CS 341 - Algorithmic Challenges

Solving any of these problems gets you an automatic 100 for the course, and in some cases, a financial reward as well! Decisions about what constitutes a solution are entirely at the discretion of the instructor.

Challenge #1

From Lecture 1: here is the simplest algorithm I know whose analysis is still incomplete:

```
PIERCE(a, b)   /* a, b are integers with a > b > 0 */

while (b <> 0) do
        b := a mod b;
```

It is known that the running time (in the unit cost model) of this algorithm is $O(a^{1/3})$, and $\Omega(\log a)$ for infinitely many a, which is a big gap between upper and lower bound.

I offer an automatic 100 in the course and $100 to anyone who can substantially improve these upper or lower bounds.

Challenge #2:

From Lecture 7: An *abelian* square is a string of length $2n$ where the last n symbols are a permutation of the first n. An example in English is `reappear`.

Find an algorithm for deciding if a string of length n has an abelian square appearing somewhere in it, that runs significantly faster than $\Theta(n^2)$. Recently this has been improved to $O((n/\log n)^2)$, so I'd want something significantly faster than that. (For recent work see here.)

Or prove some nontrivial lower bound for the problem. (If you change the problem so that you have to report *all* abelian squares, there is a trivial lower bound of $\Omega(n^2)$ because that's how many abelian squares there are in a string of n a's. So we are looking for a lower bound for the problem where all you have to do is determine the existence of the abelian square, not report all of them.)

I offer an automatic 100 in the course and $100 for the solution.

Challenge #3:

From Lecture 8: Prove or disprove that every fraction of the form 4/n, n an odd integer > 1, has an Egyptian fraction representation with exactly 3 distinct terms. Nobody currently knows how to do this in all cases, although many special cases are known. For more information, see Richard Guy, *Unsolved Problems in Number Theory*, Section D11. I offer an automatic 100 for the course and $50 for the solution.

Challenge #4:

From Lecture 8: Prove or disprove that every m/n, n odd, has a finite expansion where at each step we greedily choose the least possible *odd* integer 1/i which leaves a non-negative remainder. I offer an automatic 100 for the course and $50 for the solution.

Challenge #5:

From Lecture 19: consider a directed acyclic graph on n vertices, with a single distinguished vertex called the "source", and a single distinguished vertex called the "sink". (In other words, a unary NFA that accepts a finite language.) Compute the *lengths of all possible paths* from the source to the sink, and report them all (in any reasonable compressed form). It is known how to do this in $O(n^e \log n)$ time, where e is the optimal exponent for matrix multiplication, so I'd want something significantly better than this, say $o(n^e)$. Here the model of computation is essentially the log cost model (but multiplication of n-bit numbers costs $n^{1+\varepsilon}$).

A very common approach in trying to solve this problem is the use of breadth-first or depth-first search. However, these techniques alone are very unlikely to be helpful, because they record whether a vertex is reached, but not the length of all possible paths to get to this vertex, which is what is needed.

I offer an automatic 100 in the course and $100 to anyone who can substantially improve the running time.

Solved! (sort of) by Aaron Potechin here.

Challenge #6: Restivo's problem

Let S be a finite set of words (strings) over a finite alphabet Σ. Recall that S^* means the set of all words formed from all possible concatenations of words of S. By Fact(S^*) we mean all possible subwords of the words of S^*. (By "subword" we mean a contiguous block within a word.)

Suppose that Fact(S^*) ≠ Σ^*, and suppose k is the length of the largest word of S. Let t be the length of the shortest word in Σ^* - Fact(S^*). Restivo's problem is to determine how big t can be as a function of k. Examples are known where t is about $5k^2$. No examples are known where 5 is replaced by something larger.

This is a problem that, in principle, could be solved by a massive search for counterexamples. I offer 100 for the course and $10 for any significant improvement on the existing bounds.

Challenge #7: Morphism generating additive squarefree word

Consider the map h that sends

- 0 to the string 03
- 1 to the string 43
- 3 to the string 1
- 4 to the string 01

It is called a morphism.

Now apply h to itself on 0 over and over again: h(0) = 03, h(h(0)) = 031, h(h(h(0))) = 03143, etc. In the limit you get the infinite word 0314301103434303101101103 ... which has the following amazing property: it contains no additive cubes, that is, no three consecutive blocks of the same length and same sum.

Nobody currently knows how to create an infinite word over a finite subset of **N**, the non-negative integers, that avoids additive *squares*: having no two consecutive blocks of the same length and same sum.

Try to find a *candidate* morphism whose infinite iteration on the symbol 0 avoids additive squares (that is, let's say, it works for the first 10,000 symbols of the infinite string so produced). It should be over a small alphabet; say under 15 symbols.

You might be able to find a candidate by a vast brute-force search. Maybe some heuristics will help. There is no need to actually prove that it works! If it is over a small alphabet and works (say) for the first 10,000 symbols, it'll probably work forever. You will need an alphabet of size at least 4. At the moment nobody even has a candidate.

I offer 100 for the course and $100 for anyone coming up with a good candidate that actually works.

Challenge #8: Optimal method for sorting 16 numbers

Currently we know the optimal sorting method for sorting *n* numbers for *n* = 1, 2, ..., 15. But for 16 numbers the answer is not known. Here we are talking about comparison-based methods and counting the worst-case number of comparisons. A decision tree for 16 numbers has 16! = 20922789888000 leaves. It is known that 46 comparisons suffice in the worst case (i.e., somebody found a method that achieves this), but 44 comparisons are not enough (i.e., any method using 44 comparisons will not have enough information to distinguish between two different orderings). So, what is the optimal number? 45 or 46? (See here.)

I offer 100 for the course to anyone solving this problem. Theoretically it could be done with a large calculation. You'd probably also get a paper out of it.

Challenge #9: Longest common subsequence problem

Here's an open problem about longest common subsequences. It was suggested by French mathematician Jean Berstel in 2006, and to my knowledge is still unsolved. Here it is: define two sequences of strings as follows:
$X_0 = 0$
$Y_0 = 1$,
and for $n \geq 1$, we define
$X_{n+1} = X_n Y_n$ and
$Y_{n+1} = Y_n X_n$.
Finally, define a(n) to be the length of the LCS of X_n and Y_n. It's easy to verify that a(1) = 1, a(2) = 2, a(3) = 5, and so forth.

First problem: find a good formula for a(n).

Second problem: prove or disprove that $a(n) = 2^n(1-o(1))$. I offer an automatic 100 for the course for your own solution to either of these problems. (No points for finding the solution in the literature already, but I would be pleased if you would tell me if you find it already solved somewhere.) The first few values of the sequence a(n) can be found here.