ARC take-home test

You should first see these instructions before starting the test, but they are copied at the top of the actual test for reference.

Instructions

Thank you for taking our take-home test!

The test lasts for 3 hours and consists of math and computer science puzzles of varying difficulty.

You may compose your submission however you like, but please convert any documents to text, images or PDF before sending (e.g., using print to PDF software). We are more than happy to accept either plain text with approximate LaTeX formatting for mathematical expressions, or photos of handwritten solutions, as long as they are legible. When finished, include your files as attachments in a reply to the email message inviting you to take this test.

Before starting the test, we recommend that you:

- Have available something to write on for scratch work.
- Have ready any software you may need to compose your submission.
- Practice solving and writing up a few puzzles of your choice, if you wish.

You answers should generally come with an informal but clear explanation of how you arrived at your answer. If a question asks you to prove or show something, an informal explanation is enough, as long as it is relatively clear how to convert it to a proof. If a question asks you to "sketch" something, there may be technicalities that it is OK for you to ignore; if a question asks you to "prove carefully" something, it is more important to spell out the details explicitly.

Some of the questions are challenging and we expect many excellent candidates to get stuck at some point. If you do not manage to solve a question, please submit any ideas you have towards a solution, even if they are rough thoughts. Please still explain these thoughts clearly rather than submitting scratch work.

To give you a sense of the level of detail you should include in your solutions, here is an example question (simpler than the ones on the test) and solution (in plain text format).

> **Question.** There are n cats and n dogs sat round a circular table, arranged randomly, for some $n > 1$. What is the expected number of dogs that have a cat sat next to them on both sides?

Solution. Each dog has a cat on both sides with probability n/(2n-1) * (n-1)/(2n-2) = n/(2(2n-1)), so by linearity of expectation the answer is n^2/(2(2n-1)).

Some more information about the questions on the test:

- The test consists of four questions, each of which is longer than the example above, often split into multiple parts.
- We put similar weight on each of the four questions, but the weight we give to each part within each question has more variation.
- We are primarily interested in the correctness of your answers and the logical validity of your explanations. We will try to ignore superficial aspects of presentation such as spelling, grammar, phrasing and layout, unless they hinder our ability to understand your work.
- The final part of the final question (question 4 part 3) is significantly more challenging than the rest of the test, but it is still possible to perform well without attempting this part.

You may use Wikipedia and numeric programming software such as Python, although it should be possible to complete the test without these. You may not use other websites or Mathematica.

Please do not share any information about the contents of this test. After the test, you may keep a copy of your solutions as long as you do not share them. Please do not keep a copy of the test itself or the link to it.

If you have any questions about these instructions, please email hiring@alignment.org. We won't be able to respond to any inquiries that are sent while you are taking the test.

1 Threads

Consider the following pseudocode:

```
for $i$ from 1 to 10 do
    $x \leftarrow \text{read}()$
    $x \leftarrow x + 1$
    $\text{write}(x)$
end for
```

Suppose we run this code in parallel on 1,000,000 processors, with the read and write operations acting on a single shared memory location, which is initialized to 0. Assume that each line of code is atomic, but the lines can be interleaved in arbitrary ways. What is the smallest possible value for the number in the shared memory location once all the processors have finished running? Prove carefully that you can't end up with anything smaller.

(If you don't know what it means to run code in parallel, imagine each processor is a person in a line with a copy of the code. Every second, someone is selected at random. That person executes the next line of their code (remembering their local values of x and i, and where they are in the code). If that person is done, they get removed from the line. The program halts when there are no more people in the line.)

2 Running fast and slow

This problem considers algorithms that take as input a sequence of binary strings $x_1, \ldots, x_k \in \{0,1\}^*$ and output a single bit $z \in \{0,1\}$. We say that a function $f(x_1, \ldots, x_k)$ is *easy* if is computed by some algorithm whose running time is at most cN for some constant c, where $N = \max_i |x_i|$ is the maximum length of any of the input strings. We say that a function $f(x_1, \ldots, x_k)$ is *hard* if it is not computed by any algorithm which runs in time cN^d for any constants c and d.

Helpful fact: there exists a hard function $H : \{0,1\}^* \to \{0,1\}$. You may use the function H as an ingredient in any of your constructions.

1. Exhibit a function $U : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$ such that
$$R_1^U(x, y, y') := \mathbb{1}_{U(x,y) > U(x,y')}$$
is easy but
$$R_2^U(x, x', y) := \mathbb{1}_{U(x,y) > U(x',y)}$$
is hard.
2. Exhibit a function $U : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$ such that both R_1^U and R_2^U are easy, but
$$R^U(x, y, x', y') := \mathbb{1}_{U(x,y) > U(x',y')}$$
is hard.

3 Variance of numbers

You have a button. When you press the button, you get a random sample from the standard normal distribution (i.e. mean 0, variance 1). (The sample is independent of the samples you got from all previous presses.) For each $z \in \{0,1\}^*$ (the set of all finite bit strings), we define a random variable X_z as follows:

- $X_\emptyset = 0$ (where $\emptyset$ denotes the empty bit string).
- To get X_0, you press the button and add the number you get to $X_\emptyset$; similarly, to get X_1, you press the button again and add the number you get to $X_\emptyset$ (so X_0 and X_1 are independent random samples from the standard normal distribution).
- To get X_{00}, you press the button and add the number you get to X_0; similarly, to get X_{01}, you press the button and add the number you get to X_0. To get X_{10}, you press the button and add the number you get to X_1; similarly, to get X_{11}, you press the button and add the number you get to X_1.
- In general, to get X_{zb}, where b is 0 or 1, you press the button and add the number you get to X_z.

1. For a positive integer n, what is the expected value of the population variance of the $\{X_z : z \in \{0,1\}^n\}$ (i.e. the set of 2^n random variables corresponding to bit strings of length n)? That is, compute

$$\mathbb{E}\left[\frac{1}{2^n}\sum_{z\in\{0,1\}^n}(X_z - \overline{X})^2\right],$$

where $\overline{X} = \frac{1}{2^n}\sum_{z\in\{0,1\}^n} X_z$.

2. Answer the previous question if instead of a sample from a standard normal, when you pressed the button, you would get either 0, 1, or 2 at random (with equal probability).

4 Interval chopping

In this problem, you will be chopping the interval $[0, 1]$ into pieces with n cuts. Your goal will be to make the longest piece as short as possible.

1. (Warm-up / check for understanding) Show that you can make the longest piece have length $\frac{1}{n+1}$. Also show that it is impossible to do better: no matter how you chop the interval, there will always be a piece with length at least $\frac{1}{n+1}$.

2. You are now playing a game against an opponent, Longfellow, who wants the longest piece to be as *long* as possible. You and Longfellow take turns making chops. Longfellow goes first, and you each make n chops. But now, when it is your turn, you *must* chop the currently-longest piece (or, if there is a tie, you must choose one of the longest pieces to chop).[1] (Longfellow can chop anywhere, on the other hand.) If you and Longfellow both play optimally, how long will the longest piece be? Explain your reasoning.[2]

3. Now you don't have an opponent, but there is a different restriction on where you can chop. Before every chop, a random piece is selected according to its length, and you must chop that piece.[3] Your goal is now to minimize the *expected value* of the length of the longest piece.

   (a) Describe a strategy that performs well and prove (or sketch a proof of) an upper bound on the expected length of the longest piece under your strategy. (You should think of n as being very large. We are interested in strategies that perform well asymptotically in n, and do not care about constant factors.)

   (b) Prove (or provide a proof sketch of) a lower bound for this problem. That is, find an expression in terms of n and argue that there is *no strategy* for which the expected length of the longest piece is smaller than the expression.

   (To get full credit on this problem, your lower bound should equal your upper bound from the previous Part (a) (up to a constant factor), but we will be generous with partial credit, so don't sweat it if you don't end up with matching lower and upper bounds!)

[1] For example, if the interval has been chopped three times – at 0.2, at 0.7, and at 0.9 – then you must chop somewhere between 0.2 and 0.7, inclusive.

[2] Chops at 0 and at 1 and at the endpoints of pieces are allowed.

[3] For example, if you've made two chops – at 0.1 and at 0.4 – then the piece $[0, 0.1]$ will be selected with probability 10%, $[0.1, 0.4]$ with probability 30%, and $[0.4, 1]$ with probability 60%.