NOI Team Round 2017 Reflections

```
char author[] = "Carl Joshua Quines";
char date[] = "Sun May 07 01:35:20 2017";
```

At this point (the time of release), it is well-known that Robin and I took the NOI Team Round a week earlier, hence the earlier date of writing this. Yep, I'm writing this the evening of the round. I will try to describe my thought process on solving the problems, in the event someone wants to read how ~~a total failure~~ I think through a contest. Same disclaimer: some details might be misplaced temporally.

Pre-round

On the morning of May 6, I had a session at UPD for MOSC, our selection camp for the IMO. In fact, I had an afternoon session as well, but I did not attend because of the round. The morning session was supposed to last until noon, we had a quiz for the last hour and I pass an incomplete solution early so I could leave for ADMU.

I left UPD at a quarter to noon, and arrive at ADMU a bit before noon. We were supposed to start at noon, but I haven't eaten lunch yet and I couldn't find Faura Hall. I go to Gonzaga, grab some food, and afterward looked at a map. I then realized that Faura Hall was very near Gonzaga, so I went.

I arrive and Robin is already there, in front of the room. We make conversation about me coming from UPD and his multitudes of IB exams. Vernon arrives shortly and lets us in the room, telling us to wait as he looks for Kevin. I eat lunch. Robin and I talk about the NOI training, and he observed that the URL for the contest was a... cryptographic hash?

A few minutes pass and Kevin comes in. He sets the contest to start at half past noon. We register, and I open up Notepad++. I then realize that the laptop had Sublime Text, hooray. I embiggen the font size and set up a build system. I also ask if viewing `cppreference` was allowed, and it was. The contest starts shortly and the silence starts.

Start of round

Five problems, six hours. I was setting up the text editor as the round started, as if a minute or two would make a difference. I then read each of the problems. I instantly spot the reference to Guardians of the Galaxy and Lito Lapid, but nothing else.

I grab some paper and write down the first algorithm I could think of for each problem. Guardians felt doable with DP, though I was not exactly sure how to. It did seem similar to other questions I've seen before. Agadoo was several ranged sum updates followed by kth minimum queries, so the first thought I had in mind was a segment tree, followed by a difference array. I decided to debate which one to use later.

Spreadsheets was a game of nim. With two-dimensions. On a non-rectangular connected portion of the two-dimensional plane. And the squares have weights, and

one player wants to maximize, and the other wishes to minimize. Again, this seemed doable with DP, and the test cases are well-spread-out, so I had a chance at some number of points.

I tried to read City Plan, but was put-off by the oddly specific test case information. $0.1303N^3$? Really, Kevin? I thought of doing Floyd-Warshall's since N was small, but didn't spend much time thinking of it. It seemed like this would be the "odd one out problem", something I picked from math – the one problem you put off for last because it is the one that seems most intimidating.[1]

Lapida was an interesting problem, ranged queries on a graph. It felt like MSTs, but I was unsure. I was reminded of Kruskalblocked.[2] That was only four months ago, and I did not know how to do union-find at the time. In either case, it seemed like the MSTs depending on the choice of edges.

Guardians

I decided to take a few minutes writing out stuff on paper. I worked out a recursion for Guardians, which takes me some time to think about and check if it works. At first, I was having some trouble picking what the state should be and what information should be included about it. The key idea, which is the same key idea of a lot of DP problems, is that newly appended stuff to the end do not affect earlier appended stuff.

This appeared recently in our training at MOSC. Suppose we have a $1 \times n$ array of yellow and green squares. To each yellow square, assign two points, and three additional points per each adjacent green square. What is the maximum number of points? There is an algorithmic solution, arguing we can increase the points by replacing two adjacent yellow squares, and so on. But recursion is nice: each added square at the end only makes effects locally, around the one or two squares before it. Thus we can recurse per case.

It was the same idea in Guardians if you use DP. Observe that we have to end with a guard ending at the last square. If we're placing a guard ending at square i, we need to consider where the previous guard was placed. It could be placed ending at $i-1$, $i-2$, and so on, until $i-k$. It could not be placed back further because then there'd be an uncovered square. We need only to take the minimum of these and add the sum underneath the new guard.

This gives you a $O(Pk)$ time algorithm, $O(P)$ entries and $O(k)$ for the minimum operation, which was enough for all subtasks except the last. The last subtask had k on the order of P, and $O(P^2)$ is too slow. I reasoned an $O(P)$ or $O(P \lg P)$ solution must exist, and spend a few minutes trying to think of one, but fail. I decided early on that my strategy would be to get lots of partials, as I failed to AC any of the finals tasks the previous year.

Programming the code was straightforward and there were no major issues. I spend a few minutes trying some test cases by hand to check if my logic was correct, a tactic I learned from doing CF rounds.[3] It seemed to give the same answer when done by hand. I submit it and get all except the last subtask, as desired.

[1] The logic behind this is simple: you are trying to maximize points. Often it is better to devote more thought to getting more points a problem you are making progress on than start a new train of thought.

[2] Cf. Kapuluan ng Kalayaan, NOI 2017 Reflections.

[3] Cf. CF789B Masha and Geometric Progression.

Agadoo

I take a brief look at the scoreboard, which only had me and Robin, which felt odd. It was just me and him, no one else to compare to. I thought this was nice because there was less people to compare too, but then again, it's deprecating to compare myself to Robin. I move to the next problem.

Agadoo was ranged sum updates followed by kth element queries. Since we could do all the ranged updates before each query, we could take advantage of this and simply avoid optimizing for kth element queries until after applying each ranged update. Now how does one apply a lot of ranged sum updates quickly?

This is when, fortuitously, I remember a trick I read somewhere. A naïve ranged sum update takes $O(N)$, using a segtree takes $O(\lg N)$, but it is possible to do it in $O(1)$. The key idea is to take the difference array: instead of adding a sum to each element, add the sum to the start of the range and subtract the sum from the end of the range. This makes updating constant time, a total of $O(k)$.

Next were the problem of queries. I tried to think of a way to find the kth smallest element without generating the whole array, but I could not. Generating the whole array made my algorithm $O(N)$, which would not be tractable at all for the last two subtasks. I spend a few minutes on paper.

The idea I got was to take the reverse approach – rather than the perspective of each individual element, we take the perspective of the updates.[4] This would remove the $O(N)$ factor. I thought of making a list of point updates on the difference array and then compressing the indices to retain only where there is a change, but I did not have enough faith in myself to implement this.

This was when I decided to fold. I had to fold to 55 points instead of the whole problem. It was when I started writing the code when I realized how *devious* this whole problem was. The choice of non-explicit range updates, the use of a circular array, the problem of wrapping around, possibly multiple times, and the issue of starting from the pineapple disintegrator. There were at least a dozen opportunities for OBOEs.[5]

I managed to code up my solution, which was indeed nontrivial. It works for the two or three test cases I think of at the moment. I submit it, and it got... no points? I checked my code again, tried some test cases by hand, tried a lot of test cases by hand, before spotting the error. It was an indexing error for wrapping around. I brute forced more cases and submitted again, and it got...

Eight points!? Surprising. I expected 47 more. I went bug-hunting, and noticed that I only passed the first test case, which had small updates, while the full problem had updates up to 10^{10}. Perhaps an overflow? I changed all the variables to `long long`s, being reminded by a similar mistake I made in a CF round. It did not help.

I began to have a slight panic. I went out to the restroom to calm down and I drank some water. That helped. I came back and examined my code. What was the error? I tentatively ruled out a logical error – it seemed unlikely that it could pass for small updates but not large updates if the logic was wrong. But it still felt like an error with the program. I spend a couple more minutes trying test cases and move on, telling myself I'd return later.

[4] This is what I did last year too! Cf. Jabahw.

[5] OBOE: off-by-one error

Spreadsheets

The next problem was some sort of game problem, so the first instinct was automatically DP. It was a nim-type game on a two-dimensional grid, not entirely unlike what one would encounter as a math competitor. There is an additional restriction of each player wanting to maximize or minimize the value of the number in the last square of the grid, which complicates things. If this were a math problem, I would have spent time solving it. However, this was a programming problem, so I used DP.

I did not bother calculating the space constraints. The problem can be reduced to each player retaining a rectangular section from any of the corners, so you need four variables for storing the current state of the board, as well as an additional variable for storing which player is to move next. I was not entirely sure of how to do this; however I have coded something similar for our weekly training for the NOI, so I was not entirely unfamiliar. I just dived right in.

The first problem to deal with was the irregular shape of the board. I chose to dealt with this by storing zeroes in the parts of the board that weren't used. Each move has the additional restraint of keeping at least one positive number left in the rectangle. Then came the problem of the base case. For nonzero numbers, this was easy, it was the zeroes that had me thinking.

I ended up testing the code thrice, trying whether zeroes should be INF or 0, and for which player. After debugging the sample test cases for several minutes, and wading through a hundred kilobytes of debug output, I ended up using a sentinel value of -1 instead and adding an additional check in the recursion. That worked.

Then I tried looking at the bounds. It would be MLE for the full DP in the last subtasks. It would also be TLE for the last subtask. It felt like the DP was at least $O(N^4)$. The problem seemed to be both MLE and TLE, so it was clear that DP was not the intended solution for the harder tasks. There was probably a clever solution.

Not that I could think of anyway, and sunk cost fallacy was getting to me. I wanted to submit the code I spent a lot of time writing. So I did submit the code, it got up to the test case with $N \leq 12$, that was enough. I also look at the scoreboard, and Robin is already mounting a lead over me, which I anticipated to grow as the round progresses.

I tried to debug Agadoo. I had to clutch points; that was a 47 point waste looking for a fix. I went to work the only way I knew how: try a lot of test cases and hope I find the error by then. So for Agadoo, I tried a bunch of test cases. And they were mostly small cases, and I was focusing on how the index wrapped around.

I forgot how I thought of it, but out of a whim I changed one of the numbers to be on the order of 10^9. And then it clicked! The only thing that mattered was the relative sizes of the numbers; not the numbers themselves. If the same number was added to everything, that had no effect on the answer. I commented out the two lines in my code that did this, submitted, and got... 55 points. *Clutch!*

After succeeding on debuging my solution to Agadoo, I ate some food. I didn't notice there was food until I looked behind me, and then I noticed that Kevin made an announcement on Hackerrank. Whoops. It looks like Robin hasn't noticed either, and he looks like he's intensely coding.

Lapida

I already decided earlier that I would not attempt City Plan, so I moved to Lapida, the last problem of the set. I spent a good number of minutes trying sample test cases on a sheet of paper and trying to think of an algorithm that worked. Not a good algorithm; there was no time for that. I merely decided to just think of the easiest to code algorithm that would get some points and try that.

After a solid half hour of trying, I confirmed my intuition that this was, in essence, a ranged MST query. I'm slow, right? The first algorithm I thought of was to determine the closing range for each possible smallest edge weight of the MST, which was $O(E^2)$, slow. Then I realized it would be easier to do backwards, so you can do it with a single MST, which was $O(E \lg E)$. The problem was I had no idea how to implement this.

So I decided to stick with that very slow $O(E^2)$ algorithm. I didn't even check which subcase it would get, I just know that, so far, all my algorithms were quite naïve and yet got some number of points, so anything was better than nothing at this point. The round was closing and I was clutching.

It turns out my "easy to code" solution was not that easy to code. In particular, I messed up the UFDS the first time, and then I messed up Kruskal's (but I fixed it!) and then I messed up I/O, and then it was RTEing for no good reason. Then it turned out I was doing an out-of-bounds indexing on some array, which I fixed, and then it was WAing the samples! Then I realized it was an OBOE at the ranges, and I fixed it, and it worked.

It worked on the samples, that is. As soon as I submitted, it was very quickly judged – zero points. I didn't get any test case correct except the sample case. That sucked, I tried to look at my code. It was probably a logical error, I thought, since it only passed the samples. I tried to do a bunch of `assert`s to check if my logic was right.

True story: I was trying to verify that the largest edge in the MSTs were monovariant nondecreasing. So I did an `assert`, and it passed the sample case. Then I tried submitting, and it RTEd all the other cases. I then doubted the fact that the MSTs' largest edges were monovariant nondecreasing, even though they should if my logic holds any water.

I looked at my code and tried to check if there were any logical errors, and I couldn't find any. I double checked for typos and OBOEs, but the problem was, it was passing the sample but not the actual, and the sample itself was not straightforward to solve. Thus the problem was probably a technical issue. But I couldn't figure out what it was – there wasn't any room for overflow, I checked the edge cases, nothing.

Then I sort of resigned, as there were only a few minutes left. The only way to easily get points would be to debug Lapida. But I spent the rest of the time examining my code, trying to check if there were any mistakes. I tried test case out of test case, and did several cases by hand, but to no avail. I could not spot the error. My code's output matched my expected output, but it still got WA.

Post-round

The round ended briefly after that, and as the last second ticked, Kevin and Vernon began to applaud. Robin and I followed suit, even though there were only four of us in the room. Programming for six hours straight is no kidding. Robin noticed the food at the table, and was pleasantly surprised. It looked like he wanted to continue programming, and so did I – I just got an idea for Guardians. But the time was up.

Kevin debriefed us on the round. He said he didn't want to discuss the solutions to the problems one-by-one for one reason or another, but he still discussed some of the specifics anyway. He remarks on the difficulty of the problems, on how for him, Guardians was the easiest, followed by Lapida and Agadoo. This puzzled Vernon, who placed Lapida on the harder end, and found Spreadsheets easier. It was generally agreed that Guardians was on the easy side.

Kevin remarked on how it was good I managed to clutch the points for Agadoo in the last few minutes. He also asked me if I knew what my mistake was for Lapida. I replied I did not, and he revealed that I wasn't resetting my variables between test cases. The revelation sank in like Hiroshima and Nagasaki. That was 23 points down the drain simply because I forgot to reset my variables.

Kevin remarked that a simple solution to Spreadsheets existed. "*'Pag nalaman mo, mararamdaman mo yung parang, 'ay, oo nga no.' Pero hindi mo mararamdaman 'yung, 'ay, bakit hindi ko iyon naisip'.*" Kevin also said that a simple construction for City Plan existed. Robin said he wanted to try a randomized algorithm, but was worried about the edge cases, Kevin rebuked him.

Kevin[6] then reveals that the last problem was similar to a problem from the last ICPC–Manila.[7] I was startled at this, since I've briefly read the ICPC–Manila problems from last year. I looked the problem up and it was indeed quite similar – both involved how the MST was affected with edge deletions.

We were then asked not to discuss the details of the round with anyone else, for obvious reasons. Robin asked who did the illustrations, and it turns out they're done by a friend of Kevin. Robin then remarked on the flavor for the problems, and the question turned to Agadoo. It turns out that Agadoo is not arbitrary – apparently it is the title of a song which has lyrics like "push pineapple / shake the tree". Kevin searches it up for us and Robin and I get cancer from listening. Jared outcultures us once again!

The other problems had relatively simple references. Robin remarked that he once thought that the same person wrote the problems and thought of the stories, which would be quite cool. Kevin assented, and remarked that would indeed be quite cool – to be good in coming up with problems and to be cultured enough to sneak in plenty of references. But well, both acts individually are still quite cool.

Various other topics were discussed, and to go over all of them in detail would fill this article up. A brief survey:

- The metagame for competitive programming is discussed. In particular, how Russians and the Chinese were so pro, and how they have their own literature and everything. Implications were discussed.

- Metagame within a contest is also discussed. Kevin uses his second place finish in GCJ as an example. He explains his goal was not to get first place and beat Gennady, but to get second place. He talks about how he decided to code an easy solution to the small input of the last problem instead of trying to risk a solution for the large input, which turned out to be the right thing to do.

- Vernon and Kevin shared stories of ICPC. How it was different from individual contests, how teamwork played a big role. Personal experiences were shared:

[6] Observe that the last four paragraphs all began with the word "Kevin". This is a conspiracy.
[7] Cf. LA7856 Frog Pushers.

Kevin talked about how he and his teammates would practice weekly and the rules they had; Vernon remarked "*di kami nakapagqualify sa* ICPC *dahil [kina Kevin] eh.*"

- A conversation with Robin. Me: "Where are you going after thi–". Robin: "NUS.[8]" "No I mean, tonight." "Oh." Robin is still undecided between UPD, ADMU and NUS. He has doubts about going abroad – "I could die there".
- Kevin shares stories of his college life. Apparently he finished his coursework in three years, since he intentionally overloaded. However, it took him another two years to finish his thesis. "*'Di bale, dalawa pang taon para sa* ICPC." He does not recommend this however, and encourages us not to go down this path.
- The state of computer science education in the Philippines is the subject of mutual lament. Unsurprising, given its dismal condition. It was depressing how similarly horrid everyone's experiences were, from learning programming by merely copying code; in worse cases, not even learning programming at all; to having a CS course focused only on stuff like the parts of a Von Neumann machine to the difference between DRAM and SRAM.

After one or two hours of conversation, we part ways. Robin went to UPTC and I go with him. Along the way, he says he read my reflections on the NOI Finals, which was of course embarrassing. He then explains to me his solution behind Agadoo, which was similar to one of the solutions I thought of but did not try typing up. He stays at UPTC and eats with his family, and I go home.

Hindsight

Now for the reflective portion of the article. We deal with the good things before the bad things.

1. I had performed much better than I did last year. The me from a year ago would have, perhaps, gotten around fifty points. Also, the me from a few months ago would have gotten maybe a hundred points. This was a reflection of the amount of training I put in studying content; I was particularly pleased for recognizing two problems as doable slowly in DP and using a difference array. Also, I did not get Kruskalblocked!
2. It felt much more natural figuring out algorithms. When I did the finals back in January, I remembered thinking a lot about The Chenes of the Chorva, without finding a slow solution. The equivalent here would be figuring out the slow solutions to most problems, which was better than January-me would have done.
3. Near miss tactical error: 48 point loss in Agadoo. I managed to spot an obvious optimization this time, which was a correct decision, as it was the best points-to-effort ratio at the time. I managed to debug correctly for Spreadsheet as well, making it not as bad as Kruskalblocked.

[8] NUS: National University of Singapore

4. I managed to control my emotions quite well this time. This is probably because I was chill the morning before the round from MOSC and chatting with Robin, and probably because I couldn't compare to anyone except Robin. Notwithstanding, I should work on this even more – on at least one occasion during the round, I got quite flustered during debugging, which slowed me down.

I made plenty of mistakes in hindsight. Here are the strategic errors I committed:

1. Throughout the contest, I stuck with the first algorithm I thought of that satisfied me. This was the net-costliest strategic error; I would've scored more if I considered the bounds in each problem, thought of a possible complexity, and see if that was possible.

   An example. A friend told me the other day that $N \sim 300K$ was used to TLE $O(N \lg N)$ solutions, thus anything a bit less than that could potentially be doable for $O(N \lg N)$. Upon seeing $P \leq 100K$ in Guardians, I have thought of looking for an $O(N \lg N)$ solution, and I should have continued. Five minutes before the round ended, I thought this:

   It was obvious that one needed to look at each element of the array, so that was $O(N)$ down. What operations involve a factor of $\lg N$? Sorting, segment trees, divide-and-conquer. The extra $O(k)$ unwanted factor was coming from what was essentially a minimum query, which is doable in $O(\lg N)$ insertion and deletion and $O(1)$ query time with `std::map`. This would have satisfied the bounds.[9]

2. As a more general strategic error, I have chosen the "get lots of partials" path rather than "AC select problems then scrape" path. Indeed, if I had done the latter, I could have scored nearer to Robin, and not waste time scrapping partials on Lapida. That would have brought me from 0.65 Robins to 0.9 Robins.

3. Recall that Kevin mentioned post-contest the similarity of Lapida and a problem from ICPC–Manila. I have, in fact, read the problems of ICPC–Manila as well as read the editorials. However, I clearly did not manage to internalize the solutions! I suppose it is a training-failure-mode kind of error, in which I had the opportunity to learn but did not use it.

And here were the tactical errors I made:

1. I instinctively avoided City Plan despite not having read the problem description well. If I read the problem, thought about it for five or so minutes, and drew graphs on paper, I would have realized that this was more mathematical than algorithmic. I could have gotten around 55 points. What I found intimidating was the obscure scoring and the off-putting bounds.

   In this case, I should have tried the problem – a similar problem last year was confusing and had off-putting bounds, and it turned out to be precomputation.[10] These are ad-hoc style problems which I usually have an advantage at, and was probably my costliest tactical error.

[9] I am aware asymptotically faster solutions exist, in particular the $O(N)$ solution of using an `std::deque`. But these were my actual thoughts *five minutes* before the round ended. Sad.

[10] Cf. Zognoid Eggs.

2. The tactical error I can assign a real cost to, however, is Lapida. I have missed 23 points because I forgot to reset my variables in between cases. It was not as bad as my 100 point loss in Kruskalblocked. I seem to have a long list of these kinds of tactical errors: whitespace, missing to reset variables, forgetting corner cases, OBOEs, accessing out-of-bounds elements.

3. Corner-cutting. For Spreadsheets, I should have used a `#define` to make the eight possible cases nicer and easier to write and edit; small tactical error that has costed me several minutes. I also spent too much time verifying my intuition instead of placing my fingers on the keys and type-type-typing. There was plenty of room for corner-cutting and I should've done so.

To drive home the magnitude of my folly, it is to be noted that I wasted about a hundred points due to the errors listed above. A hundred points is a non-trivial amount when you're up against people who score nearly the same as you.

Comparisons

Now I'll draw parallels between competitive math and competitive programming, many of which I realized because of that night. Warning: largely opinionated. I'm interested in hearing your opinion on these:

- *Meta.* To improve one's craft is the result of knowing *how* to improve it. Therefore reflection, deliberate practice, and meta-discussion are important.

- *Structure.* Often in mathematics a problem can be separated in two parts: the "big idea" and the "technicalities". Evan Chen expresses a similar sentiment in a post of his,[11] which I think was expressed quite well. For example, the big idea for Agadoo was "take the perspective of the queries, not the array," after which the main difficulty is the technicalities. In this case, Agadoo is like a functional equation, in which the technicalities play a big role.

  The implication of this is that in training, focus must be given on both parts. Technicalities are stuff like learning how to use segment trees, Dijkstra's algorithm, or doing least common ancestor; learning these are completely acceptable. But the other part, the "big ideas", I suppose, are harder to learn, are often more important, and can usually only be picked up through practice.

  Not just spamming problems, but learning to connect the dots of the problems you've done and find patterns. I have a small list in my notes. Under "divide-and-conquer", for example, I have IOI 2016's Alien. Under "take a different perspective", there's Jabahw and Agadoo. I'm sure other people have other stuff they've learned, but this is the type of thing that's harder to teach, and can be picked up through practice – *doing lots of problems*, as is common advice.

- *Training.* On the topic of training, as above, there are parallels with math. The common advice in training for competition math is "do problems slightly above your level". Apply EMH: if there were an easy way to excel in competition math, everyone would be doing it, and it would lose its effectiveness. Competitive anything is zero-sum, therefore it takes hard work to beat the market.

[11] Cf. On Reading Solutions, Some Thoughts on Olympiad Material Design.

There is a correct way to work hard, however. Doing problems you know how to solve is okay for training on the technicalities, which is more important for programming than it is for math.[12] Like math, when it comes to algorithmic training, it's about doing problems which are neither too easy nor too hard – there should be a level of suspense, when you look at the problem, you are unsure whether you are able to solve it or not.

I suppose this is something I should apply. I am slowly going out of the phase where I need to learn technicalities (though there are still some technicalities I need to learn, like a bug-free BBST, writing segtrees quickly, *network flow* dear goodness). I observed that my limiting point at the moment is algorithmic skill – learning the ideas, the heuristics. I suppose I need to focus training on this, spending at least an hour working on figuring an algorithm before giving up.

- *Contest meta: problem order.* In math, a two-day olympiad consists of six problems, with three problems for four and a half hours each day. There are different approaches in dealing with this. One approach that *does not* work is to flitter between each problem, making little progress on each one, and writing up lots of scattered, disorganized observations. This is a sure way to score low.

  Instead, what most people do is to do one problem at a time. Problems are often sorted in order; thus the general rule is to tackle the first problem first before second and third. It is preferred to solve one problem fully than two problems partially. Therefore a correct strategy is to work on a problem until one gets it fully before moving to the next, and when no more full solves can be achieved, scrape (which is harder in math).

  The same can be said for competitive programming. A mistake I made was to avoid fully solving each problem, which would be much more efficient for my score. If, instead of spending time on Lapida, I got AC on the first one or two problems, my score would be higher. I suppose this is a lesson I need to learn, and I think I can try to correct this through training – work on an algorithm for a solid hour or so before giving up.

- *Contest meta: logistics.* The type of competitive math and programming I'm interested in involves long periods of time. The logistics of this also matter: taking breaks for food or stretching, how much time to spend on each problem, preparing for the contest, and so on.

  I suppose some of my habits I've carried over from math. I'm the type who likes to step out in the middle to take a walk, drink some water, stretch. It helps me think through the problems and get a new perspective. I'm also the type who allots some time for this, and not just work non-stop, which I find to be more effective for me.

- *Relative skill.* In the IMO, there are a few countries which always manage to perform very well. China, USA, South Korea, to name the first three that come to mind. There is a parallel in competitive programming, where it is Russia and China. The distribution of skill seems to be a log-normal distribution for both, but in a local scale it seems more discrete. This is manifested through tiering, groups of people of about the same skill.

[12] Cf. Kruskalblocked.

Asian-Pacific countries seem to perform relatively well. As Geoff Smith remarked in math, the nations which do well have at least one (and probably at least two) of the following: (1) a large population, (2) a large proportion with good education, (3) well-organized training infrastructure, (4) culture which values intellectual achievement. (Alternatively, one needs "a cloning facility and a relaxed regulatory framework.")[13]

Problems

Problems can be seen here.

1. *Guardians of the Lunatics.* An array of P integers is given. One can pick several subarrays[14] of length exactly k, such that each element in the array is in at least one subarray. The *score* of a set of subarrays is the total of the sum of the elements in each subarray. Given the array and k, output the minimum value of the score. (Up to $P \leq 100K$, $k \leq P$.)

2. *Agadoo.* A circular array with the integers $1, 2, \ldots, N$ arranged counterclockwise has a pointer starting at 1. Then k updates of the format a, b are given. This means to repeat for a times the following operation: increase the value of the pointed element by b and move the pointer once clockwise.

   We then move the pointer once counterclockwise and answer q queries, consisting of a number c. We sort the elements in ascending order, breaking ties by shorter clockwise distance from the pointer. For each query, output the original index of the cth element of the sorted list. (Up to $N \leq 10B, k \leq 10K, c \leq 10K$.)

3. *Spreadsheet Game.* A connected subset of an $a \times b$ array is given, such that if two cells in the same row/column are in the subset, all cells between them in the same row/column are also in the subset. Each cell is filled with a positive integer. Two players play a game, where a move consists of one or both of the following:

   - choose a row and delete the contents of all cells above or below that row,
   - choose a column and delete the contents of all cells to the left or right of that column.

   A move has to delete at least one cell and leave at least one cell remaining. The game ends when one cell is left. The first player wants to maximize its value, while the second player wants to minimize it. Given the array, output the value of the last cell if both players play optimally. (Up to $a, b \leq 70K$.)

4. *Constructing a City Plan.* We have an undirected, unweighted, simple connected graph. For all pairs of vertices, consider the shortest distance between them. The *score* of the graph is defined as the sum of these shortest distances. Given n and k, output the adjacency matrix of a graph with n vertices and a score of k, or state if such a graph does not exist. Scoring is unusual; one may choose not to answer a query for less points. (For each $n \in [2, 30]$, all $1 \leq k \leq 0.1303n^3 + C$ is queried, where C is a constant less than 6.)

[13] Taken from the foreword to the book Infinity.

[14] Here we make the convention that a *subarray* is a contiguous portion, while a *subsequence* not necessarily.

5. *Lito Lapida's Lost Lapida.* An undirected, weighted, simple connected graph is given. Consider the graph made by removing all edges with weights outside an interval $[a, b]$. If the resulting graph is still connected, the interval $[a, b]$ is called *good*. For several queries of A, B, output the number of good intervals with $A < a, b < B$. (Up to 100 vertices, 10^5 edges, 10^5 queries.)

Results

A dashed score and a score of 0 mean different things: in the former, the participant did not submit, in the latter, the participant submitted and got a score of zero. Robin and I took early, the rest took it in-house.

#	Name	1	2	3	4	5	Total
1	Robin Christopher Yu	100	100	63	14.82	—	277.82
2	Carl Joshua Quines	68	55	63	—	0	186.00
3	Farrell Eldrian Wu	100	55	0	14.82	—	169.82
4	Dan Alden Baterisna	100	0	—	—	35	135.00
5	Franz Louis Cesista	8	8	100	0.80	0	116.80
6	Alexander Go	8	0	83	14.82	0	105.82
7	Andrew Ting	8	8	—	0.00	—	16.00
8	Kim Bryann Tuico	8	0	—	—	0	8.00