Alan Turing is probably the most influential computer scientist in the entire world, and there will never be another man like him. He cracked the enigma code during WW2, developed the Turing test, created turing machines, and asked problems still unanswered today. Some call him the founding father of artificial intelligence and computer science. But what often goes unnoticed and what I think may be his most impressive and extensive achievement is inside his paper on the Halting problem. His paper "On Computable Numbers, with an Application to the Entscheidungsproblem" was based on Kurt Godel's incompleteness theorem. From my memory, the theorem essentially states that there does not exist a complete rigorous axiomatic mathematical system, or any system in general! The system will either render itself incomplete due to the rules not being powerful enough to compute problems, or if such a powerful system exists, there are theorems that are true but unprovable. Basically boils down to asking the question " Is this theorem provable in axiomatic system ___ ". The genius of the incompleteness theorem shocked the mathematical world and deserves its own essay (for more information on Godel Incompleteness Theorem see Godel Esher Bach: The Golden Braid). Alan Turing's paper tweaked Godel's Theorem slightly to show the limits of computing. The (interesting part of the proof) boils down to this. There does not exist an algorithm that always correctly decides whether, for an arbitrary program and input, the program halts when run with that input. In layman's terms, there is no one single test or rule that will determine if a program works or not. The rest of the paper was devoted to hypothetical turing machines which could in theory calculate anything computable, which is the basis of modern computing today. But that's boring. What's interesting is P vs NP. Formally, P is the class of decision problems that a deterministic Turing machine can solve in polynomial time. NP on the other hand, are problems that can be solved on nondeterministic Turing machines in polynomial time. Only solutions to a NP problem can be verified on a deterministic Turing machine in polynomial time. Informally, it asks whether every problem whose solution can be quickly verified can also be quickly solved. In essence if P = NP and we can solve every problem that can be quickly verified then we could automate mathematical proofs. P vs NP is a millennium problem, which if proven or disproven earns you a quick million dollars, and has profound implications for mathematics, cryptography, artificial intelligence, game theory, multimedia processing, philosophy, and economics. But most importantly is its implications in computational complexity theory. Which should be quite obvious, so I'll leave it as an exercise for the reader. Here is the issue. Turing's proof of the halting problem seems to suggest that P does not equal NP. For example if P = NP then the halting problem would be decidable (I'm pretty sure). But it isn't, so one might assume that P does not equal NP. And while it has not been rigorously proven, it is where many in the field lean today.