1 Ideas

The following prompts are merely suggestions, and in many cases I list more than can reasonably be discussed in a single paper to give you multiple options to explore. Please concentrate on explaining a few ideas well instead of superficially covering many. Also, many of these suggestions involve comparing the speed of algorithms. To time computations in Python, use the `timeit` module.

- P versus NP. Explain what P, NP, and NP-hard mean, what the P versus NP problem is, and why it is important. The first four sections of this survey are essential reading. The first two chapters of [2] will also be useful. (In lieu of an expository paper, I will also accept any proof that resolves the $P \neq NP$ conjecture.)

- There are some problems in computer science that can't (?) be solved efficiently, such as the traveling salesman problem or other combinatorial optimization problems. In practice, a variety of probabilistic and heuristic approaches (that may not obtain the true solution, but only a good substitute), such as simulated annealing, are used. Find one of these "NP-complete" problems and report on practical methods to find solutions or near-solutions. Chapter 7 and Part II of this book are good starting points, but you will need to do some further research. Good intuition for simulated annealing is given in [8].

- The practical use of RSA encryption depends on our ability to find large prime numbers quickly. Figure out how this is done. (Hint: generate large random numbers and test for primality.) Why can we test for primality quickly, but actually factoring the numbers takes a long time? Report on one of your favorite primality testing algorithms, such as AKS, Lucas, Miller–Rabin, or the Fermat test [11]. The latter two are probabilistic. Explain what this means, and why it does or doesn't affect their day-to-day use. How fast do such algorithms run? Include both a theoretical analysis and a practical demonstration in Python (e.g. of producing keys for RSA).

- Suppose you want to factor a relatively large number n. What's the best algorithm for doing this? The naive approach of checking every prime up to $\sqrt{n}$ ("trial division") is slow and bad. The state of the art here involves quite advanced number theory (elliptic curves), but there are some very good and relatively simple algorithms, such as the quadratic sieve and Pollard's rho algorithm, that might make for a good paper. Pick one algorithm, prove it works, discuss its performance theoretically, and provide practical examples comparing its speed to trial division in Python. (You might also discuss why trial division is so slow from a theoretical perspective; a sketch of this argument is given on Wikipedia.)

- What is a block cipher? Describe an example. What are the best practices for using them? (What should be avoided? Why is ECB bad?) Blog post (see bulleted list).

- People often believe that if they encrypt a message twice, they get the double the security. This is wrong. Report on the Data Encryption Standard, including its technical details (at least in outline) and history, then explain why meet-in-the-middle attacks mean that double DES is not much more secure than single DES, and why this motivated the US government to adopt triple DES instead [5, 12, 6].

- Explain what a Markov chain is, then the importance and implementation of Markov Chain Monte Carlo methods (for instance the Metropolis algorithm). Program an example (for example sampling from the Gibbs measure for the Ising model on a lattice). The first two sections of this paper give a rapid introduction, while [8] is more leisurely. The example could be from Bayesian statistics, combinatorial optimization, code breaking [4], or statistical physics.

- It is often the case that communication channels are not perfect and produce some noise or error in the transmission. The naive solution is to transmit the message multiple times for redundancy, but this is horribly inefficient. Explain Shannon's noisy channel coding theorem and/or your favorite efficient error-correcting code [8]. (Linear codes might be a good starting point.) A similar idea is to explore a good file compression algorithm.

- The following list of ideas is too long for one paper, so pick and choose your favorites. Any other interesting thoughts about password security are welcome.

  Explain hash functions [6] and their relevance to computer passwords and data integrity checking. What techniques can be used to make hashed passwords more secure (e.g. salting)? What techniques can be used to break them? In this direction you could research some combination of masking attacks, dictionary attacks, and rainbow tables. For the latter, the original paper is here, but perhaps the Wikipedia article is easier to read and provides a less technical starting point. (If you're writing about rainbow tables, also explain what a space–time tradeoff is.) With recent advances in computing power and parallel processing (e.g. GPUs), how long and complex should a password be in order to be secure against a motivated adversary? What methods are there for generating secure passwords? (For a fun example, look up Diceware, and analyze its security against a brute force attack mathematically.) How insecure is the average password in use today? What are the social costs of weak passwords?

  See here for an introductory talk. (It was given in 2011 and covers the basics well, but be aware it is slightly out of date in the sense that insecure passwords and now even more insecure due to the rise of GPUs.) There also exist real-world guides to password storage and password cracking (especially [10]); try Googling terms and ideas from them. Please use any information learned in this project responsibly and ethically.

- Do you know some quantum mechanics? Report on Shor's algorithm for quantum computers and explain why it makes certain encryption schemes (e.g. RSA) insecure (assuming an adversary has access to a quantum computer).

- The discrete random walk has a ton of fun properties you could write about. For example, there is a connection between the discrete random walk and the discrete heat equation. Explain it. (Brownian motion and the continuous heat equation are also intimately related. Understanding this is more difficult but would also make a good project.) Other ideas include proving that the random walk is recurrent in dimensions $d \leq 2$ but transient in dimensions $d \geq 3$, or using random walk techniques to prove Bertrand's ballot theorem.

- Consider the following gambling strategy, called the martingale betting system: I go to a casino and bet 100 dollars on roulette, choosing red. This gives me about a 1/2 chance of success. If I win, I have gained 100 dollars and leave happy. If not, I double my bet to 200 dollars. If I win, I have gained $200 - 100 = 100$ dollars, and again I have gained 100 dollars and leave happy. If not, I double my bet again.... Is this a good betting strategy? You may investigate using either the optional stopping theorem (and consider more general strategies) or through the long-term behavior of a (biased) random walk.

  In roulette, the casino has a slight inherent edge that makes my bet negative expected value. If I can instead bet on a fair coin flip, does this change your conclusion?

- Suppose I want to compute a million digits of π on my computer. What's a good way to do this? What ways should be avoided? How many digits can you find? Both Wikipedia and [1] have a lot to say about this problem. Compare the speed of various algorithms by using each to compute some fixed number of digits. Use the Python module `mpmath` for high-precision arithmetic. You might also discuss and implement "spigot" algorithms such as the Bailey–Borwein–Plouffe formula, or provide commentary on historical methods of computation and the sociology of π. Some methods rely on advanced number theory, but some (such as the BBP formula or certain Taylor series expansions) can be proved with elementary methods.

- Some claim there is a "hot hand" effect in basketball and other sports. Is there? Investigate, explain the history of this problem, and render your judgment. This blog post is a good starting point.

- Investigate what is known about the percolation problem in random graph theory [3, 7]. State some important results, explain their meaning, and check them empirically via simulation. This project is a great opportunity to make cool pictures by simulating percolation clusters in Python. If you choose this topic and don't include pictures, I will be sad. (And don't stop at two dimensions!) (Unfortunately, the proofs in this area are rather technical, so I recommend against including them.)

- Discuss auction theory [9] and its real world applications, for example to the design of eBay and FCC wireless spectrum auctions. You should include an in-depth

discussion of at least one type of auction, for example the first-price sealed-bid auction or Vickrey auction.

- For another project with an economic flavor, discuss the Stable marriage problem and its applications (and variations?), for example assigning medical students to residency positions, or public school students in New York City to schools.
- Probability on infinite sample spaces, such as an infinite sequence of coin flips, can get strange. For example, there are sets of of sequences of coin flips to which we cannot coherently assign a probability! In order words, we cannot take our set $\mathcal{F}$ of measurable sets to be all subsets of Ω. Want to learn more? Check out the discussion of non-measurable sets in [13].
- Don't like any of the ideas on this list? Propose your own!

References

[1] Jörg Arndt and Christoph Haenel. *Pi-unleashed*. Springer Science & Business Media, 2001.

[2] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[3] Béla Bollobás, Bela Bollobás, Oliver Riordan, and O RIORDAN. *Percolation*. Cambridge University Press, 2006.

[4] Jian Chen and Jeffrey S Rosenthal. Decrypting classical cipher text using markov chain monte carlo. *Statistics and Computing*, 22(2):397–413, 2012.

[5] Matt Curtin. *Brute force: cracking the data encryption standard*. Springer, 2007.

[6] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography engineering: design principles and practical applications*. John Wiley & Sons, 2011.

[7] Geoffrey Grimmett. What is percolation? In *Percolation*, pages 1–31. Springer, 1999.

[8] Marc Mezard and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.

[9] Paul Milgrom and Paul Robert Milgrom. *Putting auction theory to work*. Cambridge University Press, 2004.

[10] Joshua Picolet and N LLC. Hash crack: Password cracking manual, 2016.

[11] Lasse Rempe-Gillen, Rebecca Waldecker, and Rebecca Waldecker. *Primality testing for beginners*. American Mathematical Society, 2014.

[12] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[13] Elias M Stein and Rami Shakarchi. *Real analysis: measure theory, integration, and Hilbert spaces*. Princeton University Press, 2009.