The Million Dollar Question: How Hard is that Problem?

Abstract:

One of the questions at the forefront of computer science and mathematics research is "Does P = NP?" P and NP are classes of problems based on how difficult they are to solve. An example of a relevant problem is that of the traveling salesperson. The task is to find the shortest path through a set of cities so that the salesperson visits each city exactly once and then returns home. It's easy to check if a candidate solution meets these criteria, but we're not sure how difficult it is to find the solution in the first place. The implications of P ?= NP affect every discipline, and if it was proven that P = NP, things would drastically change.

The Million Dollar Question

Have you ever thought you deserved a reward for finishing a particularly difficult math test? Well, the Clay Mathematics Institute has created seven "millennium problems" deemed important and difficult enough to be deserving of a million dollar prize [1]. One of these is the deceivingly simple question, "Does P = NP?" This question has been at the forefront of computer science and mathematics research since it was posed in 1971 [2]. P and NP are categories of problems, and the question P =? NP asks what problems can be solved quickly. Despite its attention, no one has proven an answer either way.

In computer science, problems are solved with algorithms. An algorithm is like a recipe. Given an input, you can simply follow the step-by-step instructions to get a result. For example, if I asked you to find the tallest person in a line of people, you could follow the following algorithm:

1. Say that the first person in the line is the tallest person you've seen so far.
2. For each individual in the line, compare them to the tallest person you've seen so far.
    2a. If the individual is taller, say that they're the tallest person you've seen so far.
3. Once you've gone through the entire line, the tallest person you've seen so far is the tallest person in the group.

For the purposes of the question P =? NP, this would be considered a fast algorithm. So, finding the tallest person in a group is an easy problem. But how do we decide which algorithms are fast?

What is Running Time?

Running time has to do with how fast an algorithm is. But this definition leaves many questions unanswered. For one thing, the efficiency of an algorithm has to do with its implementation [3]. Let's say that I have an algorithm to split pizza between my friends. My algorithm says that I should hand out all of the pizza slices in a set order so that everybody gets one before anybody gets two. This might take a couple of minutes to accomplish, but shouldn't take too long, right? Well, it depends on how this algorithm is implemented. I could have all of my friends come to my house, and I could hand the pizza out quickly. However, I could also drive across town to each of their houses for every single slice.

Another issue is how the algorithm scales, meaning how it will behave on different input sizes [3]. Imagine that you're making pizzas and you have to slice up the pepperoni. The input size is the number of pizzas you have to make. If you're only making a few pizzas, it's probably more

efficient to slice the pepperoni up by hand than to set up an automatic slicer. But, if you own a restaurant and you're making hundreds of pizzas, the slicer would save you a lot of time.

We also don't know what the inputs will look like [3]. If I give you a stack of 30 pizza boxes and ask you if any of the pizzas are plain cheese, it will take you a lot longer to give me an answer if there's only one cheese pizza than if all of them are cheese pizzas.

In an attempt to create a reasonable metric, runtime is based on how the algorithm would behave on large input sizes with the worst-case input, assuming smart algorithm implementation.

The running time of an algorithm matters because it can dictate whether or not it's possible to solve a problem in a reasonable amount of time. For example, let's say that I'm hosting a blind date pizza night for 100 people. I give you a list of 500 applicants and their favorite types of pizzas. Your job is to give me 50 pairs of people so that both people in each pair have the same pizza preferences. This problem is infeasible because it would take far too long; the number of possible pizza night combinations is more than the number of atoms in the universe.

P versus NP

P stands for polynomial time, meaning that a problem in class P has an algorithm to solve it, and the time this algorithm takes is a polynomial function of the input size [5]. This means that if your input size is n, the amount of time it takes to run the algorithm will be something like 1, n, or n^3. Remember that running time is based on large input sizes. On large numbers, the difference between a polynomial function and a superpolynomial function, one that grows faster

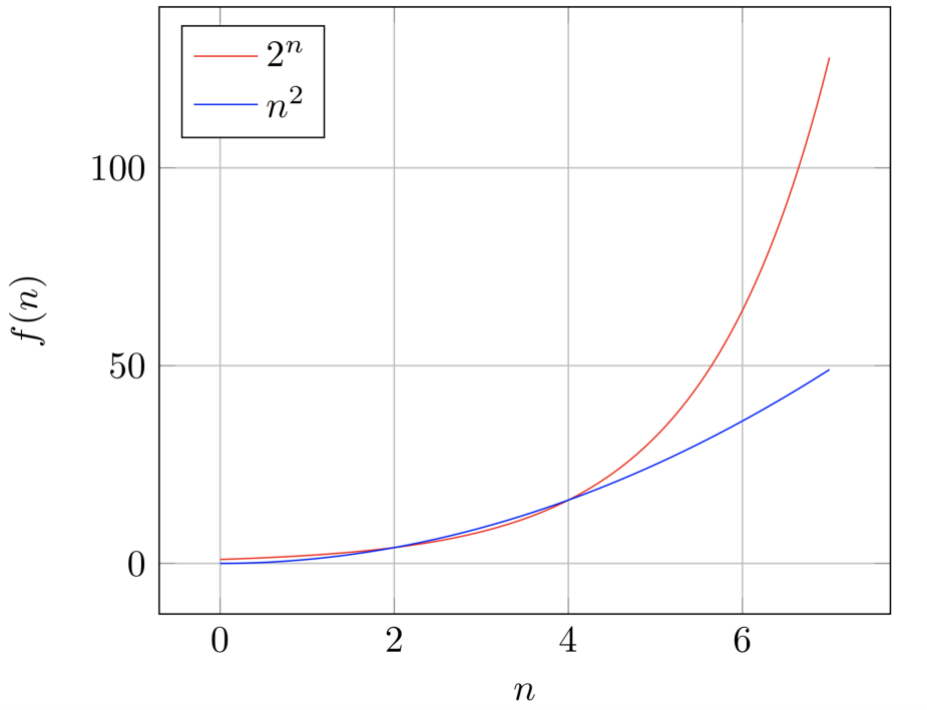

Figure 1. Comparison of polynomial and superpolynomial function.

than its polynomial counterpart, is drastic. See Figure 1 for a comparison between a polynomial function, n^2, and a superpolynomial function 2^n. Notice that as *n* increases, the polynomial function is much smaller. This means that on a large input size, an algorithm running in polynomial time would be much faster. In practice, a problem being in P means that there is an efficient algorithm to solve it and the problem is easy to solve.

NP stands for nondeterministic polynomial time. That's a lot of big words, but it's actually a pretty simple idea; NP means that, given a candidate solution to a problem, this solution can be checked for correctness in polynomial time [6]. Essentially, checking an answer is efficient and easy. For example, let's say I gave you a bunch of pizza orders and asked you total the cost. Given the pizza orders and your answer, it would be easy for me to check if you were correct. This puts the problem in NP.

Some problems are in both P and NP. The previous problem is in NP because it would be easy for me to check your answer. It's also in P because it would be easy for you to find the answer; you simply add the costs of each pizza order.

Some problems are extremely difficult and are in neither P nor NP. If I gave you a large number n and you had to make 2^n pizzas, that would take a long time because this is a superpolynomial function. It would also take me a long time to check if you had done it properly because I would have to count up all of your pizzas (and there are a lot of them).

We also know that if a problem is in P, it is in NP [5]. This is because any problem that's easy to find a solution for will also be easy to check a solution for; we could simply find the solution and see if it matches.

Now comes the big question, are there any problems that are in NP but are not in P? Are there any problems that are easy to check, but difficult to find solutions for? The instinctual answer is yes, and most academics would agree [4]. However, the question has never been definitively answered. See Figure 2 as a reminder of which combinations of P and NP are possible.

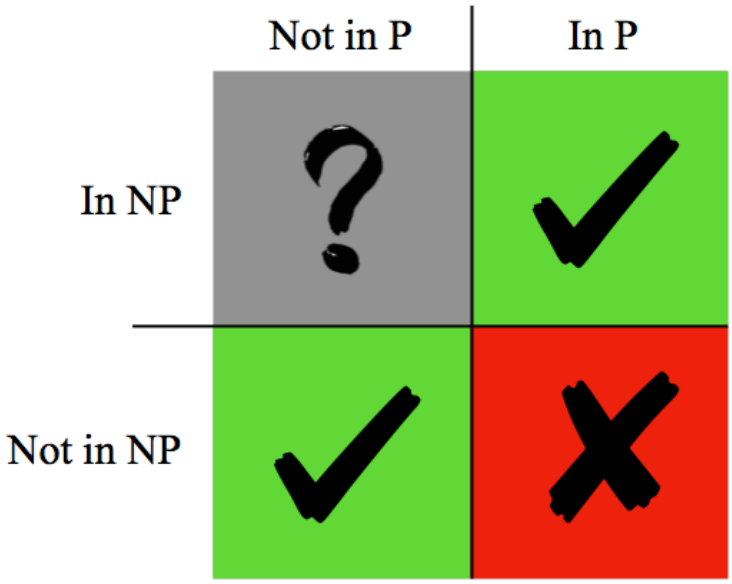

Figure 2. Chart depicting which combinations of P and NP are possible.

There are a set of problems that are provably the hardest problems in NP, and these are classified as NP-complete. These are defined as the hardest problems because if we could find a simple solution to even one NP-complete problem, we could find a simple solution to every NP problem [7]. This means that if even one NP-complete problem was shown to be in P, all NP problems would be in P, proving P = NP. Most academics think that this isn't the case because there are thousands of NP-complete problems, and millions of researchers have tried to find efficient solutions for them, but to no avail [8]. However, nobody has proven a definitive answer either way. The question remains, does P = NP?

What if P = NP?

Right now, we operate under the assumption that P and NP are not the same, that P != NP. While it would be interesting to actually prove this assumption, there would be much more important implications if it turned out that P = NP.

For one, if P = NP, your financial transactions would no longer be secure! Right now, messages are securely sent across the internet using RSA (the acronym comes from the initials of the creators). The math behind RSA is complicated, but the reason a secure message can't be decoded comes down to the difficulty of factoring large numbers [8]. It's believed that factoring large numbers isn't in P because nobody has been able to come up with an algorithm that would do so in polynomial time. However, nobody has been able to explicitly prove that such an algorithm doesn't exist. So, it's unknown if factoring large numbers, and thus breaking RSA, is in P. We do know that the problem is in NP; if I gave you a set of numbers and told you they're the factors of some large number, you could easily check this by multiplying them together and seeing if the result was equal to the original number. Since RSA is in NP, if it was proven that P = NP, then breaking RSA would be in P. This means that it would be a computationally simple task to decode private messages.

While finding that P = NP would be devastating to internet security, it would also have widespread positive effects in practically every field because it would mean that many problems we currently consider difficult are actually easy to solve. As Fortnow states, "what we will gain from P = NP will make the whole Internet look like a footnote in history" [7].

One of the hottest topics at this moment in computer science is machine learning. Machine learning looks at large volumes of data and creates models based on these. These models are then used to predict future outcomes. It's easy to verify a model by simply checking if it works on all the test data; this means that learning problems are in NP. So, if P = NP, it would also be easy to

come up with a model to fit the test data [7]. Learning tasks such as language recognition, weather prediction, and computer vision would all become trivial [7].

This question even infiltrates the financial sector. Market efficiency is a term in the financial field meaning that current stock prices are all of the information relevant to future prices [9]. The practical implication of market efficiency is that there's no way to beat the market average through stock trades besides luck. This is because everyone has immediate access to all relevant information [10]. Although complicated to prove, it turns out that if P = NP, financial markets are efficient [9]. So, if it was proven that P = NP, that would also mean that there's no way besides luck to beat the market average through stock trades. So, if P = NP, trading stock would be pointless because everyone would be able to fully analyze past market data, and no-one would have an advantage.

A particularly interesting result of P = NP is that computers could easily write mathematical proofs [7]. Checking the validity of a proof is simple to do, so if P = NP, finding a proof would also be easy. However, it's possible to ask your algorithm to prove something for which there is no proof. Because of this, we have to set a limit where we would stop looking for possible proofs. So, if P = NP, it would be a simple task to find proofs with the caveat that they are of reasonable length (Fortnow suggests 100 pages as a sensible limit) [7]. This means that if you could prove P = NP, you could walk away with not only the million dollar prize for that proof, but also the prizes for the proofs of the six other millennium problems! [7].

Finding an Answer

An answer to the question P ?= NP would have widespread implications. It would tell us if our information is secure and if we can create efficient solutions to problems that touch every discipline. A 2002 poll of one hundred professionals in the field found that most of them think this question will be answered by 2040 [4]. However, there's no way to know when it will be resolved; in fact, several of the respondents believe it will never be answered [4].

References

[1] Clay Mathematics Institute (2018, Sep. 20). *Millennium Problems* [Online]. Available: http://www.claymath.org/millennium-problems

[2] M. Sipser, "The history and status of the P versus NP question," in *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing (STOC '92)*. ACM, New York, 1992, pp. 603-618.

[3] J. Kleinberg and E. Tardos, "Basics of algorithm analysis" in *Algorithm Design*, Indian Subcontinent Version. Tamil Nadu, India: Pearson India Education Services Pvt Ltd, 2014, pp. 29-32.

[4] L. A. Hemaspaandra, "SIGACT news complexity theory column 36," *ACM SIGACT News*, Vol. 33, Issue 2, pp. 34-47, June 2002.

[5] S. Cook, "The P versus NP problem" in *The Millennium Prize Problems*. Providence, RI: American Mathematical Society, 2006, pp. 87-106.

[6] H. S. Wilf, *Algorithms and Complexity*, 2nd ed. New York: A K Peters/CRC Press, 2002.

[7] L. Fortnow, "The status of the P versus NP problem," *Communications of the ACM*, Vol. 52, No. 9, pp. 78-86, September 2009.

[8] S. Cook. "The importance of the P versus NP question," *Journal of the ACM*, Vol. 50, Issue 1, pp. 27-29, January 2003.

[9] P. Maymin, "Markets are efficient if and only if P = NP," *Algorithmic Finance*, Vol. 1, No. 1, pp. 1-11, March 2011.

[10] M. Hall. (2018, Feb. 7). *What is Market Efficiency?* [Online]. Available: http://www.investopedia.com/insights/what-is-market-efficiency/