One Answer to solve them ALL : P vs NP Problem EXPLAINED

By Prasad Madhale -http://www.mixscoop.com/one-answer-to-solve-them-all-p-vs-np-problem-explained/

The most notorious problem in theoretical computer science remains open, but the attempts to solve it have led to profound insights.

The P versus NP problem is a major unsolved problem in computer science.If you spend time in or around the programming community you probably hear the term “P versus NP” rather frequently.

So here’s a simple and concise explanation:

P vs. NP

> *The P vs. NP problem asks whether every problem whose solution can be quickly verified by a computer can also be quickly solved by a computer.*

So let’s figure out what we mean by P and NP.

P problems are easily solved by computers, and NP problems are not easily *solvable*, but if you present a potential solution it’s easy to *verify* whether it’s correct or not.

All P problems are NP problems. That is, if it’s easy for the computer to solve, it’s easy to verify the solution. So the P vs NP problem is just asking if these two problem types are the same, or if they are different, i.e. that there are some problems that are easily verified but not easily solved.

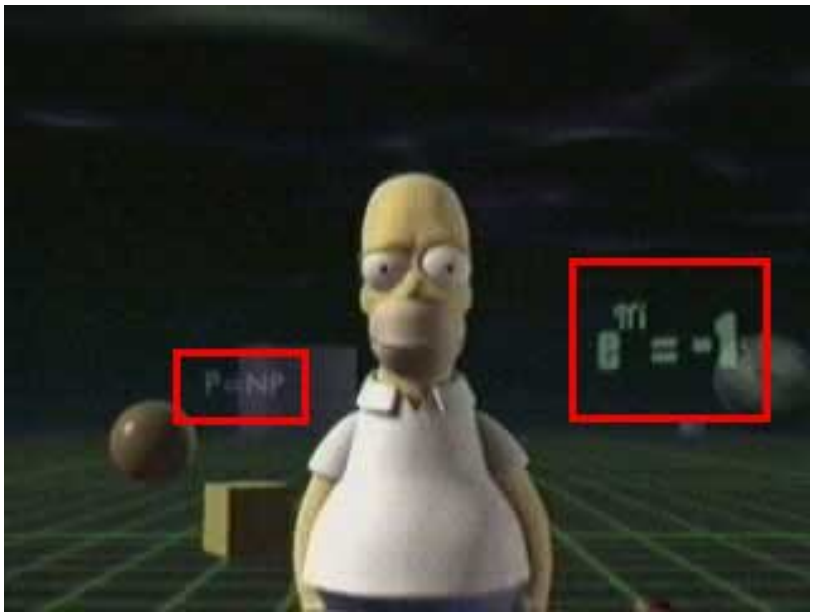

Illustration 1: “In the 1995 Halloween episode of The Simpsons, Homer Simpson finds a portal to the mysterious Third Dimension behind a bookcase, and desperate to escape his in-laws, he plunges through. He finds himself wandering across a dark surface etched with green grid lines and strewn with geometric shapes, above which hover strange equations. One of these is the deceptively simple assertion that P = NP.”

It currently appears that P ≠ NP, meaning we have plenty of examples of problems that we can quickly verify potential answers to, but that we can’t solve quickly.

But “P versus NP” is more than just an abstract mathematical puzzle. It seeks to determine–once and for all–which kinds of problems can be solved by computers, and which kinds cannot.

Let’s look at a few examples:

- A traveling salesman wants to visit 100 different cities by driving, starting and ending his trip

at home. He has a limited supply of gasoline, so he can only drive a total of 10,000 kilometers. He wants to know if he can visit all of the cities without running out of gasoline.

•A farmer wants to take 100 watermelons of different masses to the market. She needs to pack the watermelons into boxes. Each box can only hold 20 kilograms without breaking. The farmer needs to know if 10 boxes will be enough for her to carry all 100 watermelons to market.

All of these problems share a common characteristic that is the key to understanding the intrigue of P versus NP: In order to solve them **you have to try all combinations**.

Here's an explanation from a programmer's point of view.

Imagine, for instance, that you have an unsorted list of numbers, and you want to write an algorithm to find the largest one. The algorithm has to look at all the numbers in the list: there's no way around that. But if it simply keeps a record of the largest number it's seen so far, it has to look at each entry only once. The algorithm's execution time is thus directly proportional to the number of elements it's handling — which computer scientists designate N. Of course, most algorithms are more complicated, and thus less efficient, than the one for finding the largest number in a list; but many common algorithms have execution times proportional to N2, or N times the logarithm of N, or the like.

A mathematical expression that involves N's and N2s and N's raised to other powers is called a polynomial, and that's what the "P" in "P = NP" stands for. P is the set of problems whose solution times are proportional to polynomials involving N's.

NP (which stands for non-deterministic polynomial time) is the set of problems whose solutions can be verified in polynomial time. But as far as anyone can tell, many of those problems take exponential time to solve. Perhaps the most famous problem in NP, for example, is finding prime factors of a large number. Verifying a solution just requires multiplication, but solving the problem seems to require systematically trying out lots of candidates.

What if we find the solution to P vs NP problem?

•If anyone were able to show that P is equal to NP, it would make difficult real-world problems trivial for computers.

•Now, if P=NP, we could find solutions to search problems as easily as checking whether those solutions are good. This would essentially solve all the algorithmic challenges that we face today and computers could solve almost any task.

•If P=NP, then a robotic hand would be capable of doing anything an actual human hand can do. If we could find the solution to show that P=NP, we could help cure diseases like cancer and revolutionize society.

Here's a fun fact:

This problem got its fair share of fame when it appeared on the famous TV series 'The Simpsons'.