# A Simple Proof of P versus NP

Frank Vega

# A Simple Proof of P versus NP

Frank Vega

*Apartamento 14 Edificio 6, La Portada, Cotorro, Havana, Cuba*

---

**Abstract**

P versus NP is one of the most important and unsolved problems in computer science. This consists in knowing the answer of the following question: Is P equal to NP? Another major complexity class is coNP. Whether NP is equal to coNP is another fundamental question that it is as important as it is unresolved. We shall show there is a problem in coNP that is not in P. Since P = NP implies P = coNP, then we prove that P is not equal to NP.

*Keywords:* P, NP, coNP, maximum, succinct

---

## 1. Introduction

$P$ versus $NP$ is a major unsolved problem in computer science. This problem was introduced in 1971 by Stephen Cook [1]. It is considered by many to be the most important open problem in the field [2]. It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution [2].

In 1936, Turing developed his theoretical computational model [3]. The deterministic and nondeterministic Turing machine have become in some of the most important definitions related to this theoretical model for computation. A deterministic Turing machine has only one next action for each step defined in its program or transition function [4]. A nondeterministic Turing machine could contain more than one action defined for each step of its program, where this one is no longer a function, but a relation [4].

Another huge advance in the last century was the definition of a complexity class. A language over an alphabet is any set of strings made up of symbols from that alphabet [5]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [5].

In computational complexity theory, the class $P$ contains those languages that can be decided in polynomial-time by a deterministic Turing machine [6]. The class $NP$ consists in those languages that can be decided in polynomial-time by a nondeterministic Turing machine [6].

The biggest open question in theoretical computer science concerns the relationship between these two classes:

Is $P$ equal to $NP$?

In a 2002 poll of 100 researchers, 61 believed the answer to be no, 9 believed the answer is yes, and 22 were unsure; 8 believed the question may be independent of the currently accepted axioms and so impossible to prove or disprove [7].

---

*Email address:* vega.frank@gmail.com (Frank Vega)

If $NP$ is the class of problems that have succinct certificates, then the complexity class $coNP$ contains those problems that have succinct disqualifications [4]. That is, a "no" instance of a problem in $coNP$ possesses a short proof of its being a "no" instance [4]. We discuss one problem that is not in $P$. At the same time, we show this language is also in $coNP$. But, we already know if $P = NP$, then $P = NP = coNP$ [4]. In conclusion, we demonstrate the existence of a $coNP$ language that is not in $P$, and therefore, $P \neq NP$ [4].

## 2. Results

**Definition 2.1.** *Given a set $S$ of $n$ (distinct) positive integers and an integer $x$, MAXIMUM is the problem of deciding whether $x$ is the maximum number in $S$.*

How many comparisons are necessary to determine when some integer is the maximum in a set of $n$ elements? We can easily obtain a upper bound of $n$ comparisons: examine each element of the set in turn and keep track of the largest element seen so far, and finally, compare the final result with $x$ [5]. Is this the best we can do? Yes, since we can obtain a lower bound of $n - 1$ comparisons for the problem of determining the maximum in a set of integers [5]. And one final comparison for the verification of whether this maximum is equal to $x$. Hence, $n$ comparisons are necessary to determine whether an element $x$ is the maximum in $S$. This naive algorithm for MAXIMUM is optimal with respect to the number of comparisons performed [5].

On the other hand, a Boolean circuit may be viewed as the computation on the binary input sequence proceeds by a sequence of Boolean operations (called gates) from the set $\{\wedge, \vee, \neg\}$ (logical AND, OR and NEGATION) to compute the output(s). While an algorithm can handle inputs of any length, a circuit can only handle one input length (the number of input gates it has). The efficiency of a circuit is measured by its size.

**Definition 2.2.** *A succinct representation of a set of (distinct) $b$-bits positive integers is a Boolean circuit $C$ with $b$ input gates [4]. The set represented by $C$, denoted $S_C$, is defined as follows: Every possible integer of $S_C$ should be between $0$ and $2^b - 1$. And $j$ is an element of $S_C$ if and only if $C$ accepts the binary representations of the $b$-bits integer $j$ as input. The problem SUCCINCT MAXIMUM is now this: Given the succinct representation $C$ of a set $S_C$ and a $b$-bits integer $x$, where $C$ is a Boolean circuit with $b$ input gates, is $x$ the maximum in $S_C$?*

Let's state our principal Theorem.

**Theorem 2.3.** *SUCCINCT MAXIMUM $\notin P$.*

*Proof.* As we mentioned before, we should need $n$ comparisons to know whether $x$ is the maximum in a set of $n$ (distinct) positive integers when the set $S$ is arbitrary. And this number of comparisons will be optimal [5]. This would mean we cannot always accept every instance $\langle C; x\rangle$ of SUCCINCT MAXIMUM in polynomial-time, because we must use at least $n = |S_C|$ comparisons for infinite amount of cases, where $|S_C|$ is the cardinality of $S_C$. However, $n$ could be exponentially more large than the size of $\langle C; x\rangle$. □

Now, let's define a new problem.

**Definition 2.4.** *Problem **SUPREME**:*

***INSTANCE**: The succinct representation $C$ of a set $S_C$ and a $b$-bits integer $x$, where $C$ is a Boolean circuit with $b$ input gates.*

***QUESTION**: Does every element $y$ of $S_C$ comply with $x \geq y$?*

This previous language is very similar to SUCCINCT MAXIMUM, but the set $S_C$ might not contain the *b-bits* integer $x$.

**Lemma 2.5.** *SUPREME* $\in$ *coNP.*

*Proof.* Let's state the complement language of SUPREME.

**Definition 2.6.** *Problem* ***coSUPREME****:*
***INSTANCE****: The instances of SUPREME.*
***QUESTION****: Is there some element y of* $S_C$ *such that* $x < y$*?*

Every "yes" instance $\langle C; x\rangle$ of coSUPREME could be verified in polynomial-time with a given certificate. Indeed, we can prove in polynomial-time whether $x$ is an integer of a bit-length equal to $b$, where $b$ is the number of input gates in $C$. Moreover, given a *b-bits* integer $y$, we can check whether $C$ accepts the binary representation of $y$ (which means that $y$ is an element of $S_C$) and $x < y$ in polynomial-time, since the comparison of two *b-bits* integers can be done in polynomial-time and the efficiency of the acceptance of $y$ by $C$ only depends in the size of the circuit. Since the bit-length of the certificate $y$ is more short than the size of $\langle C; x\rangle$, then the language coSUPREME will be in $NP$. Consequently, SUPREME would be in $coNP$. □

We say that a language $L_1$ is polynomial-time reducible to a language $L_2$ if there exists a polynomial-time computable function $f : \{0, 1\}^* \to \{0, 1\}^*$ such that for all $x \in \{0, 1\}^*$,

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

**Theorem 2.7.** *SUCCINCT MAXIMUM* $\in$ *coNP.*

*Proof.* Given an instance $\langle C; x\rangle$ of SUCCINCT MAXIMUM and over the assumption that $x$ is in $S_C$, we can state that $\langle C; x\rangle$ would belong to SUCCINCT MAXIMUM if and only if $\langle C; x\rangle$ is in SUPREME. In addition, we could decide whether $x$ is in $S_C$ in polynomial-time just verifying whether $C$ accepts the binary representation of $x$, because the efficiency of the acceptance of $x$ by $C$ will only depend in the size of the circuit. Hence, it will exist an identity polynomial-time reduction from SUCCINCT MAXIMUM to SUPREME, such that this will first check whether $x$ is in $S_C$. We say that a complexity class $G$ is closed under reductions if, whenever $L_1$ is reducible to $L_2$ and $L_2 \in G$, then also $L_1 \in G$ [4]. Since $coNP$ is closed under reductions, then we obtain that SUCCINCT MAXIMUM is in $coNP$ [4]. □

**Theorem 2.8.** $P \neq NP$.

*Proof.* The existence of a problem in $coNP$ and not in $P$ is sufficient to show that $P \neq NP$, because if $P$ would be equal to $NP$, then $P = coNP$ [4]. □

## 3. Conclusions

This proof explains why after decades of studying the $NP$ problems no one has been able to find a polynomial-time algorithm for any of more than 300 important known *NP–complete* problems [8]. Indeed, it shows in a formal way that many currently mathematically problems cannot be solved efficiently, so that the attention of researchers can be focused on partial solutions or solutions to other problems.

Although this demonstration removes the practical computational benefits of a proof that $P = NP$, it would represent a very significant advance in computational complexity theory and provide guidance for future research. In addition, it proves that could be safe most of the existing cryptosystems such as the public-key cryptography. On the other hand, we will not be able to find a formal proof for every theorem which has a proof of a reasonable length by a feasible algorithm.

## References


[1] S. A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the 3rd IEEE Symp. on the Foundations of Computer Science, 1971, pp. 151–158.

[2] L. Fortnow, The Status of the P versus NP Problem, Communications of the ACM 52 (9) (2009) 78–86. doi:10.1145/1562164.1562186.

[3] A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem, Proceedings of the London Mathematical Society 42 (1936) 230–265.

[4] C. H. Papadimitriou, Computational complexity, Addison-Wesley, 1994.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, MIT Press, 2001.

[6] O. Goldreich, P, Np, and Np-Completeness, Cambridge: Cambridge University Press, 2010.

[7] W. I. Gasarch, The P=?NP poll, SIGACT News 33 (2) (2002) 34–47.

[8] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, 1st Edition, San Francisco: W. H. Freeman and Company, 1979.