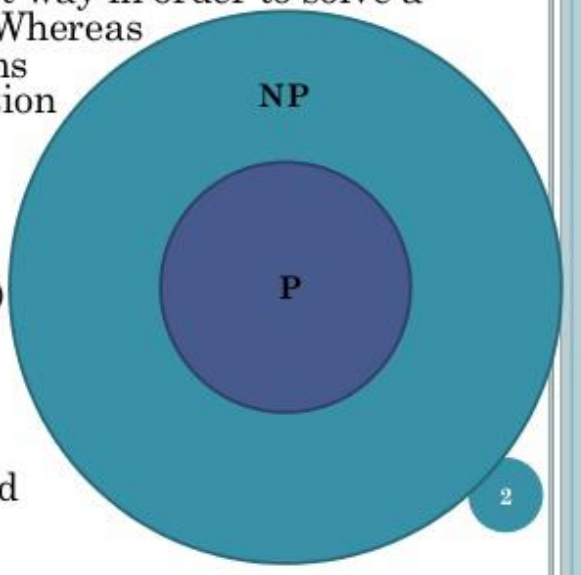
P v/s NP Problem

Submitted by:

VIKAS PALIWAL

Overview of P versus NP Problem

- The P versus NP problem belongs to complexity theory. P denotes the class of computer algorithms which can be executed in a fast way in order to solve a decision or a search problem. Whereas NP corresponds to the problems which can be checked for solution correctness in a fast way whenever a solution is provided.
- Every P problem belongs to NP set (P is included in NP)
- The opposite question whether NP is identical to P is a major unsolved question. We don't know if there exists a fast algorithm which can find the solution to a NP problem



Formal Definition of P versus NP Problem

- Formally, P is the class of problems solved by a deterministic Turing machine in a polynomial time. NP is the class of problems which are solved by a non-deterministic Turing machine in a polynomial time, which means it can be solved by brute force algorithm in an exponential time.
- We know that any P problem is also an NP one. In fact the fast algorithm that solves the P problem can be its fast verification algorithm. The question is could we find an NP problem which doesn't have definitely a fast algorithm to solve it. If so then NP is different from P. If no then NP is identical to P.

Example

Consider Sudoku, a game where the player is given a partially filled-in grid of numbers and attempts to complete the grid following certain rules. Given an incomplete Sudoku grid, of any size, is there at least one legal solution? Any proposed solution is easily verified, and the time to check a solution grows slowly (polynomially) as the grid gets bigger. However, all known algorithms for finding solutions take, for difficult examples, time that grows exponentially as the grid gets bigger. So, Sudoku is in NP (quickly checkable) but does not seem to be in P (quickly solvable). Thousands of other problems seem similar, in that they are fast to check but slow to solve. Researchers have shown that many of the problems in NP have the extra property that a fast solution to any one of them could be used to build a quick solution to any other problem in NP, a property called NP-completeness. Decades of searching have not yielded a fast solution to any of these problems, so most scientists suspect that none of these problems can be solved quickly. This, however, has never been proven

3. WHAT IF P = NP?

To understand the importance of the P versus NP problem let us imagine a world where P = NP. Technically we could have P = NP but not have practical algorithms for most NP-complete problems. But suppose in fact that we do have very quick algorithms for all these problems. Many focus on the negative, that if P = NP then publickey cryptography becomes impossible. True but what we will gain from P = NP will make the whole internet look like a footnote in history. Since all the NP-complete optimization problems become easy, everything will be much more efficient. Transportation of all forms will be scheduled optimally to move people and goods around quicker and cheaper. Manufacturers can improve their production to increase speed and create less waste. And I'm just scratching the surface. P = NP would also have big implications in mathematics. One could find short fully logical proofs for theorems but these fully logical proofs are usually extremely long. But we can use the Occam razor principle to recognize and verify mathematical proofs as typically written in journals. We can then find proofs of theorems that have reasonably length proofs say in under 100 pages. A person who proves P = NP would walk home from the Clay Institute not with one million-dollar check but with seven (actually six since the Poincare Conjecture appears solved).