ON THE SOLUTION OF THE 3RD CLAY MILLENNIUM PROBLEM.
A SHORT AND ELEGANT PROOF THAT P ≠ NP IN THE CONTEXT
OF DETERMINISTIC TURING MACHINES AND ZERMELO-FRANKEL SET THEORY.

Constantine E. Kyritsis*
*Associate Prof. of TEI of Epirus Greece. ckirisi@teiep.gr
C.kyrisis@yahoo.com, TEI of Epirus, Dept. Accounting-Finance Psathaki Preveza
48100

ABSTRACT
In this paper we try to handle the confusing situation in the literature of having plenty many proofs published in respectable journals with referees, some of them of 100 or more pages, that prove contradictory things like P ≠ NP, P = NP, or suggestions that both are non-provable. Obviously not all of them can be correct. We try to clear the situation by providing a very short but decisive proof that P ≠ NP, so short that anyone familiar with the area, would discover any flaw or error if it existed. We provide a simple proof in the context of the ZF set theory and deterministic Turing machines. We discuss also the subtle implications of considering the P versus NP problem, in different axiomatic theories! The results of the current paper definitely solve the 3rd Clay Millennium problem P versus NP, in a simple and transparent way that the general scientific community, but also the experts of the area, can follow, understand and therefore becoming able to accept.

Key words: 3rd Clay Millennium problem, EXPTIME-complete problems, NP-complexity, P-complexity
Mathematical Subject Classification: 68Q15

1. Introduction
In the history of mathematics, it is known that difficult problems that have troubled a lot the mathematicians, turned out to have different proofs one simple and one very complex. Such an example is if the general 5th order polynomial equation can be solved with addition, subtraction, multiplication, division and extraction of radicals starting from the coefficients. This was a problem that troubled the mathematicians for centuries! No doubt a very difficult problem. Yet it was the famous mathematician Niels Henrik Abel who gave a very simple proof that a general the order polynomial equations cannot be solved so, in about 5-6 pages! On the other hand the proof of the same, by the E. Galois theory, is a whole book of dozens of pages! This is not strange, as it depends on the right twist of semantics and symbols in mathematics. The proof in the way of E. Galois is much longer as it shows also how to solve any such equation when it is solvable! The fact that N. H. Abel solved it in only 5-6 pages should not make us think that it was an easy problem! A second issue that is important to mention, is a statement, that is usually attributed to the famous mathematician Yuri Manin, that “A correct proof in mathematics is considered a proof only if it has passed the social barrier of being accepted and understood by the scientific community and published in accepted Journals” Passing the obstruction of the social barrier, sometimes is more difficult than solving the mathematical problem itself!

An example in the history of mathematics is the Hyperbolic or Bolyai–Lobachevskian geometry. If it was not that the famous and well established J.C.F. Gauss, assured the mathematic community that he himself had also discovered this geometry, the scientific community, would not accepted it so easily. Gauss also mentioned that he refrain from publishing it for obvious reasons. It seems that he was afraid that he might be ridiculed, and so although he himself seemed that he had solved the very famous and century old problem of the independence of the 5th axiom of parallel lines in the Euclidean geometry, he did not dare to try to pass also the social barrier. These two observations seem to apply also in the famous 3rd Clay millennium problem of P versus NP. From the references the published papers in respectable Journals with referees those under the numbers [10], [23],[11], [2], [8.5], [8], [18] claim that they prove that P ≠ NP. While in the references the published papers again in respectable Journals with referees, those under the numbers [3], [12],[21],[23],[27] claim that they prove that P = NP. The [7] suggest that neither P=NP, neither P ≠ NP is provable in the ZF set theory! And there are dozens more proving one or the other direction, or non-provability, that they exist online as reprints, while they have not yet passed the social barrier of being published yet. (See e.g. [25]). It seems to me that it is not probable that all of them have correct solutions.
I was not able to go through and follow all the details, of all these proofs in the above three categories of papers, as I considered it more valuable to try to discover myself a simple, transparent and decisive proof.

We must notice here that the P versus NP problem, is in fact a set of different problems within different axiomatic systems. And in the context of what axiomatic system is the Complexity Theory of Turing machines? Since the complexity theory of Turing machines requires entities like infinite sets of words etc then it is in the context of some axiomatic set theory, together with the axiom of infinite. So we notice that the next are different problems:

1) The P versus NP problem in the Zermelo-Frankel axiomatic system of sets without the axiom of choice and this axiomatic system formulated in the 2nd order formal languages.
2) The P versus NP problem in the Zermelo-Frankel axiomatic system of sets with the axiom of choice and this axiomatic system formulated in the 2nd order formal languages.
3) Etc

We might try to think of the P versus NP problem within the context of the axiomatic system of Peano Arithmetic with or without the axiom of induction and within second order formal languages. But to do so, we must carefully define, what additional axioms or definitions give the existence of infinite subsets of natural numbers that are used in the Complexity Theory.

My main hidden guiding idea in searching for a simple proof of the P versus NP problem, was that what the “arbitrary human-like free-will” of a non-deterministic Turing machine as human-machine interactive software, can do in polynomial time cannot be done by a purely mechanical deterministic Turing machine in polynomial time. Then I had to find the right twist of symbols and semantics to make a valid proof from this idea. The proof of the P versus NP problem in the direction P ≠ NP, is supposed also to mean that the standard practice of encryption in the internet, is safe.

We notice also that the P versus NP:

1) It is a difficult problem, that has troubled the scientific community for some decades
2) It may have simple proofs of a few paragraphs, hopefully not longer than the proof of the Time Hierarchy theorem, which seems to be a deeper result.
3) But it can also have very lengthy and complex proofs, than may take dozens of pages.
4) There many researchers (tens and tens of them) that have claimed to have solved it, either as P=NP, or as P ≠ NP, and even as suggestions that neither are provable, but only a handful of them seem to have been able to pass the preliminary social barrier and publish their solution in conferences or Journals with referees. The rest of them have published online only preprints (see e.g. the [25] P versus NP page). It seems to me though that it is not probable that all of them have correct solutions. Especially in the direction P=NP, there is a common confusion and mistake, that has been pointed out by Yannakakis M. 1998 in [26]. Furthermore this confusing situation has contributed so that although there are publications in respectable Journals, the experts and the scientific community does not seem of being able to decide if the P versus NP problem has been solved or not. This is reasonable, as there are proofs of close to 100 pages, and no average reader would feel comfortable to go through them, and decide for himself if there a flaw or error somewhere. Still it is better to have published results than non-published, and then let the large number of readers to try to find errors or flaws in the solutions if there are any.

So here comes the need of a more challenging problem: Not only to solve the P versus NP problem, but also solve it in such an simple, elegant and short way, so that the researchers will know a decisive proof that can understand and control that P ≠ NP or not, so short that anyone familiar with the area, would discover any flaw or error if it existed.

This is I believe the value of the present paper that provides such a proof in the context of the Zermelo-Frankel set theory (we do not use the axiom of choice), formulated within 2nd order formal languages.

What this proof is or is not:

1) It does not introduce new theoretical concepts in computational complexity theory so as to solve the P versus NP.
2) It does not use relativization and oracles
3) It does not use diagonalization arguments, although the main proof, utilizes results from the time hierarchy theorem
4) It is not based on improvements of previous bounds of complexity on circuits
5) It is proved with the method of counter-example. Thus it is transparent short and “simple”. It takes any Exptime-complete DTM decision problem, and from it, it derives in the context of deterministic Turing machines a decision problem language which it is apparent that it belongs in the NP class decision problems while it does not belong the class P of decision problems.

6) It seems a “simple” proof because it chooses the right context to make the arguments and constructions. So it helps that the scientific community will accept that this 3rd Clay Millennium problem has already been solved.

In relation to the use of oracles, in arguments of complexity theory, we must notice, that their use sometimes is equivalent to the introduction of new axioms that guarantee their existence in complexity theory within the context of ZFC set theory, which sometimes may lead to contradictions and non-consistent axiomatic system, that can prove anything. It is known that often are claimed by authors oracles that decide non-decidable sets. In this paper we do not use in the arguments oracles.

In the paragraph 4, we give an advanced, full proof that P ≠ NP, in the standard context of deterministic Turing machines, solving thus the 3rd Clay Millennium problem.

Some further remarks here are worth making. Time and space complexity of programs and algorithms (or Turing machines) is measured by the speed of convergence to infinite of real functions. This a modern corresponding to the historic parallel, in the ancient times of Euclid, that the geometric linear segments where measured by comparing to one with another (ratios of linear segments). Here the analogue of the Turing machine is the linear segment, and the complexity of the Turing machine corresponds to the measure of length of the geometric linear segment. The history of many centuries proved, that our understanding of the lengths of geometric linear segments was greatly improved when these “ratios” of them were organized as a new ontology of measures and lengths and finally as the system of real numbers, the well known today linearly ordered field of the real numbers. We shifted from the ontology of linear segments to the more abstract ontology of real numbers By analogy to this, one may ask if it is possible to organize such speeds of convergence to infinite of real functions, to a new ontology of larger partially ordered (and non-Archimedean of course) topological field or ring that will measure complexities of software programs (Turing Machines). Shifting therefore from the ontology of Turing machines (software programs) to the ontology of complexity measures of them as elements of partially ordered topological algebraic structure. It is important to understand that such an (non-Archimedean) algebraic structure of complexity measures may be partially ordered rather than linearly or totally ordered, as not all speeds of convergence of real numbers to infinite are always comparable. I was making such thoughts during 1990-1993 when I was working on linearly ordered (topological) fields, e.g. the well known field No that includes the ordinal numbers defined by J.H. Conway (see [4] Conway J.H. On numbers and games, Academic press 1976)

2. Preliminary concepts, and the formulation of the 3rd Clay millennium problem, P versus NP.

In this paragraph, for the sake of the reader, we will just mention the basics to understand the formulation of the 3rd Clay Millennium problem. The official formulation is found in [6] (Cook, Stephen (April 2000), The P versus NP Problem (PDF), Clay Mathematics Institute site). Together with an appendix where there is concise definition of what are the Determinstic Turing machines, that is considered that they formulate, in Computational Complexity theory, the notion and ontology of the software computer programs.

In the same paper are also defined the computational complexity classes P, NP. The elements of the classes P, NP etc strictly speaking are not only sets of words denoted by L, that is not only languages, but also for each such set of words or language L at least one DTM, M that decides it, in the specified complexity so they are pairs (L,M). Two such pairs (L₁, M₁) (L₂, M₂) are called equidecidable if L₁ = L₂ although it may happen that M₁ ≠ M₂. E.g. if the complexity of M₁ is polynomial-time while that of M₂ exponential-time choosing the first pair instead of the second means that we have turned an high complexity problem to a low complexity feasible problem.

The definition of other computational complexity classes like EXPTIME etc can be found in standard books like [13],[16],[17]. In the official formulation [6] there is also the definition of the concept of a decision problem language in polynomial time reducible to another decision problem language.

Based on this definition it is defined that an EXPTIME-complete decision language of EXPTIME is EXPTIME-complete, when all other decision problems languages of EXPTIME have a polynomial time reduction to it. Here is the exact definition
**Definition 2.1** Suppose that \( L_i \) is a language over all words \( \Sigma_i, i = 1, 2 \). Then \( L_1 \leq_{p} L_2 \) (\( L_1 \) is \( p \)-reducible to \( L_2 \)) iff there is a polynomial-time computable function \( f: \Sigma_1 \rightarrow \Sigma_2 \) such that \( x \in L_1 \) if and only if \( f(x) \in L_2 \) for all \( x \in \Sigma_i \).

In the same books [13],[16],[17] can be found the concepts and definitions of **NP-complete and EXPTIME-complete decision problems**. See also [14], [18] where its proved that specific decision problems are EXPTIME-complete.

For simplicity we will consider here only binary alphabets \( \{0,1\} \) and binary set of words \( \Sigma \).

### 3. Well known results that will be used.

We will not use too many results of the computational complexity theory for our proof that \( P \neq NP \).

A very deep theorem in the Computational Complexity is the **Time Hierarchy Theorem** (see e.g. [13],[16],[17],[15],[20]. This theorem gives the existence of decision problems that cannot be decided by any other deterministic Turing machine in less complexity than a specified.

Based on this theorem , it is proved that:

**Proposition 3.1** There is at least one EXPTIME-complete decision problem, that cannot be decided in polynomial time, thus \( P \neq \text{EXPTIME} \).

### 4. The solution: \( P \neq NP \) in the context of deterministic Turing machines.

We will prove in this paragraph that \( P \neq NP \) in the context of second order formal language of the Zermelo-Frankel set theory.

The strategy to do so is quite simple: We will start with an exptime-complete decision problem and its language \( L_{\text{exp}} \), and we will derive from it to an NP class decision problem than cannot be solved in the polynomial time (it does not belong to the class P).

Since we are obliged to take strictly the official formulation of the problem , rather than text books about it, we make the next clarifications.

We will use the next conditions for a Language to be in the class NP, as stated in the official formulation of the P versus NP problem (see [6] Cook, Stephen (April 2000), The P versus NP Problem (PDF), Clay Mathematics Institute.).

We denote by \( \Sigma^* \)all the words of an alphabet \( \Sigma \).

**Definition 4.1** A language \( L \) of binary words is in the class NP if and only if the next conditions hold

1) There is a deterministic Turing machine \( M \) that decides \( L \). In other words for any word \( x \) in \( L \), when \( x \) is given as input to \( M \), then \( M \) accepts it and if \( x \) does not belong to \( L \) then \( M \) rejects it. In symbols: \( \exists \) a deterministic Turing machine \( M \), such that \( \forall x \in \Sigma^* \), \( x \) is either accepted or rejected by \( M \) and if \( M \) accepts \( x \rightarrow x \in L \), and if \( M \) reject \( x \rightarrow x \notin L \).

2) There is a polynomial-time checkable relation \( R(x,y) \), and a natural number \( k \) of \( N \), so that for every word \( x \), \( x \) belongs to \( L \) if and only if there is a word \( y \), with \( |y| \leq |x| \), and \( R(x,y) \) holds. In symbols:\( \exists \) relation \( R \) which is polynomial-time checkable , and \( \forall x \in \Sigma^* \), \( x \in L \leftrightarrow (\exists y \in \Sigma^*, \ |y| \leq |x| \text{ and } R(x,y) \text{ holds}) \).

**Remark 4.1.** In the official statement of the P versus NP problem (see [6] Cook, Stephen (April 2000), The P versus NP Problem (PDF), Clay Mathematics Institute) the condition 1) is not mentioned. But anyone that has studied complexity theory, knows that it is required. The condition 2) alone cannot guarantee that there is a deterministic Turing machine that decides the language, as the polynomial checkable relation works only if we provide it with certificate \( y \), and not with only \( x \) as input. Indeed we shall see below at the end of the proposition in **Remark 4.4**, that there is even an undecidable language \( L \), for which nevertheless there is a polynomial checkable relation \( R \), so that condition R is satisfied. The languages of NP cannot be semidecidable (or undecidable). The NP class is also defined as \( \text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k) \), but this definition is also in the context of non-deterministic Turing Machines. The situation
with P, is more clear, because the mere requirement that a language of P is of polynomial time complexity as it standard to define it, involves already that there exist a deterministic Turing machines that for every input word, it halts within polynomial time steps and either accepts or rejects it, therefore it decides it. And not that the language of a deterministic Turing machine, and therefore maybe only semi-decidable.

**Remark 4.2.** Notice that in the condition 2) the k depends on the relation R and is not changing as the certificate y changes. In other words k does not depend on y and we did not state the next:

There is a polynomial-time checkable relation \( R(x,y) \), so that for every word \( x \), \( x \) belongs to \( L \) if and only if there is a word \( y \), and \( k \) in \( N \) with \( |y| \leq |x| \) and \( R(x,y) \) holds. In symbols: \( \exists \) relation \( R \) which is polynomial-time checkable, such that \( \forall x \in \Sigma^* \), \( x \in L \leftrightarrow (\exists y \in \Sigma^* \text{ and } 3k \in N \text{ such that } |y| \leq |x|^k \text{ and } R(x,y) \) holds).

In the official statement of the P versus NP problem (see [6] Cook, Stephen (April 2000). The P versus NP Problem (PDF), Clay Mathematics Institute) this is not made clear, in the natural language that the definition is stated. But that k does not depend on the certificate, but on the polynomial checkable relation becomes clear, when we look at the proof in any good textbook about complexity theory, of how a non-deterministic Turing machine which runs in polynomial time, can define a deterministic Turing machine with a polynomial time checkable relation, which is considered that replaces it.

**Proposition 4.1.** There is at least one infinite binary sequence, that can be computed and decided in exptime-complete complexity.

**Proof.** Let an exptime-complete decision problem \( A \), that its existence is guaranteed by Proposition 3.1, and its language \( L_{exp} \in \text{EXPTIME} \). We will need for the sake of symbolic convenience this language and decision problem, in the form of a binary sequence. If \( \Sigma \) is the set of all words of the binary alphabet \( \Sigma \) of the language \( L_{exp} \), then we give a linear order to the binary alphabet \( \Sigma = \{0,1\} \) 0<1, and then the inherited linear lexicographic order to the set of words \( \Sigma^* \). Since \( \Sigma \) is linearly and well ordered with a first element and after excluding all words with a left sequence of consecutive zeros (which is obviously a polynomial time decision on the length of the words) reducing to the set denoted by \( \Sigma^* \), we fix the identity map as an arithmetization with an 1-1 and on to correspondence \( F: \Sigma^* \rightarrow \mathbb{N} \) to the set of natural numbers, so that the language \( L_{exp} \) can be considered after this fixed arithmetization identity mapping correspondence \( F \), as a subset of the natural numbers. So let \( \text{Char}(L_{exp}) : \mathbb{N} \rightarrow \{0,1\} \) be the characteristic function of the set \( L_{exp} \) in the Natural numbers encoded thus in a binary base. Then \( \text{Char}(L_{exp}) \) consists of \( d_i \) for \( i \in \mathbb{N} \), and \( d_i \) is binary digit, that is equal to 0 or 1. A first finite 7-digits segment of it, would seem for example like (0010110...). Since \( L_{exp} \) is an exptime-complete decision problem \( L_{exp} \in \text{EXPTIME} \), its characteristic function is computable with an exptime-complexity too on the length of the binary words, and conversely any Turing machine computation of this characteristic function and also infinite binary sequence \( \text{Char}(L_{exp}) : \mathbb{N} \rightarrow \{0,1\} \) consisting from \( d_i \) for all \( i \in \mathbb{N} \), and \( d_i \) is binary digit, that is equal to 0 or 1, is also a Turing machine decision computation of the language \( L_{exp} \). Therefore there is no polynomial time complexity computation of this infinite binary sequence, as this would make \( \text{EXPTIME} \subseteq \text{P} \) and we know that \( \text{P} \neq \text{EXPTIME} \). For the sake of intuitive understanding of the following arguments we call this binary sequence “An exptime-complete binary DNA sequence” and we denote by \( \text{DNA}_{exp} \). This simplification from the original exptime-complete decision problem and Language \( L_{exp} \) of \( \Sigma \) to the \( \text{DNA}_{exp} \) of \( \mathbb{N} \) can be considered also as a polynomial time reduction of decision problem and languages \( L_{exp} \leq_p \) \( \text{DNA}_{exp} \) (L_{exp} is p-reducible to \( \text{DNA}_{exp} \))(see Definition 2.1). QED.

**Proposition 4.2 (3rd Clay Millennium problem)** There is at least one decision problem language of the class NP which is not also in the class P.

**Proof.** In the next we show that there is a language \( L_{exp} \) belonging to the class NP that cannot also belong to the class P without making, the previous binary sequence in the proof of the Proposition 4.1 called “exptime-complete binary DNA sequence” and denoted by \( \text{DNA}_{exp} \), computable in polynomial time complexity! To ensure that a language \( L_{exp} \) belongs to the class NP it must hold that there is a polynomial-time checkable relation \( R(x,y) \) and a natural number \( k \), so that for every word \( x \), it holds that \( x \) belongs to the language \( L_{exp} \) if and only if there is another word \( y \), called “certificate” with length
Now comes the intuition behind calling the binary sequence DNA_{exp} of the previous proof, a DNA sequence: The trick here is to define this language denoted by L_{np} with the information encoded in the binary sequence DNA_{exp} so that, although a human with deterministic Turing machines and exptime-time complexity can compute DNA_{exp} and therefore decide L_{np}, no deterministic Turing machine within polynomial-time complexity can compute and decide the L_{np}. In addition for every word x, if a human will give to such deterministic machines the necessary information in the form of a “certificate” y, then a deterministic Turing machine can decide if x belongs or not to L_{np} within polynomial-time complexity.

We define such a language L_{np} with the previous requirements simply as follows:
For any word x ∈ Σ, x ∈ L_{np} if and only if, the d_{|x|} = 1, d_{|x|} ∈ DNA_{exp}, in other words, the |x| order digit d_{|x|} of the infinite binary sequence DNA_{exp} is equal to 1. And of course x does not belong to L_{np} if and only if the d_{|x|} = 0, d_{|x|} ∈ DNA_{exp}, in other words, the |x| order digit d_{|x|} of the infinite binary sequence DNA_{exp} is equal to 0.

Then we define “certificate” y of the word x, the finite sequence y=(d_1, d_2,...,d_{|x|}), and as polynomial time checkable relation R(x,y), and that R(x,y) holds, the fact that given x, and y, the last digit of y is 1. Notice that here a human gives a lot of information to a Turing machine that will check if x belongs or not to L_{np}, in the form of the |x|-length initial segment y of the infinite binary sequence DNA_{exp} that we know that no Turing machine can compute within polynomial-time complexity.

That this relation R(x,y) is checkable in polynomial time relative to the length |x| of x, is obvious as the Turing machine with input x and y, will have only go through |x|-many steps to check the last digit of y. Now no deterministic Turing machine M can decide the language L_{np}, in other words decide given as input only the word x (without its “certificate” y), if x ∈ L_{np} or not. And this is so, because if it exist such a deterministic Turing machine M, then it could also decide (or compute) the digit d_{|x|} of DNA_{exp} which we know that is not computable in polynomial-time complexity. Thus L_{np} does not belong to P, and therefore P ≠ NP

**Remark 4.3** A very interesting question here is if the language L_{np} in the previous proof, is also exptime-complete in complexity, since the information of the DNA_{exp} on which it is defined is also exptime-complete. Such a fact would immediately derive that NP=EXPTIME.

**Remark 4.4** Notice that instead of taking the characteristic function DNA_{exp} of an exptime-complete language, we could have taken the characteristic function DNA_{und} of an undecidable language and we know that, there is at least one, and repeat the definition of the Language L_{np}, deriving thus an undecidable language, which still it has a polynomial time checkable relation, that nevertheless works only if a human feeds it with a certificate y and there is not a Turing machine that can decide it by taking as input the word x alone. This confirms that in the definition of NP in, Definition 4.1, the condition 1) is required. Alternatively we may prove the same thing in a different way. By using the axiom of choice of the ZFC set theory we may define for example an arbitrary infinite sequence L_p of passwords p_n, each one of length exactly n, from the infinite set of the sets of words Σ^n of length n. It is known that the axiom of choice of the ZFC set theory, gives no information at all about what are the elements of such a set, besides that each p_n belongs to Σ^n. We cannot expect that any such infinite choice L_p of n-length passwords p_n can be decided by a deterministic Turing machine. If it was so, as such Turing machines are countable, we order all such languages L_{p,n}, i∈N in a sequence and with the diagonal method we define a new and different such language L_0 of passwords, differing to at least one password from all those L_{p,i}, thus this L_0 is undecidable. Still again there is a polynomial-time checkable relation R(x,p_{|x|}), which simply is checking if x= p_{|x|} so that for every word x, there is a “certificate”, here the password p_{|x|}, and x belongs to the language L_0 of passwords iff R(x,p_{|x|}) holds.

5. Conclusions
Sometimes great problems have relatively short and elegant solutions provided we find the convenient context, symbols and semantics to solve them. But even relatively simple paths of reasoning, may be difficult to travel, if there are not the right conceptual “coins” of symbols and semantics to exchange and convert. If the P versus NP problem is researched without a main strategy, that would require a short proof, it might become a very complex problem to solve. My main hidden guiding idea in
searching for such a simple proof, was that what the “arbitrary human-like free-will” of a non-deterministic Turing machine as human-machine interactive software, can do in polynomial time cannot be done by a purely mechanical deterministic Turing machine in polynomial time. Since in my opinion the Hierarchy Theorem is a deeper result than the P versus NP problem, in principle there should exist a not much more complicated proof of the P versus NP problem, compared to the proof of the Hierarchy Theorem. The proof of the P versus NP problem in the direction P ≠ NP, is supposed also to mean that the standard practice of encryption in the internet, is safe.

References


[21] Tepliz Miron 2005 *Sigma-notation and the equivalence of P and NP classes* Journal of Information and Organizational Sciences (JIOS) Faculty of Organization and Informatics.


