General Judging Notes

- Basic format: students (in groups of 1, 2 or 3) will come from their table to you with their solution to a problem, and explain it to you. They can also come if they're stuck and need some help. If you think they should move on to the next problem, hand it to them (printed on a piece of paper) and they'll go back to their table.
- Be helpful and encouraging. Some weaker students will likely attend, so identify their strengths and make them feel good about them.
- If a student/group presents a solution that doesn't quite work, preferably they will fix it on the spot, or go back to their table to think more, and you can provide a hint if they want one. Alternatively, if you think it's more appropriate (e.g. they seem stressed), consider letting them move on to the next problem.
- Ask for clarification if their algorithm is unclear, it should be a two-way dialogue.
- Ask them to state and justify their algorithm's time complexity.
- Ask for an informal justification of their algorithm's correctness (since proving correctness is a key skill learned in the course).
- Adjust the level of detail and rigour required to move on based on the length of the queue. At the beginning the queue will likely build up, so let people move on very quickly.
- Feel free to ask other tutors if you need any assistance when judging.
- If the queue is empty, feel free to get up and talk to students at their tables and see how they're going. Or join another tutor and judge with them.

Question 1 *Matroid*

Let $M_1 = (S, I_1)$ be a matroid and $M_2 = (S, I_2)$ is the uniform matroid. Prove that $M_3 = (S, I_1 \cap I_2)$ is also a matroid.

Proving that $\emptyset \in (I_1 \cap I_2)$ and the hereditary property should be trivial. We show the exchange property holds.

Let $X, Y \in (I_1 \cap I_2)$, with $|X| > |Y|$. Since M_1 is a matroid, then there exists some $x \in X \setminus Y$ such that $Y \cup \{x\} \in I_1$. As $|Y \cup \{x\}| \leq |X|$, then $(Y \cup \{x\}) \in I_2$. Thus, $Y \cup \{x\} \in I_1 \cap I_2$ so M_3 fulfils the exchange property and is a matrix.

Question 2 *Skyscraper Design*

You are an architect looking at the horizon which has N buildings casting silhouettes. The i'th skyscraper is represented by the range it takes up horizontally along the horizon line $[l_i, \ldots, r_i]$ and its height h_i. Determine the total silhouette area in $\mathcal{O}(N \log N)$.

Use a sweepline algorithm.

Maintain the set H of currently active skyscraper heights in some kind of SBBST structure, sweeping from left to right.

Let's assume we are currently at horizontal position x, and we move to a skyscraper which turns on at $l > x$. We add $(l - x) \times \max(H)$ to our answer before adding the building height h to H. Similarly, if we encounter the ending of a building $r > x$ we add $(r - x) \times \max(H)$

before removing the building height h from H. Update x to the new event point position.

This runs in $\mathcal{O}(N \log N)$, since there are $2N \sim \mathcal{O}(N)$ event points and we do $\mathcal{O}(\log N)$ work at each point.

Question 3 *Manipulation I*

You want to manipulate a group of people by sending viral misinformation which will spread within friend groups. There are N friend groups which happen to be in the form of ranges: the i'th friend group has $[l_i \dots r_i]$ being friends. If *you* send misinformation to a person, they'll spread it to everyone in all their friend groups (people won't spread misinformation if they hear it from friends). Find in $\mathcal{O}(N \log N)$ the minimum number of people you have to send misinformation to so that you can manipulate everyone.

Draw some diagrams. Can we reduce this into a known problem?

The problem reduces to minimising the number of points required such that every interval contains at least one point. This is the **Interval Stabbing** problem, which has a standard greedy solution.

In particular, a key observation is that we should only consider right endpoints of intervals as potential 'stabbing' locations. Secondly, since the range with the rightmost right endpoint must be stabbed, we can greedily stab intervals in descending order of right endpoint location.

Efficient implementation yields an $\mathcal{O}(N \log N)$ solution.

The reduction doesn't need to be detailed that much; just focus on the details of the interval stabbing problem for revision purposes.

Question 4 *Manipulation II*

Again, you want to manipulate people. There are N friends and M one-way friendships, the i'th of which is (u_i, v_i) – person u_i will talk to v_i. If a person is sent misinformation (whether from you or another person), they'll spread it to everyone they talk to. Design an $\mathcal{O}(N + M)$ algorithm to find the minimum number of people you need to misinform to achieve the same goal.

How do we represent these relationships?

In constructing a graph $G = (N, M)$ where nodes represent friends and edges represent friendships, quickly we are able to make a crucial observation: in a strongly connected component, misinformation will spread to everyone as long as one person gets affected.

Motivated by this, we construct a reduced graph, finding all SCCs utilising a greedy algorithm and subsequently collapsing them into nodes. The reduced graph forms a directed acyclic graph (DAG). Greedily, by targeting the nodes with an in-degree of zero, we can spread our misinformation to everyone.

The final answer is thus the number of sources in the reduced graph. Finding SCCs and constructing the reduced graph takes $\mathcal{O}(N+M)$, and finding source nodes takes $\mathcal{O}(N)$. Thus, the time complexity is the desired $\mathcal{O}(N + M)$.

- If they seem confident, check they know the details of the SCC-finding algorithm.
- Although self-evident, you can get people to prove why the greedy selection of source vertices results in a minimal solution.

Question 5 *Subarray Splitting*

There is an array of N positive integers, the largest being a. Spend $\mathcal{O}(N(\log a + \log N))$ time to split the array into K non-empty subarrays, maximising the subarray with minimum sum.

If we can make the minimum sum subarray have sum $\geq x$, can we make it $\geq y$ where $y < x$?

We observe a monotonicity condition on the sum of the minimum sum subarray – if it is possible to split the array such that our min-sum subarray has sum $\geq x$, then we need only verify whether it can have sum $\geq y$ where $y > x$.

This motivates binary searching for the greatest x such that the min-sum subarray can have sum $\geq x$. Verification can be easily done in $\mathcal{O}(N)$ via greedily scanning through the array and ensuring all subarrays have sum $\geq x$. If the end of the array is reached before there are K full subarrays, it cannot be done.

The search space is $[1, Na]$, so binary search runs in $\mathcal{O}(N(\log a + \log N))$, which solves.

Question 6 *Lazy Runner*

You are a runner starting at vertex u in a directed, weighted graph $G = (V, E)$. You are lazy and want to run the shortest cycle starting and ending at u. Luckily, to accomplish your task, an $\mathcal{O}(1)$ oracle ShortestPath(G, u) is provided:

- **Input.** A directed, weighted graph $G = (V, E)$, where each edge has a positive weight $w(e)$; a vertex $u \in V$.
- **Output.** A series of positive numbers representing the length of the shortest path from u to any reachable vertex in G (and ∞ otherwise).

Describe and analyse a linear time algorithm to accomplish your task.

One of the main problems is accidentally finding cycles which visit the same vertex twice. Is this actually a problem?

Consider running ShortestPath(G, u) (store this output in A where $A(v)$ is shortest distance from u to v) and then running ShortestPath(G, v) for all $v \neq u$ such that v is reachable from u (store this in B where $B(v)$ is shortest distance from v to u).

Let's denote the subset of nodes v where u and v are mutually reachable with the set S. Only nodes in S will have valid outputs from both runs of ShortestPath. These are also the only nodes which are in contention for a cycle.

Claim: The shortest cycle will be the minimum $A(v) + B(v)$ for all $v \in S$.

Proof: The claim is false only when the 'cycle' with $A(v) + B(v)$ length is degenerate – i.e. it visits the same nodes twice. However, if that were the case, we would be able to shrink the cycle while strictly decreasing the cost, turning it non-degenerate (cut and splice the cycle at the first common vertex). Hence, this cannot be the cycle with shortest length starting and

ending at u, leading to a contradiction.

Hence, this algorithm which finds the min in $\mathcal{O}(N)$ time is correct.

Make sure they identify and show how the problem can be resolved.

Question 7 *Monsters*

There are N monsters attacking your village! You and the monsters take turns attacking each other. Monster i has h_i health and deals a_i damage. Each turn, you can choose to attack one monster. If the monster's health reaches 0, it immediately dies. Then, all living monsters will attack you back. As the hero, you have plot armour and will never die. However, you still want to minimise the damage you take. Describe and analyse a polynomial-time algorithm that takes the N health and damage stats of each monster and computes the order to attack the monsters.

When do you want to switch targets?

Given an order of monsters to attack, write out the total damage you take.

You sort the monsters in descending order of $\frac{a_i}{h_i}$, and kill one monster at a time in this order.

First observation: if we choose to attack a monster, we want to keep attacking it until it dies. Now we just find an order of monsters to attack. Intuitively, we want to attack high attack and low health monsters first.

We will prove the greedy ordering by $\frac{a_i}{h_i}$ is optimal with contradiction. Let A be the optimal order, and assume that it is not the solution given by the greedy ordering. That means there exists two monsters, p and q with p appearing directly before q in A, such that $\frac{a_p}{h_p} < \frac{a_q}{h_q}$. We can change the order we attack p and q without affecting the other monsters. Let T be the time taken to attack all the monsters before p. The damage taken from attacking them in the given order is

$$D_1 = a_p(S + h_p - 1) + a_q(S + h_p + h_q - 1) = S(a_p + a_q) + h_p(a_p + a_q) - (a_p + a_q) + h_q a_q.$$

If we reverse the order of p and q, the damage is

$$D_2 = a_q(S + h_q - 1) + a_p(S + h_q + h_p - 1) = S(a_p + a_q) + h_q(a_p + a_q) - (a_q + a_p) + h_p a_p.$$

The difference $D_1 - D_2 = h_p a_q - h_q a_p$. Since $\frac{a_p}{h_p} < \frac{a_q}{h_q}$, $h_p a_q - h_q a_p > 0$, so this means more damage is taken in the first ordering than the second ordering. This is a contradiction, so that means every optimal solution must have $\frac{a_i}{h_i}$ in descending order. Hence the greedy solution is optimal.

Ask for some rudimentary proof that the order is correct (beyond "it just works").

Question 8 *Robbery*

You are a robber and have found a long horizontal row consisting of n houses. Each house has either 1 coin, or no coins. You know there is at least 1 coin across all houses, and at most k coins. You can perform two operations:

- Scan the number of coins in a range of houses $[L, R]$.

- Visit a house i. If you visit a house with a coin, you can collect it, otherwise you get shot and die.

Describe and analyse a polynomial time algorithm to collect all the coins scanning $O(k \log n)$ times.

How can we divide this problem into smaller subproblems?

We can use divide and conquer, repeatedly splitting the range in half. Assume we know the total number of coins in the range $[l, h]$ is x. We can query the number of coins from $[l, (l+h)/2]$. If it is 0, we know all the coins lie in $((l+h)/2, h]$ and we don't need to check the left range. If it is x, we know all the coins in $[L, R]$ lie in $[L, (l+h)/2]$ and we don't need to check the right range. Otherwise, we recurse down both paths.

The number of times we have to split the range until we're certain a house has a coin is $O(\log n)$. Given there are k coins, we have to repeat this operation maximum k times for a total number of queries $\sim O(k \log n)$.

Relatively straightforward divide and conquer.

Question 9 *Meetings (BOI '11 P5)*

There is a meeting with N people. Each person takes A minutes to present. After this, discussion and voting takes place, taking B minutes. Thus, the total time taken is $N \times A + B$. The delegation realises this can be made more efficient by splitting into groups; for instance, when splitting into two groups (group 1 with x people and group 2 with $N - x$ people):

(1) Group 1 takes $T_1 = x \times A + B$ time to come to a consensus

(2) At the same time, Group 2 takes $T_2 = (N - x) \times A + B$ time to come to a consensus

(3) They meet up after $\max(T_1, T_2)$ and each group presents their ideas (A minutes each, for $2A$ minutes total) before voting for B minutes

Sometimes, it's better to split into more than 2 groups, and sometimes each group can individually be split itself. In general, as many splits as wanted can be made, as long as they are all combined at the end. As a special case, a group with 1 person takes 0 time to arrive at a decision, since 1 person need not present to themselves. What is the minimum time it takes for the delegation to come to consensus? (Answer in $\mathcal{O}(N^2)$)

Let $T(n)$ be the minimum time it takes to resolve a meeting of n people. When n increases, $T(n)$ is non-decreasing. What does this tell us about the best strategy when we divide a group of size n into k subgroups?

If we want to split N people into K groups, it's optimal to greedily make the largest group as small as possible. To do this, set most groups to have size $\lceil \frac{N}{K} \rceil$, and the last one to be smaller.

So, letting $T(n)$ be the minimum time it takes to resolve a meeting of n people, we realise that

$$T(n) = \min_{1 \le i < n} \left(\underbrace{T\left(\frac{n-1}{i} + 1\right)}_{\text{resolve largest group}} + \underbrace{T(i)}_{\text{get together}} \right)$$

The final answer is $T(N)$. This can be computed by setting $T(1) = 0$ and computing $T(2), T(3), \ldots, T(N)$ in that order. To prove time complexity, note that each $T(i)$ takes $\mathcal{O}(N)$ time to compute, and there are N of these. Hence, the final time complexity is $\mathcal{O}(N^2)$ as desired.

- The DP is easy so hopefully they don't even realise they just rederived DP (gulp).
- If they solve this quickly, you can ask a follow up question on how to optimise this solution to $\mathcal{O}(N\sqrt{N})$.

Question 10 *String Matching*

Consider a fixed alphabet Σ. You have a text $T[0 \ldots N-1]$ with numbers in the alphabet, and a shorter pattern $P[0 \ldots M-1]$.

(1) Propose an $\mathcal{O}(N \log N)$ algorithm to find all positions that P occurs in T as a substring.

(2) Imagine that both T and P now can have wildcard characters which match any character in the alphabet. Solve the same problem.

This is an FFT problem.

Solution to (1): There is a match at position i iff for all $x \in [i, i+M)$, $T[x] = P[x-i]$. Therefore, imagine considering the sum

$$S[i] = \sum_{x=0}^{M-1} (P[x] - T[i+x])^2 = \sum_{x=0}^{M-1} (P[x]^2 - 2P[x]T[i+x] + T[i+x]^2)$$

Only when $S[i] = 0$ is i a matching. Therefore, we aim to compute $S[i]$ over all $i \in [0, N)$ in $\mathcal{O}(N \log N)$.

The $P[x]^2$ and $T[i+x]^2$ terms can be computed in $\mathcal{O}(1)$ per value of i using prefix sums. It remains to compute an array C such that

$$C[i] = \sum_{x=0}^{M-1} P[x]T[i+x]$$

This is a standard use case of the Fast Fourier Transform, often called computing the "cross-correlation". In particular, construct

$$A(z) = T[0]z^0 + T[1]z^1 + \cdots + T[N-1]z^{N-1}$$

and

$$B(z) = P[0]z^{M-1} + P[1]z^{M-2} + \cdots + P[M-1]z^0$$

Then, since the k'th coefficient of $(A * B)$ for $k \geq M-1$ is

$$\sum_{j=k-(M-1)}^{\min(N-1,k)} A[j]B[k-j]$$

The $(i + M - 1)$'th coefficient is

$$\sum_{j=i}^{i+M-1} A[j]B[i + M - 1 - j] = \sum_{x=0}^{M-1} A[x + i]B[M - 1 - x] = \sum_{x=0}^{M-1} P[x]T[i + x] = C[i]$$

as desired. So, by computing $C[i] = (A * B)[i + M - 1]$, the problem can be solved in $\mathcal{O}(N \log N)$.

Solution to (2): Only a small modification is required. If we define $P[i] = 0$ when P's i'th character is a wildcard, and similarly for T, then consider a new sum:

$$S[i] = \sum_{x=0}^{M-1} P[x]T[i + x](P[x] - T[i + x])^2$$

This sum also follows the property that $S[i] = 0$ iff there is a matching at i. Similar to (1), FFT can be used to precompute all these terms in $\mathcal{O}(N \log N)$.

- Pick out the details of the FFT in (1) since the problem is relatively standard.
- Less detail needs to be given for (2), as long as they provide a correct construction.

Question 11 *Bracket Scores*

There are $2N$ numbers in an array $A[1 \dots 2N]$. You can create a valid bracket-matching sequence S of length $2N$. To determine the 'score' of the sequence, overlay this with the array A – if the i'th bracket is ")", set $A[i] = 0$; else, leave $A[i]$ as its original value. At the end, the score of the bracket sequence is the sum of modified array A. What is the max score achievable ($\mathcal{O}(N \log N)$)?

A valid bracket-matching sequence can be defined as:

(1) For all $1 \leq k \leq N$, among $S_1, \dots, S_{2k-1}$, at least k characters are "(".

(2) Among $S_1, \dots, S_{2N}$, exactly N characters are "(".

Using this idea, how can we greedily assign a sequence with maximum score?

We can assume (for simplicity without proof) that the definition of a bracket-matching sequence as given in the hint is sufficient and necessary to determine validity.

With this in mind, we can construct a greedy algorithm to solve the problem. Firstly, $S_1 =$"(", since otherwise condition (2) does not hold. Let's store the set of positions X which we can set as an open bracket. Then, for all $2 \leq k \leq N$, we will:

1. Add positions $2k - 2$ and $2k - 1$ into X.
2. Pick an element $x \in X$ such that $x = \text{argmax}_{i \in X} A_i$, and set S_x to open bracket.
3. Remove x from X.

Now we aim to prove this greedy method is valid via an exchange argument. Suppose that when $k = t$, we choose some non-maximal y ($A_y < A_x$). If x is never chosen afterwards, we can simply replace the open bracket at y with one at x. Then, the sum increases by $A_x - A_y > 0$, and so choosing y was suboptimal.

If, on the other hand, x was chosen at some $k = t' > t$, then we can swap our decisions, by choosing x at t and y at t'. This does not affect the final bracket sequence (and thus its

validity), but maintains the sum. Thus, the greedy algorithm is always optimal, since any solution can be transformed into the greedy one while not decreasing the score.

- The proof for the claim made in the hint can be found at https://atcoder.jp/contests/abc407/editorial/13107.
- Make sure people prove the greedy.

Question 12 *Boiling Bubbles*

As anyone who does physics can tell you, bubbles expand when heated or something like that. You have a large two dimensional pot which can be modelled as an infinite plane. There are N nucleation sites (places where bubbles can form) represented as points (x_i, y_i), and from each nucleation site a bubble can sprout – a circle of variable radius r_i with center (x_i, y_i). No two bubbles are allowed to intersect at more than one point. Design an average-case polynomial algorithm which determines the radii of the N bubbles such that the sum of perimeters of bubbles is maximised.

The sum of perimeters is

$$P = \sum_{i=1}^{N} 2\pi r_i = 2\pi \sum_{i=1}^{N} r_i$$

It is sufficient therefore to maximise the sum of radii.

The problem in essence reduces to maximising

$$r_1 + r_2 + \cdots + r_N$$

subject to the constraints that

$$r_i + r_j \leq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \qquad \text{for all } 1 \leq i < j \leq N$$

and

$$r_i \geq 0 \qquad \text{for all } 1 \leq i \leq N$$

This is clearly a linear programming problem; for revision purposes –

Standard Form: Let

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & 1 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2} \\ \vdots \\ \sqrt{(x_1 - x_N)^2 + (y_1 - y_N)^2} \\ \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2} \\ \vdots \\ \sqrt{(x_{N-1} - x_N)^2 + (y_{N-1} - y_N)^2} \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Then, we seek to maximise $\mathbf{c}^\intercal \mathbf{x}$ subject to $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$ where $\mathbf{0}$ is the zero-vector and

$$\mathbf{x} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix}$$

Since $\mathbf{A}$ is an $N^2 \times N$ matrix, $\mathbf{b}$ is an N^2 dimensional vector, and $\mathbf{c}$ and $\mathbf{x}$ are N dimensional vectors, the simplex algorithm will run in $\Theta(N^3)$ average case, and exponential time worst case.

Check that people can write LP problems in standard form.

Question 13 *Transportation Renovation*

There are N cities and $N-1$ bidirectional roads which connect them such that every city is reachable from any other. Each road has an intrinsic value m_i. The mayor has chosen K cities $h_1, h_2, \ldots, h_K$ to be special hubs, each with popularity c_i. Between any two hubs (i, j), a bidirectional train line can be built with cost $c_i + c_j$. After building train lines, roads can be closed, which earns the mayor the road's value, as long as all cities are still connected. Find the max amount of money the mayor can earn by building trains and closing roads in $\mathcal{O}(N \log N)$.

The most annoying part of the problem is that we have removals of roads (which increase our answer) together with additions of trains (which decrease our answer). How can we combat this?

Imagine the mayor destroys all roads to begin with, so that all cities are disconnected with no train lines having yet been built. By storing the cost, rather than the money earned, we can set our intial answer to be $-\sum_{i=1}^{N-1} m_i$. Adding m_i to our answer every time the i'th road is 'rebuilt', we guarantee the set of unbuilt roads S contributes $-\sum_{i \in S} m_i$ to our answer – exactly the desired contribution (unbuilt roads give back the money we 'spent').

Follow the problem simplification outlined in the hint. We can therefore think of the i'th road as having a cost of m_i to build, and the train line (i, j) having a cost of $c_i + c_j$ to build. Our aim is to select a subset of roads and train lines which connects all cities with minimal cost.

Recognise that this is essentially the **Minimum Spanning Tree** problem, which can be solved for graph $G = (V, E)$ in $\mathcal{O}(E \log E)$ via Kruskal's algorithm. However, if we explicitly enumerate all train lines, this comes out to $\mathcal{O}((K^2 + N) \log(K^2 + N))$, too big for our target complexity.

Instead, consider some moment in time where there are two connected components, each with some number of train hubs. If we are to ever use a train line to merge these two, it is optimal to pick the hubs with the smallest c_i from each connected component. Any other train line has a weakly higher cost.

Therefore, when merging components, we simply maintain the minimum hub cost for each component (this is done efficiently as part of Kruskal's algorithm via a DSU) and a min heap of all of these minimum hubs. Then, at any point in the MST algorithm, we have a choice of either:

a) the current minimum cost useful (connects diff CCs) road not yet built, or

b) the current minimum cost useful train line over *all* hubs.

Retrieving and managing a) requires sorting precomputation of edges in $\mathcal{O}(N \log N)$, and querying b), via our greedy observation, requires simply popping and summing the top two elements in our min heap before updating it with the smaller value, leaving $\mathcal{O}(\log N)$ per query (over $\mathcal{O}(N)$ queries). Therefore, the final time complexity is $\mathcal{O}(N \log N)$ as desired.

Be lenient (and give hints), since this problem is one of the harder ones in the set.

Question 14 *Battleships*

Battleships is a game played on a grid with R rows and C columns. There are ships of various sizes – or so you thought, because whatever ripoff copy you purchased has only 1x1 pieces. As a battleships fanatic, you know that the i'th piece has four constraints a_i, b_i, c_i, d_i, and strategically must be placed in an empty cell (x, y) such that $a_i \leq x \leq c_i$ and $b_i \leq y \leq d_i$. Furthermore, no two ships should occupy the same row or column. Construct a polynomial time algorithm to find the maximum number of pieces you can place on the grid simultaneously.

Battleships is a game which is set in an environment with lots of water.

Another equivalent formulation is: "There are N people and R drinks and C foods. Each person only wants a certain range of drinks, and a certain range of foods. Find an assignment of food + drink to each person to maximise people that can be served."

This is a flow problem. Construct a flow graph with the following nodes:

- 2 nodes S, T for source and target
- R nodes $\{R_1, R_2, \ldots, R_R\}$ representing rows
- C nodes $\{C_1, C_2, \ldots, C_C\}$ representing columns
- $2N$ nodes $\{U_1, U_2, \ldots, U_N, V_1, V_2, \ldots, V_N\}$ with (U_i, V_i) representing ship i

All edges added will have capacity 1. Prepare the graph for flow by adding for all $i \in [1, R]$, $S \to R_i$ and for all $j \in [1, C]$, $C_i \to T$. Then, add the following edges for all $i \in [1, N]$:

- $\{R_{a_i}, R_{a_i+1}, \ldots, R_{c_i}\} \to U_i$ (choosing a row)
- $U_i \to V_i$ (limiting to one row/column choice)
- $V_i \to \{C_{b_i}, C_{b_i+1}, \ldots, C_{d_i}\}$ (choosing a column)

The answer to the problem is the max flow from S to T. This can be calculated in $\mathcal{O}(E\sqrt{V})$ using Dinic's, where $V = R + C + N$ and $E = N(R + C)$. Alternatively, use Ford-Fulkerson to get $\mathcal{O}(\min(R, C, N) \times E)$ time. Either way, the solution time complexity is polynomial.

In order to justify the flow, people need to outline/explain/provide a mapping

- from every flow to some assignment
- from every assignment to some feasible flow