in a row, we know they must be in conflict!

# Subtask 2 (9%): N ≤ 2000, Q ≤ 10, only type 1 updates

Hence, we can compute the answer in $O(N^2)$ time just by iterating over each row from left to right!

Across all Q queries → that's $O(N^2Q)$ time, which passes this subtask!

Expected score: 24

# Subtask 3 (20%): $Q \le 10$, only type 1 updates

Now we can no longer store the entire grid, since $N \le 2 \times 10^5$.

However, we can still maintain information about each row – and that is all we need, since we were iterating over each row separately!

# Subtask 3 (20%): $Q \leq 10$, only type 1 updates

Instead of storing all the cells (including open sea cells), we instead keep N vectors, one for each row, storing the 'special cells' (island / pirate ships) in **ascending column number**.

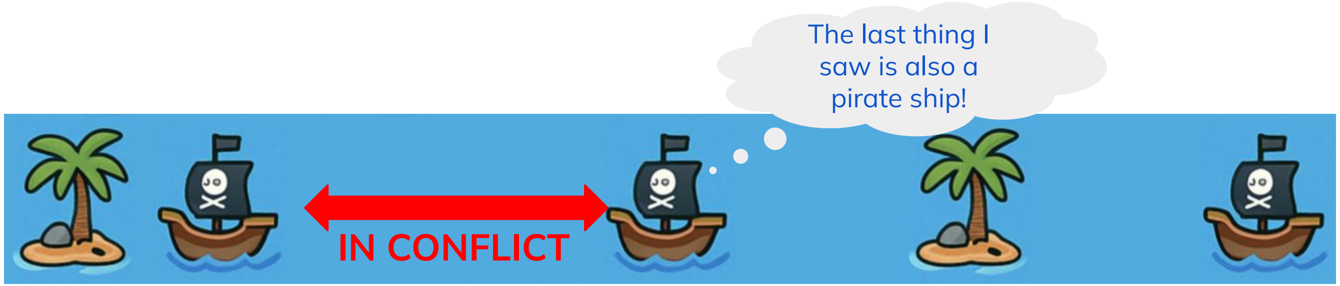
This is precisely a sped-up version of iterating over each row... except we are now doing it in O(M+K) time! // M = # pirate ships, K = # island cells

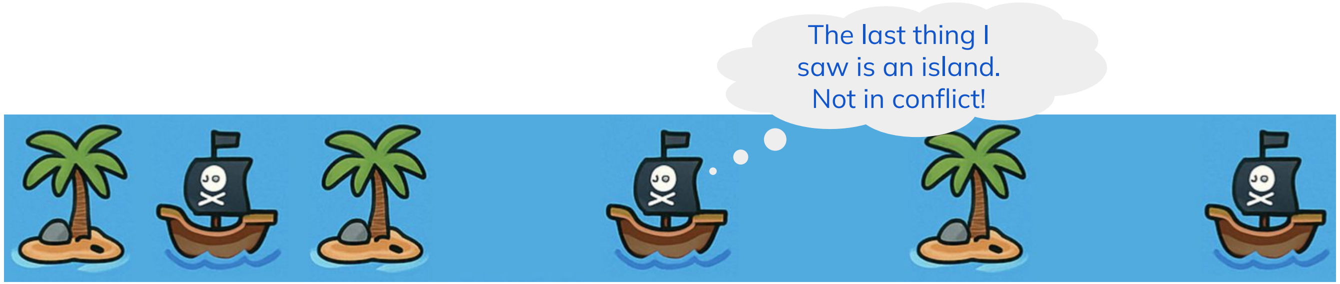
Can we handle updates efficiently?

- Each type 1 update is N "vector inserts". Then we need to sort each vector again, which takes O(N + (M+K) log (M+K)) time in the worst case.
- Do this Q times: O(Q(N + (M+K) log (M+K))
- Sufficient to pass this subtask, but it won't get you too far in this problem

# Subtask 3 (20%): Q ≤ 10, only type 1 updates

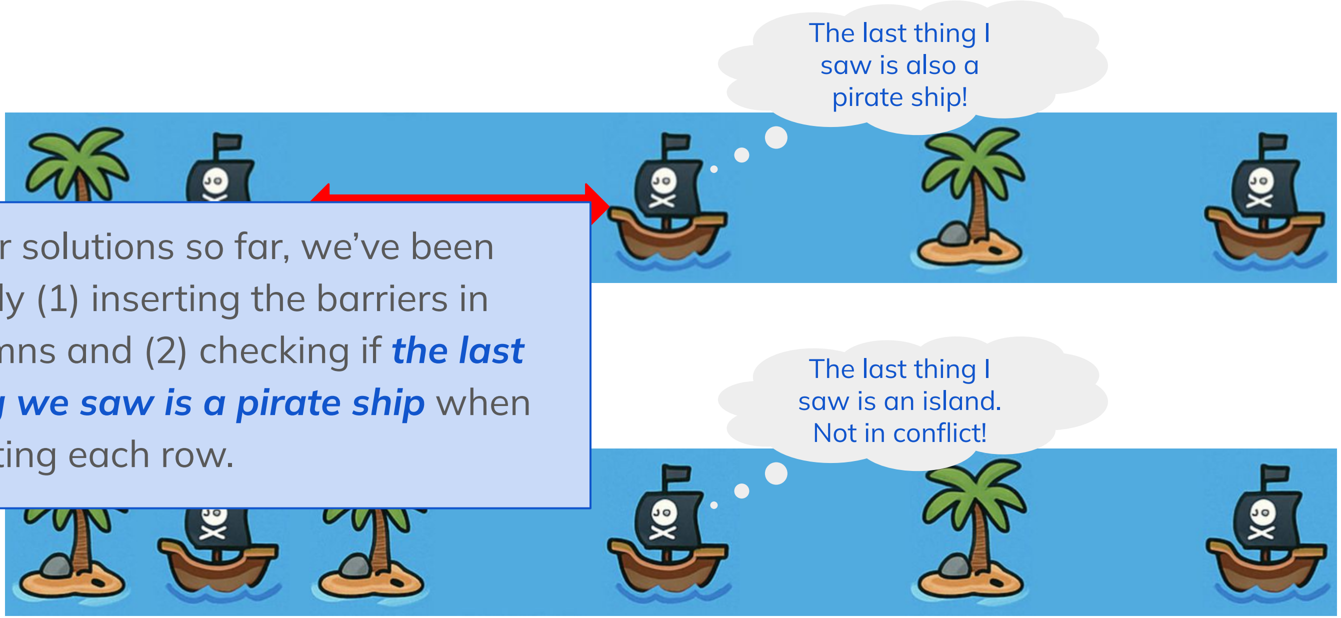
Let's reconsider the diagram from subtask 2.



# Subtask 3 (20%): Q ≤ 10, only type 1 updates

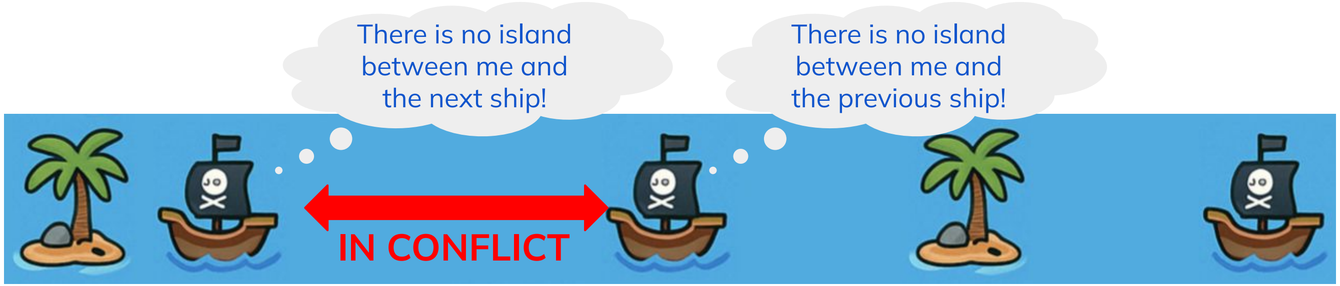
Let's reconsider the diagram from subtask 2.

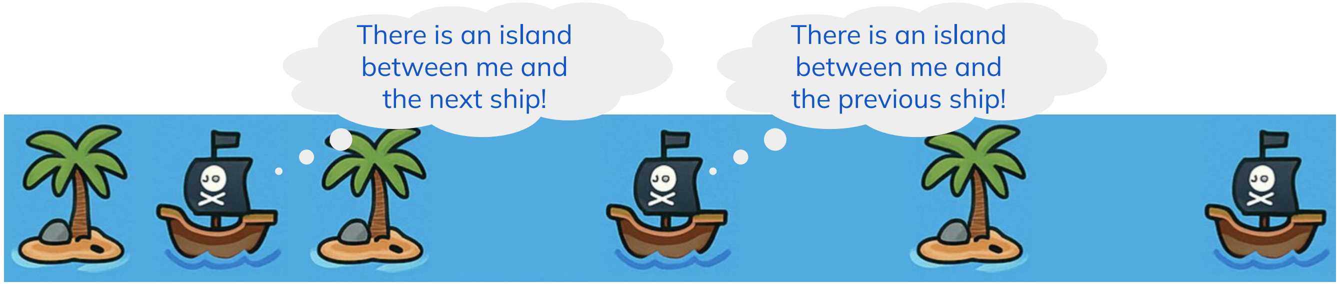



In our solutions so far, we've been simply (1) inserting the barriers in columns and (2) checking if ***the last thing we saw is a pirate ship*** when iterating each row.

# Subtask 3 (20%): Q ≤ 10, only type 1 updates

Alternatively...

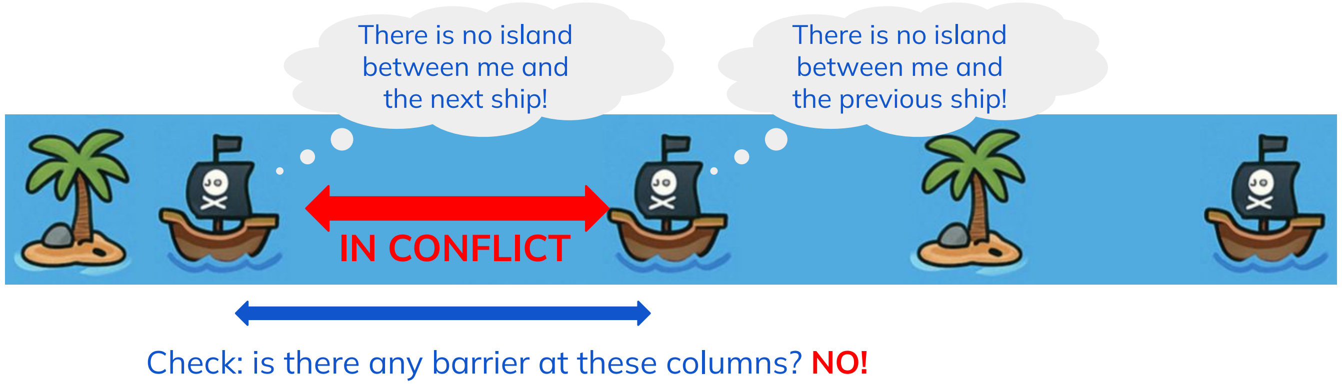


# Subtask 3 (20%): Q ≤ 10, only type 1 updates

Formally, for two pirate ships at $(r, c_1)$ and $(r, c_2)$,

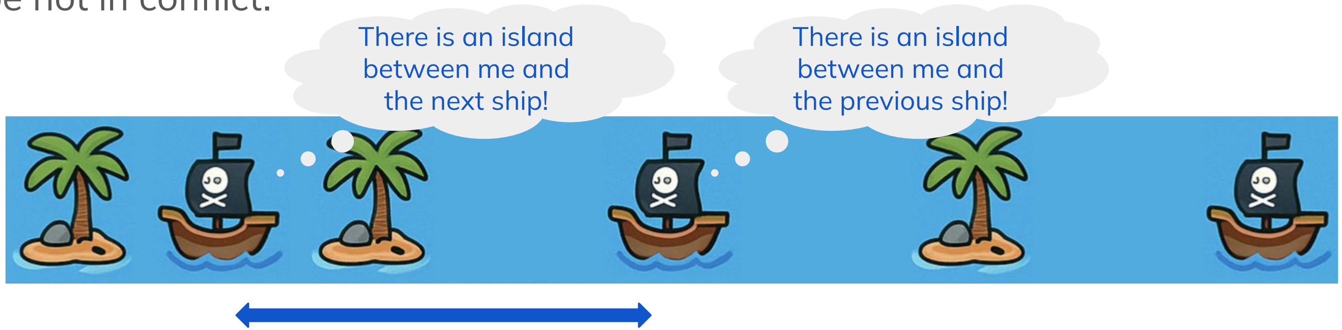
- If they were initially *not in conflict*, then they stay not in conflict forever.
- Otherwise, we want to look for a barrier at column k, for some $c_1 < k < c_2$.
    - (Case 1) If no such barrier exists, then the two ships are in conflict.



# Subtask 3 (20%): $Q \le 10$, only type 1 updates

Formally, for two pirate ships at $(r, c_1)$ and $(r, c_2)$,

- If they were initially *not in conflict*, then they stay not in conflict forever.
- Otherwise, we want to look for a barrier at column $k$, for some $c_1 < k < c_2$.
  - (Case 2) If such barrier exists, then cell $(r, k)$ is an island cell which causes $(r, c_1)$ and $(r, c_2)$ to be not in conflict.




Check: is there any barrier at these columns? **YES!**

# Subtask 3 (20%): $Q \le 10$, only type 1 updates

For two ships at $(r, c_1)$ and $(r, c_2)$ that are initially in conflict, where $c_1 < c_2$,

- If there exists a barrier at column k (where $c_1 < k < c_2$) then they are no longer in conflict.

That is, the ocean is peaceful (no pairs are in conflict) **if and only if**:

- **For all pairs of ships at (i, L–1) and (i, R+1) that are initially in conflict, there exists a barrier at column k (where $L \le k \le R$)!**
- Note if ships at (r, c) and (r, c+1) are in conflict, the answer is always NO
- Please ask now if you have any questions

# Subtask 3 (20%): Q ≤ 10, only type 1 updates

That is, the ocean is peaceful (no pairs are in conflict) **if and only if**:

- **For all pairs of ships at (i, L–1) and (i, R+1) that are initially in conflict, there exists a barrier at column k (where L ≤ k ≤ R)!**

Hereinafter we will refer to this as the *[L, R]-constraint*.
The ocean is peaceful iff all [L, R]-constraints are satisfied.

# Subtask 3 (20%): $Q \le 10$, only type 1 updates

How does all the analysis help us in this subtask?

- Note that we can have O(M) [L,R]-constraints, since there are only M pirate ships.
- After each update, iterate over all O(M) [L,R]-constraints and check if at least one of the barrier columns k satisfies $L \le k \le R$
- O(MQ) per update, $O(MQ^2)$ overall with small constant

Expected score: 44

# Subtask 4 (11%): Only type 1 updates

We need to consider ≤ $10^5$ updates. Fortunately, we only have type 1 updates.

- You may have observed that the return array should be in the form [No; No; No; ...; No; Yes; Yes; ...; Yes]. Since we just keep adding new barriers, we will eventually reach a point where the ocean is peaceful. After that, the ocean will always be peaceful.
- Hence we can binary search on the prefix of "No" answers. To check whether adding a certain set of barriers satisfies all [L,R]-constraints, we can use prefix sums to optimize the checking to O(M).
- Time complexity (exc. preprocessing): O((M + N) log Q).

Expected score: 55

# Subtask 5 (21%): All pirate ships on row 0

- Core question: is it true that for all [L, R]-constraints, there is at least one barrier currently, such that its column k satisfies $L \le k \le R$?
- Can we transform this question into an easier form?

# Subtask 5 (21%): All pirate ships on row 0

- Core question: is it true that for all [L, R]-constraints, there is at least one barrier currently, such that its column k satisfies $L \le k \le R$?
- Can we transform this question into an easier form?

- For each [L, R]-constraint, **how many barriers are there currently**, such that its column k satisfies $L \le k \le R$?
  ⇒ What is the minimum number of barriers across all [L,R]-constraints?
  ⇒ Is that minimum number equal to 0 or not?

# Subtask 5 (21%): All pirate ships on row 0

- Let's try to maintain, for each [L,R]-constraint, the count of barriers such that its column lies in between [L, R].
- The ocean is peaceful if the minimum count over all constraints > 0.

- **Type 1 update**: count++ for all relevant constraints
- **Type 2 update**: count-- for all relevant constraints
- **Check peaceful or not**: Query minimum over all counts, check if > 0
- Naively this can be achieved in O(NQ)

# Subtask 5 (21%): All pirate ships on row 0

- Can we exploit the special conditions of this subtask?

# Subtask 5 (21%): All pirate ships on row 0

- Can we exploit the special conditions of this subtask?
- YES! Under the constraints of this subtask, all [L, R]-constraints form **disjoint** (i.e. non-overlapping) ranges.
- Let’s sort all the constraints (if we haven’t already).
- Each type 1 / 2 update would update the count to one [L, R]-constraint. Binary search on the sorted constraints to find which one (if any).

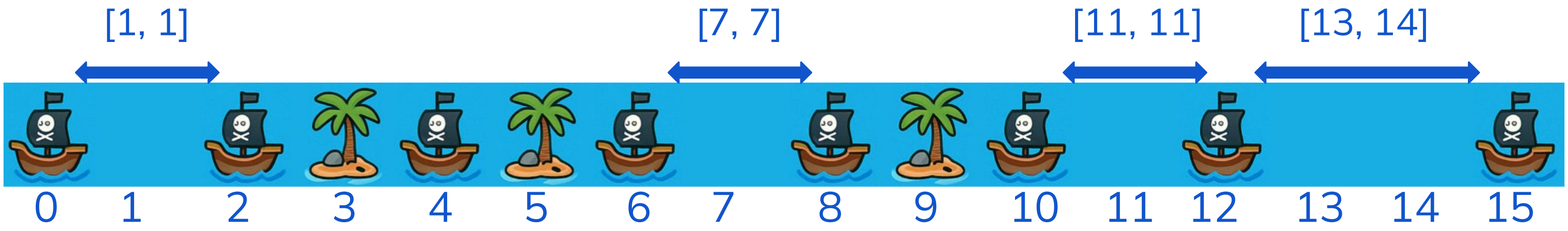
# Subtask 5 (21%): All pirate ships on row 0

- **Type 1 update**: count++ for single constraint
- **Type 2 update**: count-- for single constraint
- **Check peaceful or not**: Query minimum over all counts, check if > 0

- Hence this becomes a standard ***"point update, range minimum"*** task. The most straightforward way to solve this is to use a segment tree.
- Or you can just maintain a frequency array of size M storing the counts, and an additional variable counting how many counts are equal to 0.
- Time complexity (exc. preprocessing): O((M + Q) log M) / O(M + Q)
- Expected score: 76 (Cumulative)

# Subtask 5 (21%): All pirate ships on row 0 (Example with N = 16)

| | | |
|---|---|---|
| Add_Barrier(1) | Counts for each constraint: 1, 0, 0, 0 | No |
| Add_Barrier(7) | Counts for each constraint: 1, 1, 0, 0 | No |
| Add_Barrier(11) | Counts for each constraint: 1, 1, 1, 0 | No |
| Add_Barrier(5) | Counts for each constraint: 1, 1, 1, 0 | No |
| Add_Barrier(14) | Counts for each constraint: 1, 1, 1, 1 | Yes (min ≥ 1) |
| Remove_Barrier(7) | Counts for each constraint: 1, 0, 1, 1 | No |



# Subtask 6 (24%): No additional constraints

- Now that pirate ships can be on any column, we no longer have the guarantee that ranges are disjoint.
- **Key Observation**: Consider any two constraints $[L_1, R_1]$ and $[L_2, R_2]$. If $L_1 \le L_2 \le R_2 \le R_1$ then there is no need to consider $[L_1, R_1]$.

- Should be easy to reason why!
- Since $[L_2, R_2]$ is a “tighter” constraint than $[L_1, R_1]$, if the $[L_2, R_2]$-constraint is satisfied, then there exists some barrier on column k s.t. $L_2 \le k \le R_2$.
- Surely that means $L_1 \le k \le R_1$ as well.
- So $[L_1, R_1]$ is redundant.

# Subtask 6 (24%): No additional constraints

- By removing all the redundant constraints, we are left with a set of constraints $\{[L_1, R_1], [L_2, R_2], \ldots, [L_k, R_k]\}$ with the following special property: It is possible to sort them in a way such that
  $L_1 < L_2 < \ldots < L_k$
  and
  $R_1 < R_2 < \ldots < R_k$

- That being said, every "column update" operation now affects a **single contiguous range** of this new sequence of constraints.

# Subtask 6 (24%): No additional constraints

- **Type 1 update**: count++ for range of constraints
- **Type 2 update**: count-- for range of constraints
- **Check peaceful or not**: Query minimum over all counts, check if > 0

- Hence this becomes a standard ***"range update, range minimum"*** task. You can solve this using a segment tree.
- Time complexity (exc. preprocessing): O((M + Q) log M)
- Expected score: 100

# Conclusion

- This is a relatively standard & straight-forward data structures problem, with a small observation towards the end
- If you did not AC this task, I suggest you **upsolve it later today** after listening to this solution explanation
- Practice more on standard data structures problems so it becomes second nature to you!