A: programmingteam, 2 + 4
B: flightplans, 7 + 17
C: entirelyunsortedsequences, 3 + 5
D: primalrepresentation, 12 + 19
E: disastrousdoubling, 11 + 19
F: eatingout, 14 + 19
G: babelfish, 14 + 20
H: brickwall, 8 + 14
I: wheretolive, 11 + 19
J: coupons, 1 + 0
K: pipes, 1 + 1
L: whiteboard, 2 + 3
M: rainfall, 1 + 3

Going through problems in descending order of solve counts:

G: babelfish, 14 + 20

Implement map

F: eatingout, 14 + 19

The menu has m items, and Alice, Bob, and Clara will order a, b, and c items respectively. Is it possible for them to pick some items such that no item is picked by everyone?

Check the 'bad' conditions: if any >= m, or if a + b + c > 2m
Think of the people not choose something

D: primalrepresentation, 12 + 19

Do prime factorization for things up to 2^{31}, take care of negatives too (by adding a -1)

How many primes are there up to 2^{16}: 7000 of them, somehow this passed.
Apparently even brute force check passed.

(need to mentally check: are these jerks asking me to code pollard rho....), this is a problem where you can wait, and let the scoreboard tell you whether you need pollard rho

I: wheretolive, 11 + 19

100 instances of minimize the point with total squared distance to everyone, and then round that point to an integer point

“If the best location is not exactly on a grid point, choose the grid point closest to the best location”

The dimensions double: this is 2 1-D problems.
You can solve each dimension separately, it is just minimizing a quadratic (turns out to be average of the points)

E: disastrousdoubling, 11 + 19

Simulate the process of
For i = 1...n
x[0] = 1
x[i + 1] = x[i] * 2 - b[i]
Output whether x[] ever gets negative, and x[n] % something

KEY: b[i] can have 60 bits, and n has 20 bits.

Difficulty of this problem is to fight the precision issue. This is scary because the max one can store in a 64-bit signed int is 2^{63} - 1.

Key observation is once you have more than 2^{61} bacteria, you will just not run out...

H: brickwall, 8 + 14

Given bricks of length 1, 2, 3, of up to 300 each
Check if one can arrange a subset of them (with a specified total) so that no prefix sum is in a forbidden subset.

Do dynamic program: state = # of bricks of size 1, 2, 3 that have been used, and just check if the sum is in the ‘bad’ set.

Naive: precompute lookup table for `bad’, then fill table, O(300^3) time, O(300^3) memory

??? (was not needed) Can get memory down to O(300^2) via lookup tables

??? ???????? can probably also push O(300^3 / 64) using bitset.

B: [flightplans](), 7 + 17

Interesting phenomenon: in afternoon session (https://napc21.kattis.com/sessions/ta8g26), B got solved by way more teams because it was solved early (26 vs. 59 (after 4 wrongs) in morning session, https://napc21.kattis.com/sessions/sbceig). Usually you want to check, ~ 180ish, whether any of the 'hard' problems are actually hard... because if they are easy, you can still solve them as if they are easy.

Do shortest path on a unit weighted graph where vertices' neighbors are listed either as neighbors list, or complement(neighbors).

Simulate BFS:
* neighbors: remove all neighbors from unvisited vertices.
* complement(neighbors): unvisited becomes unvisited intersect complement(neighbors), and add unvisited \ complement(neighbors) into the queue.

C: entirelyunsortedsequences, 3 + 5

Given n (n <= 5000) numbers, count the # of permutations where every number has either: a predecessor that's more, or a successor that's less.

(some thoughts by Richard) Assume all distinct:

    Think about what happens to the max:

        Say n is placed at location i, then locations i + 1....n can be w/e

        DP state: DP[x][k]: I'm placing 1...x, considering x, & there are k `free' slots

    Transition:

        [x][k] --> [x - 1][k - 1]

            --> [x - 1][k + w] (for all w, as long as you don't run out of free slots)

For each i, count the # of sequences where i is the first sorted elements

    Any sorted element is equal to its index (up to some ties)

    ans[i] = ans[i - 1] * (# of possible arrangements after i).

    Use multinomial coefficients to count # of arrangements after i (to handle duplicate items)

A: programmingteam, 2 + 4

Given tree on n vertices, pick a connected chunk including root with size k that maximizes (sum of a[i])/(sum of b[i]).

n, k <= 2500, a[i] b[i] positive.

1. Turn it into single value via binary search, aka. Search if i can make sum_{i picked} a[i] - \lambda * b[i] positive.

So you create c[i] <-- a[i] - \lambda * b[i], and check if there is a set of k vertices where c[i] sums to >= 0.

2. Min convolution on a tree (DP[u][x] = max_{y_1} DP[v][y_1] + DP[w][y_2] for v & w children of u) takes O(n^2) time (if y_1 <= size[v], y_2 <= size[w]).

Rather messy proof of this is in Lemma 5.5. of
https://www.cc.gatech.edu/~rpeng/CS4540_F20/Sep14TreeDP.pdf

More tree convolution DP that looks like O(N^3) but is actually O(N^2)
https://atcoder.jp/contests/abc207/tasks/abc207_f
https://open.kattis.com/problems/kthsubtree
https://codeforces.com/problemset/problem/1280/D

J: coupons, 1 + 0

Can prove:
1st ticket bought goes as long as possible til it expires
Won't buy another ticket until this one expires.

Zac's code: https://pastebin.com/zKsXVx0k
Works because the # of distinct stations is so small (<= 11). With more stations, we would need to use data structures.
(A counterexample was found!

```
6
2 0
5 5000
2 5001
0 9999
2 10000
5 15000

The answer should be 9, (but this solution outputs 11):
Get ticket 1 at time 0 for zones [0, 2] for $4.
Get ticket 2 at time 5000 for zones [2, 5] for $5.

Get and use ticket 1 at time 0 to go from zone 0 to zone 2.
Get and use ticket 2 at time 5000 to go from zone 2 to zone 5 and (at time 5001) back to zone 2.
Use ticket 1 at time 9999 to go from zone 2 to zone 0 and (at time = 10000) to zone 2.
Use ticket 2 at time 15000 to go from zone 2 to zone 5.
```

L: whiteboard, 2 + 3

M: rainfall, 1 + 3

K: pipes, 1 + 1