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

2026 Mini Competition 1 Editorial

2026-03-07

Statistics

Task	Attempts	Max	Mean	Std Dev	Subtasks
M2611 - Bowling Score Recovery II	57	100	43.28	26.654	11: 56 | 8: 54 | 14: 48 | 24: 15 | 27: 9 | 16: 9
M2612 - Grid Blockers	42	100	30.19	31.505	13: 24 | 21: 6 | 18: 27 | 24: 8 | 16: 7 | 8: 5
M2613 - Hidden Grid	48	100	63.687	35.552	8: 48 | 14: 40 | 24: 36 | 11: 34 | 15: 21 | 28: 20
M2614 - Sushi Merging	47	100	65.276	43.279	7: 41 | 12: 35 | 26: 32 | 22: 29 | 33: 27
	57	**400**	**172.982**	**102.613**	

General Tips

Standard I/O Task (e.g. Minicomp 1, Minicomp 2, EGOI TFT)

- Process input/output through standard I/O
- Doing I/O Optimization trick like `ios_base::sync_with_stdio(false)`, `cin.tie(nullptr)` would be useful to lower the runtime
- Understand the difference between `"\n"` (endline character) and `endl` (endline character, forces immediate flush, slower)
- For interactive task, you should flush the output after every interaction, i.e. `fflush(stdout)`

M2611
Bowling Score Recovery II

Subtask 1: N = 1, K < 10 / Subtask 2: N = 1

- There is only one frame.
- Notice that we cannot get more than 10 points as the “bonus next frame” does not exist
- Output `(K 0)` if K < 10
- Output `(X)` if K = 10
- Impossible otherwise

Subtask 3: K ≤ 9 * N

- We can get at most 9 * N points without strikes and spares
- Greedily output `(9 0)`, `(K 0)` or `(0 0)` and update K accordingly

Subtask 4: K ≤ 30 * N - 54

- K is “large” and we need strikes or spares
- Let’s try using strikes!
- M strikes: `(X)`
- 2 adjusting frames: `(A, 0) (B, 0)`

- Total points:
  30 * (M - 2) + (10 + 10 + A) + (10 + A + 0) + (A) + (B) = 30 * (M - 1) + 3 * A + B
- We can construct any number ≤ 36 with 3 * A + B
- As 36 = 3 * 30 - 54, we can adjust M, A, and B to construct K
- Remember to handle K ≤ 9 * N

Subtask 5: K ≥ 30 * N - 58

- Let's try using strikes again!
- M strikes: `(X)`
- 1 adjusting frame: `(A, B)`

- Total points:
  30 * (M - 2) + (10 + 10 + A) + (10 + A + B) + (A + B) = 30 * (M - 1) + 3 * A + 2 * B
- Similar to subtask 4, adjust M, A, and B to construct K
- If the calculated M > N - 1, output impossible
- Remember to handle K = 30 * (N - 1) ← all strikes

Subtask 6: No additional constraints

- Subtask 4: $K \leq 30 * N - 54$
- Subtask 5: $K \geq 30 * N - 58$
- Combine them to get Accepted!

M2612
Grid Blockers

The Problem

- Two groups of players start in distinct cells on an infinite 2D grid.
- Group 1 (Players **1** to N): Always attempt to move one cell Right ($+x$).
- Group 2 (Players $N+1$ to $N+M$): Always attempt to move one cell Up ($+y$).

- Movement Rules: Players move in turns: **1**, **2**, ..., $N+M$, then repeat.
- A player moves if the target cell is Empty. They then mark that cell with their ID.
- If the target cell is Occupied, the player's journey ends immediately.

- Goal: Answer Q queries about which player ID is written in cell (A_i, B_i).

Subtask 1 (13 points): N, M, Q, X, Y, A, B ≤ 20

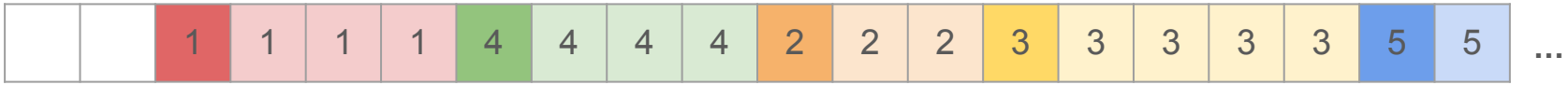
- Since coordinates and players are small (≤ 20), we can literally simulate the grid.
- Maintain a 2D array or a set of occupied coordinates.
- Iterate through turns and move players one by one.

Subtask 2 (21 points): N, M, Q ≤ 50

- Instead of step-by-step, only calculate Intersections.
- For each player, check all potential candidates in the opposing group that could block them.
- A player ***i*** (moving Right) and player ***j*** (moving Up) can only interfere at coordinate (X_j, Y_i).
- Distance for i to reach intersection: $d_i = X_j - X_i$
- Distance for j to reach intersection: $d_j = Y_i - Y_j$
- Player ***i*** marks the cell if and only if $d_i \leq d_j$.
- Simulate the grid from bottom left to top right to handle the blocking order
- Time Complexity: $O(N \times M)$ or $O((N + M)^2)$

Subtask 3 (18 points): M = 0, all Y are the same

- If ***M*** = 0, only Right-moving players exist.
- Players can only be blocked by the starting positions of other Right-moving players.
- Since all ***Y*** are the same, all players are in the same row.
- The ***Y***-th row of the grid looks like the following (sample 2):



- We can sort players by X_i. Use binary search to find the nearest player to the left or 0 if does not exist.

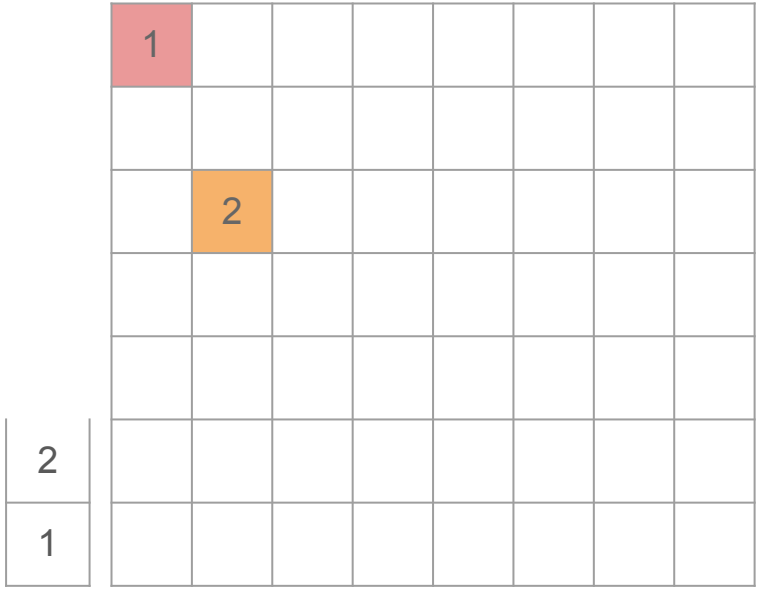
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- The grid looks like the following (sample 3):

- The most complicated area is the top right area where group 1 players and group 2 players intersect
- We can handle this area using a sweep line algorithm

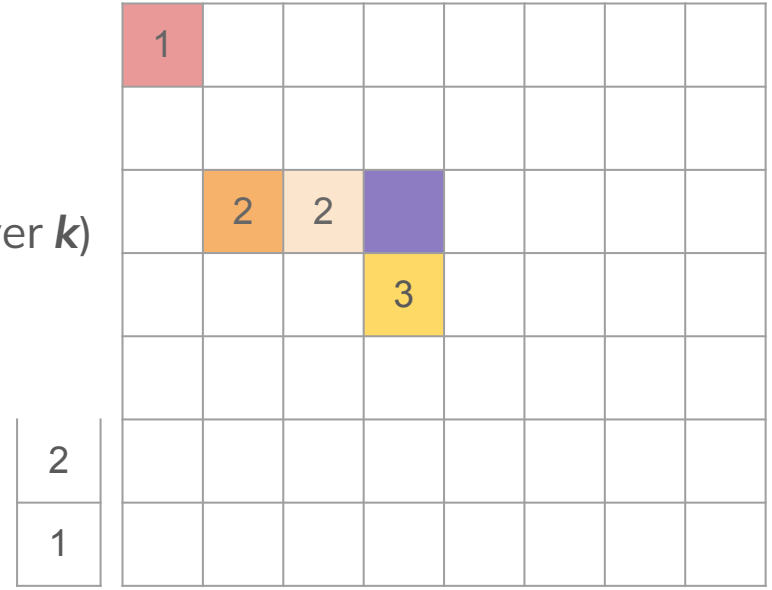
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack



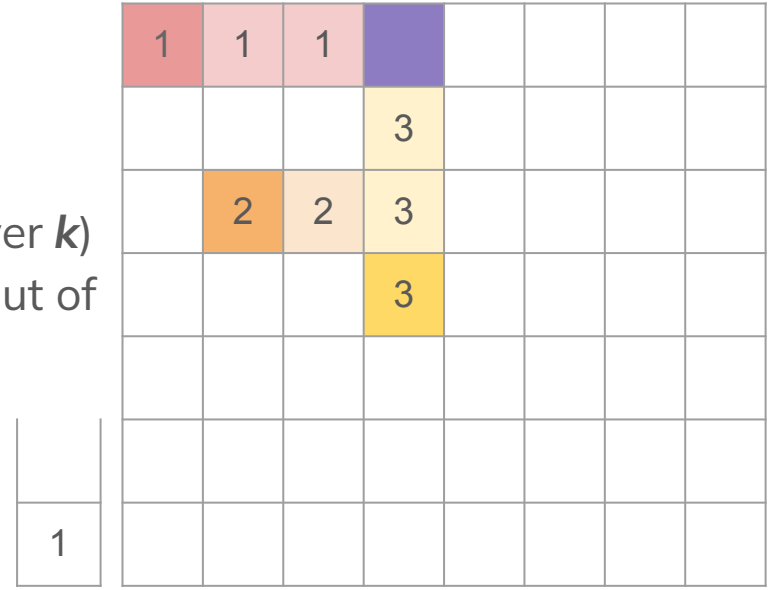
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack
- Then, process group 2 players one by one.
- We check if the group 2 player ***i*** blocks the player on the top of the stack (let it be player ***k***)



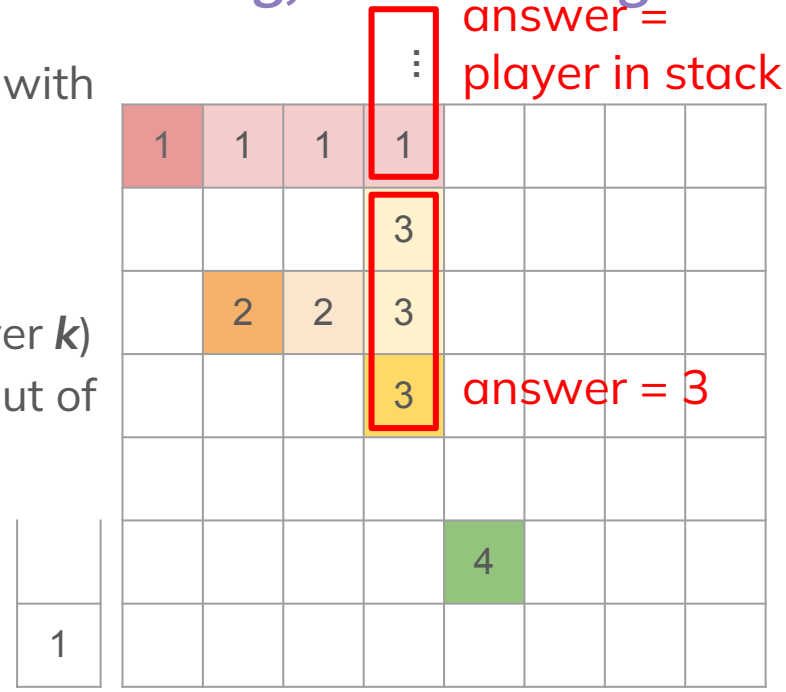
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack
- Then, process group 2 players one by one.
- We check if the group 2 player ***i*** blocks the player on the top of the stack (let it be player ***k***)
- If player ***i*** blocks the cell, we pop player ***k*** out of the stack and repeat the process



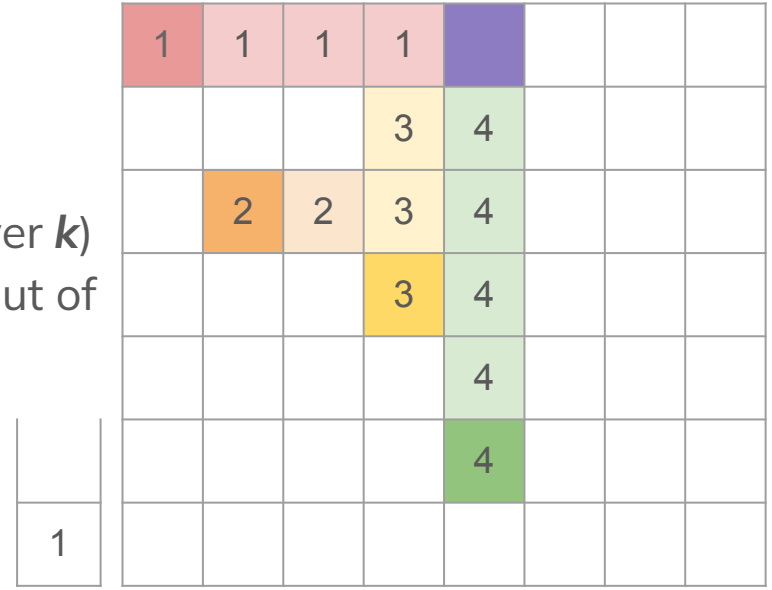
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack
- Then, process group 2 players one by one.
- We check if the group 2 player ***i*** blocks the player on the top of the stack (let it be player ***k***)
- If player ***i*** blocks the cell, we pop player ***k*** out of the stack and repeat the process
- Otherwise, we end the process and move on to player ***i*** + 1
- For each query ***j*** on this column, we can handle them by comparing B_j and Y_k



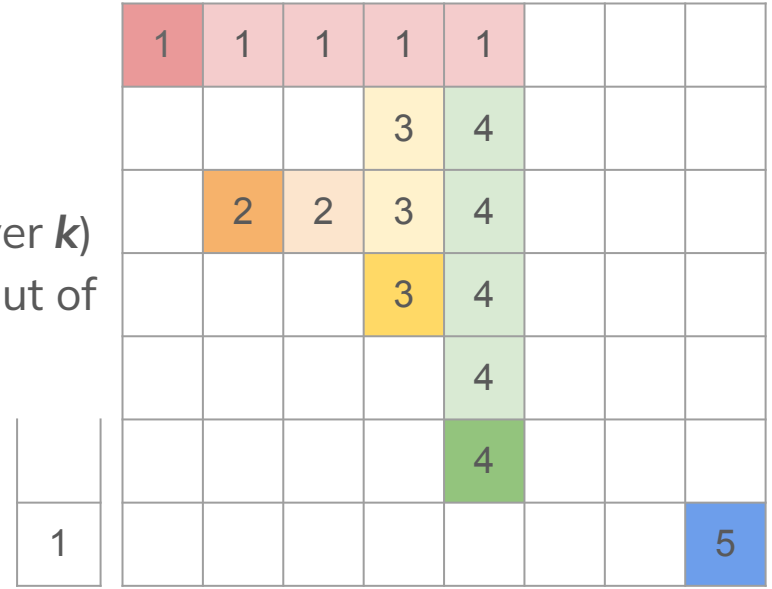
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack
- Then, process group 2 players one by one.
- We check if the group 2 player ***i*** blocks the player on the top of the stack (let it be player ***k***)
- If player ***i*** blocks the cell, we pop player ***k*** out of the stack and repeat the process
- Otherwise, we end the process and move on to player ***i*** + 1
- For each query ***j*** on this column, we can handle them by comparing B_j and Y_k



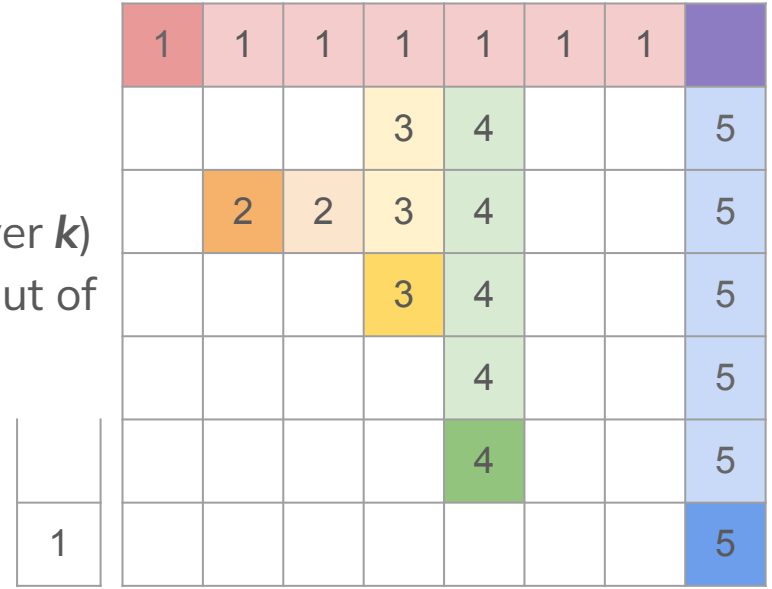
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack
- Then, process group 2 players one by one.
- We check if the group 2 player ***i*** blocks the player on the top of the stack (let it be player ***k***)
- If player ***i*** blocks the cell, we pop player ***k*** out of the stack and repeat the process
- Otherwise, we end the process and move on to player ***i*** + 1
- For each query ***j*** on this column, we can handle them by comparing B_j and Y_k



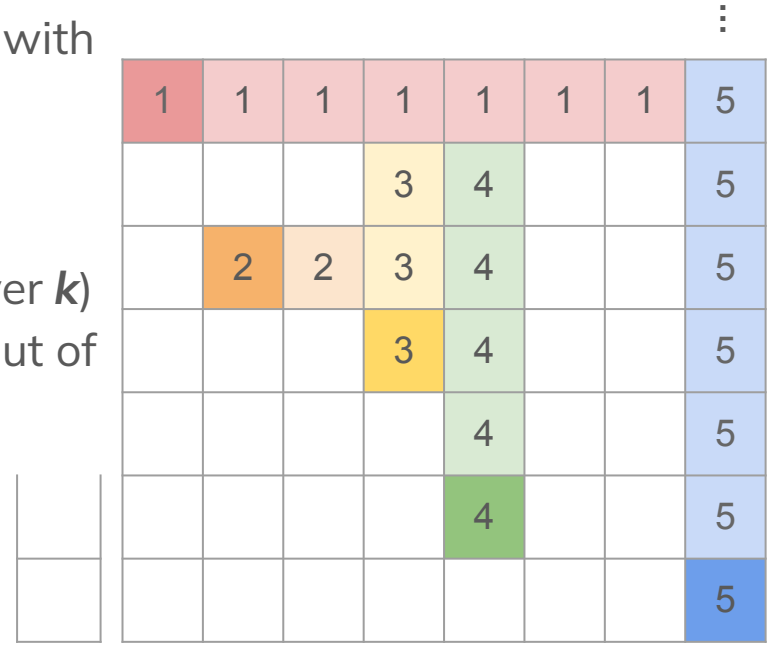
Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack
- Then, process group 2 players one by one.
- We check if the group 2 player ***i*** blocks the player on the top of the stack (let it be player ***k***)
- If player ***i*** blocks the cell, we pop player ***k*** out of the stack and repeat the process
- Otherwise, we end the process and move on to player ***i*** + 1
- For each query ***j*** on this column, we can handle them by comparing B_j and Y_k



Subtask 4 (24 points): N, M, Q ≤ 10^5, X increasing, Y decreasing

- First, push all group 1 players into a stack, with player ***N*** at the top of the stack
- Then, process group 2 players one by one.
- We check if the group 2 player ***i*** blocks the player on the top of the stack (let it be player ***k***)
- If player ***i*** blocks the cell, we pop player ***k*** out of the stack and repeat the process
- Otherwise, we end the process and move on to player ***i*** + 1
- For each query ***j*** on this column, we can handle them by comparing B_j and Y_k
- Special handle the case where the stack becomes empty



Subtask 5 - 6 (24 points): No additional constraints

- We can continue using our sweep line algorithm. However, we cannot just use a stack. We need to support the following operations:
- Inserting a group 1 player with arbitrary Y-coordinate
- Query the next group player with Y-coordinate greater than Y_i
- Remove a specific group 1 player
- Query if there is a group 1 player with Y-coordinate B_j

Subtask 5 - 6 (24 points): No additional constraints

- We can continue using our sweep line algorithm. However, we cannot just use a stack. We need to support the following operations:
- Inserting a group 1 player with arbitrary Y-coordinate — s.insert(x)
- Query the next group player with Y-coordinate greater than Y_i — s.upper_bound(x)
- Remove a specific group 1 player — s.erase(x)
- Query if there is a group 1 player with Y-coordinate B_j — s.find(x)

- We can use a std::set!
- Store the pair (Y_i, i) in the set to perform all operations

Subtask 5 - 6 (24 points): No additional constraints

- Time Complexity: **O((*N* + *M* + *Q*) log (*N* + *M* + *Q*))**
- Log factor comes from sorting before sweep line and std::set

Subtask 5 is designed for slightly incorrect or slower solutions (e.g. did not handle same X or Y coordinates correctly, or using segment tree with higher constant factor)

M2613
Hidden Grid

The Problem

- There is a hidden grid of integers A.
- You are given a grid of integers B, where B[i][j] is equal to the sum of elements in A that are not in the i-th row nor in the j-th column.
- Your task is to derive A from B.

Subtask 1 (8 points): N = M = 2

By the definition of B, we have:

- B[1][1] = A[2][2]
- B[2][1] = A[1][2]
- B[1][2] = A[2][1]
- B[2][2] = A[1][1]

Output A accordingly.

Time Complexity: O(1)

Expected Score: 8

Subtask 2 (14 points): N = 2, M = 3

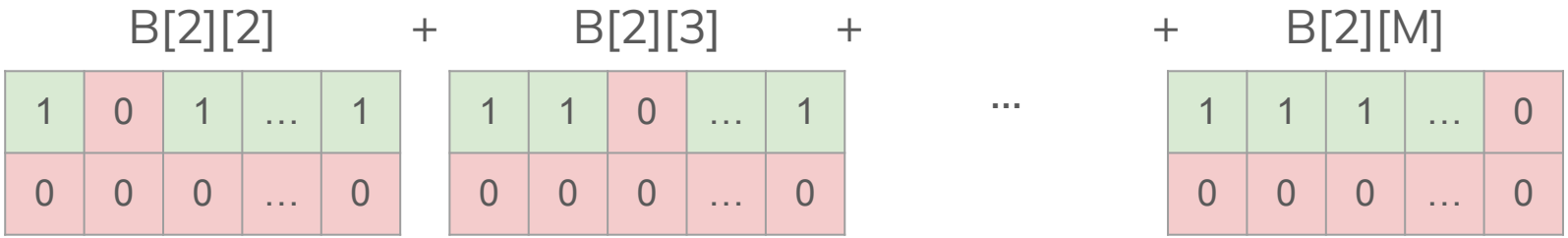
- Consider the first row of B, we have:
  - B[1][1] = A[2][2] + A[2][3]
  - B[1][2] = A[2][1] + A[2][3]
  - B[1][3] = A[2][1] + A[2][2]
- Then, A[2][1] = (B[1][2] + B[1][3] − B[1][1]) / 2.
- The formula for A[2][2] and A[2][3] can be derived in a similar way.
- To find the first row of A, consider the second row of B in a similar way.

Time Complexity: O(1)

Expected Score: 14

Subtask 3 (24 points): N = 2, M ≤ 100

- To find the i-th element in the first row of A (i.e. A[1][i]), consider the sum of elements in the second row of B except B[2][i].
- Observe that for each j ≠ i, A[1][j] contributes to the sum (M − 2) times, while A[1][i] contributes to the sum (M − 1) times.
- Note that the sum of all A[1][j] where j ≠ i is B[2][i].



Subtask 3 (24 points): N = 2, M ≤ 100

- So, A[1][i] can be found by first calculating the sum of elements in the second row of B except B[2][i], then subtracting (M – 2) × B[2][i] from the sum, and dividing the result by (M – 1).

B[2][2] + B[2][3] + ... + B[2][M] -

M - 1	M - 2	M - 2	...	M - 2
0	0	0	...	0

(M - 2) * B[2][1]

0	M - 2	M - 2	...	M - 2
0	0	0	...	0

=

M - 1	0	0	...	0
0	0	0	...	0

Subtask 3 (24 points): N = 2, M ≤ 100

- We can calculate the sum of elements in the second row of B except B[2][i] in O(M) time for each i.
- To find the second row of A, consider the first row of B in a similar way.

Time Complexity: $O(NM^2)$

Expected Score: 46

Subtask 4 (11 points): N = 2

- Note that we don't have to recalculate the sum of elements in the second row of B except B[2][i] for every i.
- Precompute the sum of elements in the second row of B, then we can obtain the *sum of elements in the second row of B except B[2][i]* in O(1) time by subtracting B[2][i] from the sum of elements in the second row of B.
- Do the same for the first row of B.

Time Complexity: O(NM)

Expected Score: 57

Subtask 5 (15 points): N = 3

- If we consider the first row of B and apply our solution for subtask 4, we obtain the values of (A[2][i] + A[3][i]) for all $1 \le i \le M$.
- Similarly, we obtain the values of (A[1][i] + A[3][i]) and (A[1][i] + A[2][i]) by considering the second and the third row of B respectively.
- This gives us a system of 3 equations in the form of that in subtask 2 for every column. Solve the system with the solution for subtask 2.

Time Complexity: O(M)

Expected Score: 15

Subtask 6 (28 points): No additional constraints

There are a few things that we want to do:

- Find the sum of elements in A, denoted as *sumA*.
- Find the sum of elements in the i-th row of A, denoted as *row[i]*.
- Find the sum of elements in the j-th column of A, denoted as *col[j]*.
- Derive a formula for A[i][j] in terms of *sum*, *row[i]*, *col[j]*, and a trivial number of elements in B.

Subtask 6 (28 points): No additional constraints

Finding the sum of elements in A

- Consider the sum of all elements in B.
- Note that every element in A contributes to the sum (N − 1)×(M − 1) times.
- This is because every A[i][j] affects the value of all elements in B except for the i-th row or the j-th column, so it affects (N − 1) rows and (M − 1) columns.
- So, *sumA* is equal to the sum of elements in B divided by (N − 1) × (M − 1).

Subtask 6 (28 points): No additional constraints

0	0	0	…	0
0	1	1	…	1
0	1	1	…	1
…	…	…	…	…
0	1	1	…	1

+

0	0	0	…	0
1	0	1	…	1
1	0	1	…	1
…	…	…	…	…
1	0	1	…	1

+ … +

1	1	1	…	0
1	1	1	…	0
1	1	1	…	0
…	…	…	…	…
0	0	0	…	0

=

(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)
(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)
(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)
(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)	(N-1)(M-1)

Subtask 6 (28 points): No additional constraints

Finding the sum of elements in the i-th row of A

- Consider the sum of all elements in the i-th row of B.
- Note that every element that are not in the i-th row of A contributes to the sum (M − 1) times.
- This is because every A[x][j] where x ≠ i affects the value of all elements in the i-th row of B except for the j-th column, so it affects (M − 1) elements in the i-th row of B.
- So, *row[i]* is equal to *sumA* minus the sum of elements in the i-th row of B divided by (M − 1).

Subtask 6 (28 points): No additional constraints

0	0	0	…	0
0	1	1	…	1
0	1	1	…	1
…	…	…	…	…
0	1	1	…	1

+

0	0	0	…	0
1	0	1	…	1
1	0	1	…	1
…	…	…	…	…
1	0	1	…	1

+ … +

0	0	0	…	0
1	1	1	…	0
1	1	1	…	0
…	…	…	…	…
1	1	1	…	0

=

0	0	…	0
M - 1	M - 1	…	M - 1
…	…	…	…
M - 1	M - 1	…	M - 1

Subtask 6 (28 points): No additional constraints

Finding the sum of elements in the j-th column of A

- Consider the sum of all elements in the j-th column of B.
- Note that every element that are not in the j-th column of A contributes to the sum (N – 1) times.
- This is because every A[i][y] where y ≠ j affects the value of all elements in the j-th column of B except for the i-th row, so it affects (N – 1) elements in the j-th column of B.
- So, *col[j]* is equal to *sumA* minus the sum of elements in the j-th column of B divided by (N – 1).

Subtask 6 (28 points): No additional constraints

0	0	0	…	0
0	1	1	…	1
0	1	1	…	1
…	…	…	…	…
0	1	1	…	1

+

0	1	1	…	1
0	0	0	…	0
0	1	1	…	1
…	…	…	…	…
0	1	1	…	1

+ … +

0	1	1	…	1
0	1	1	…	1
0	1	1	…	1
…	…	…	…	…
0	0	0	…	0

=

0	N - 1	…	N - 1
0	N - 1	…	N - 1
…	…	…	…
0	N - 1	…	N - 1

Subtask 6 (28 points): No additional constraints

Deriving a formula for A[i][j] in terms of *sumA*, *row[i]*, *col[j]*, and a trivial number of elements in B

- *row[i]* is the sum of elements in the i-th row of A
- *col[j]* is the sum of elements in the j-th column of A
- B[i][j] is the sum of elements that are not in the i-th row nor the j-th column of A.
- Summing them up, we have *row[i]* + *col[j]* + B[i][j] = *sumA* + A[i][j].
- Rearranging the terms, we have A[i][j] = *row[i]* + *col[j]* + B[i][j] − *sumA*.

Subtask 6 (28 points): No additional constraints

0	0	0	…	0
0	0	0	…	0
1	1	1	…	1
…	…	…	…	…
0	0	0	…	0

+

0	1	0	…	0
0	1	0	…	0
0	1	0	…	0
…	…	…	…	…
0	1	0	…	0

+

1	0	1	…	1
1	0	1	…	1
0	0	0	…	0
…	…	…	…	…
1	0	1	…	1

=

1	1	1	…	1
1	1	1	…	1
1	2	1	…	1
…	…	…	…	…
1	1	1	…	1

Subtask 6 (28 points): No additional constraints

Precompute the values of *sumA*, *row[i]* and *col[j]* for all $1 \le i \le N$ and $1 \le j \le M$.

We can then obtain the value of A[i][j] in O(1) time.

Time Complexity: O(NM)
Expected Score: 100

M2614
Sushi Merging

Subtask 1: $C_1 + C_2 + \ldots + C_N \leq M$

- Key observation: merging two dishes won't change the total tastiness on the table, but it saves one place.
- The table is big enough to hold all the sushi, the maximum total tastiness is simply the sum of all C_i.

Subtask 2: $N = 1, C_1 \leq 100$

- Key observation: merging two dishes won't change the total tastiness on the table, but it saves one place.
- Greedily merge dishes and pick up sushi whenever there is space.

Subtask 3: N = 1, C_1 is power of 2

- With the greedy strategy, building a level k dish requires k + 1 places on the table.
- If $M \geq \log_2 C_1 + 1$, the answer is C_1.
- Else the final state of levels of dishes on the table will be M - 1, M - 2, ... , 1.
- The answer is $\min(C_1, 2^M - 1)$.
- Reminder: you cannot directly calculate 2^M as it may exceed the range of int / long long / double. You should find $\log_2 C_1$ using the function __lg to compare instead.

Subtask 4: All C_i are power of 2s

- Sort C in descending order.
- Let C_1 be the largest element after sorting.
- If $M \geq \log_2 C_1 + 1$, we will build a dish of level $\log_2 C_1$ first.
- Since the number of places used is minimized at the end, we should finish building a dish of level $\log_2 C_1$ before building anything else.
- Repeat the procedure on C_2 with $M' = M - 1$ and so on until $M' < \log_2 C_k + 1$, then apply the solution of subtask 3.

Subtask 5: No additional constraints

- Split every C_i into $2^a + 2^b + \ldots$
- Each power of 2 can be treated as an individual type.
- Then, apply the solution of subtask 4. Remember that you still need to sort the power of 2s in descending order.