Global Journal of Computer Science and Technology: G Interdisciplinary
Volume 21 Issue 3 Version 1.0 Year 2021
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

P vs NP: P is Equal to NP: Desired Proof

By Zulfia A. Chotchaeva
Moscow State University

Abstract- Computations and computational complexity are fundamental for mathematics and all computer science, including web load time, cryptography (cryptocurrency mining), cybersecurity, artificial intelligence, game theory, multimedia processing, computational physics, biology (for instance, in protein structure prediction), chemistry, and the P vs. NP problem that has been singled out as one of the most challenging open problems in computer science and has great importance as this would essentially solve all the algorithmic problems that we have today if the problem is solved, but the existing complexity is deprecated and does not solve complex computations of tasks that appear in the new digital age as efficiently as it needs. Therefore, we need to realize a new complexity to solve these tasks more rapidly and easily. This paper presents proof of the equality of P and NP complexity classes when the NP problem is not harder to compute than to verify in polynomial time if we forget recursion that takes exponential running time and goes to regress only (every problem in NP can be solved in exponential time, and so it is recursive, this is a key concept that exists, but recursion does not solve the NP problems efficiently). The paper's goal is to prove the existence of an algorithm solving the NP task in polynomial running time. We get the desired reduction of the exponential problem to the polynomial problem that takes O(log n) complexity.

Keywords: P vs. NP, P=NP, computational complexity, NP-complete problems, exponential running time.

GJCST-G Classification: I.2.8

Strictly as per the compliance and regulations of:

P vs NP: P is Equal to NP: Desired Proof

Zulfia A. Chotchaeva[1]

Abstract- Computations and computational complexity are fundamental for mathematics and all computer science, including web load time, cryptography (cryptocurrency mining), cybersecurity, artificial intelligence, game theory, multimedia processing, computational physics, biology (for instance, in protein structure prediction), chemistry, and the P vs. NP problem that has been singled out as one of the most challenging open problems in computer science and has great importance as this would essentially solve all the algorithmic problems that we have today if the problem is solved, but the existing complexity is deprecated and does not solve complex computations of tasks that appear in the new digital age as efficiently as it needs. Therefore, we need to realize a new complexity to solve these tasks more rapidly and easily. This paper presents proof of the equality of P and NP complexity classes when the NP problem is not harder to compute than to verify in polynomial time if we forget recursion that takes exponential running time and goes to regress only (every problem in NP can be solved in exponential time, and so it is recursive, this is a key concept that exists, but recursion does not solve the NP problems efficiently). The paper's goal is to prove the existence of an algorithm solving the NP task in polynomial running time. We get the desired reduction of the exponential problem to the polynomial problem that takes O(log n) complexity.

Keywords: P vs. NP, P=NP, computational complexity, NP-complete problems, exponential running time.

I. Introduction

Another mention of the underlying problem occurred in a … letter written by Kurt Gödel to John von Neumann. Gödel asked whether theorem-proving (now known to be co-NP-complete) could be solved in quadratic or linear time, and pointed out one of the most important consequences – that if so, then the discovery of mathematical proofs could be automated (Wikipedia, 2021).

The P vs. NP problem remains one of the most important problems in computational complexity. Until now, the answer to that problem is mainly "no". And this is accepted by the majority of the scientific world. What is the P versus NP problem, and why should we care? The question is represented as P=?NP. P-class problems take polynomial time to solve a problem (less time), NP-class problems take "non-deterministic" polynomial time to quickly check a problem (more time), therefore, P problems are easier to solve while NP problems are harder. NP-complete problems are the hardest and take more time than P-class problems. If P=NP, we could find solutions to search problems as easily as checking since a solution for any NP-class problem can be recast into a solution for any other problem of this class. Thus, finding the efficient algorithm would prove that P=NP and revolutionize (completely turn) many fields in mathematics and computer science. "The development of mathematics in the direction of greater exactness has – as well known – led to large tracts of it becoming formalized so that proofs can be carried out according to the few mechanical rules (Gȍdel, 1931)." "Perhaps in most cases where we seek in vain the answer to the question, the cause of the failure, lies in the fact that problems are simpler and easier than the one in hand have been either not at all or incompletely solved (Hilbert, 2000)." "Do NP-complete languages exist? It may not be clear that NP should process a language that is as hard as any other language in the class. However, this does turn out to be the case (Arora and Barak, 2009)." All previous attempts to solve the problem did not lead to the desired solution. But we declare that the desired solution exists. The paper will get easy proof of the equality of complexity classes P and NP through (with) the new computational complexity that takes polynomial running time and completely rearranges these complexity classes (we will get an exponential-time reduction to polynomial time using a sorted array). The paper intends to prove that the use of logarithmic looping of matrices through a sequence of matrix loops replaces recursive iterating that takes $O(2^n)$ with a completely new and another method (approach) that is more efficient and faster than existing and takes O(log n) complexity instead of $O(2^n)$ when we solve the NP task. (Notice, we will not compare this work and its methods (operators) with what has been done before, because it is so different from everything that already exists that it simply makes it impossible, for instance, like the Boolean satisfiability problem (SAT), the Cook-Levin theorem, the Curry-Howard isomorphism, the Davis-Putnam algorithm, the Davis-Putnam-Logemann-Loveland procedure, the Karp-Lipton theorem, and others that are conversions of the listed, that is, have nothing in common what lies at the basis of these approaches, except for the fractional differentiation, but it does not rely on old, previously

Author: Moscow State University, Krasniy Kurgan 369387, Russia.*
e-mail: lesya.chotchaeva8@gmail.com
**The self-study*

known algorithms, methods, principles, concepts, or models, this light tutorial is completely new and will easily refute the unsolvability, or intractability, of the P vs. NP problem. More precisely, to change the NP, we needed to change the P, i.e., we use a polynomial-time reduction that is the perfect way to provide (get) the reducibility and computability of NP that make the problem of NP the problem of P.)

II. Literature Review

a) Background

The P versus NP problem is a major unsolved problem in computer science. P-complexity is a deterministic polynomial, we consider this complexity class as $O(n^c)$, where the base is variable, and the exponent of the base is constant; and NP-complexity is a nondeterministic polynomial, we consider this complexity class as $O(c^n)$, where the base is constant, and the exponent of the base is variable. The polynomial and exponential time complexities are the most prominently considered and define the complexity of an algorithm. The question is - whether every problem whose solution can be quickly verified in polynomial running time can be solved quickly in polynomial running time too? If NP-complete problems were efficiently solvable, it could advance considerably the solution of other complex problems.

"The precise statement of the P versus NP problem was introduced in 1971 by Stephen Cook in his seminal paper "The complexity of theorem-proving procedures". …Although the P versus NP problem was formally defined in 1971, there were previous inklings of the problems involved, the difficulty of proof, and the potential consequences…The relation between the complexity classes P and NP are studied in computational complexity theory, the part of the theory of computation dealing with the resources required during computation to solve a given problem. The most common resources are time (how many steps it takes to solve a problem) and space…the class P consists of all those decision problems that can be solved on a deterministic sequential machine in an amount of time that is polynomial in the size of the input; the class NP consists of all those decision problems whose positive solutions can be verified in polynomial time … on a non-deterministic machine. …To attack the P=NP question, the concept of NP-completeness is very useful. … NP-complete problems are a set of problems to each of which any other NP-problem can be reduced in polynomial time and whose solution may still be verified in polynomial time. … Based on the definition alone, it is not obvious that NP-complete problems exist… The first natural problem proven to be NP-complete was the Boolean satisfiability problem, also known as SAT… However, after this problem was proved to be NP-complete, proof by reduction provided a simpler way to show that many other problems are also NP-complete, including the game Sudoku…a polynomial-time to Sudoku leads, by a series of mechanical transformations, to a polynomial-time solution of satisfiability, which in turn can be used to solve any other NP problem in polynomial time… In 1975, R. E. Ladner showed that if P≠NP, then there exist problems in NP that are neither in P or NP-complete. Such problems are called NP-intermediate problems. The graph isomorphism problem, the discrete logarithm problem, and the integer factorization problem are examples of problems believed to be NP- intermediate. …P means "easy" and "not in P" means "hard", an assumption known as Cobham's thesis. It is a common and reasonably accurate assumption in complexity theory; …There are algorithms for many NP-complete problems, such as the knapsack problem, the traveling salesman problem, Boolean satisfiability problem that can solve to optimality many real-world instances in reasonable time…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 (Wikipedia, 2021)".

III. Methodology

a) Definition of the Task

Any NP class problem can be solved by exhaustive search of all instances, i.e., by brute force search that requires exponential execution time, this is unacceptable in practice, therefore, we need to solve the NP problems in polynomial time, and if one of these NP problems is solved in polynomial time, then the others will also be solved in polynomial time. To solve the task where the worst-case run-time on an input of size n is $O(n^n)$ that have the highest growth rate, i.e., is greater than exponential and factorial time complexities that take $O(2^n)$ and O(!), we need to transform this task from infinitely exponential complexity class to polynomial complexity class using logarithmic looping of n^n if the value of n^n is explicit, but even then, when this task is solved, it will have no practical use as it leads to infinity only. Therefore, we need to solve the task of exponential time complexity that takes $O(2^n)$ to get the P vs. NP problem solution.

Exponential runtime complexity $O(2^n)$ is often seen in recursive functions that make 2 recursive calls that mean that growth doubles with each addition to the input data set (every problem in NP is recursive, and every recursive problem is recursively enumerable). Let us take, for example, a set with n elements, where we need to find (generate) all subsets of this set (the set theory is commonly used as a foundational system for the whole of mathematics and has various applications in computer science; its implications for the concept of infinity and its multiple applications have made set

theory a field of major interest; current research into the set theory covers a vast array of topics, ranging from the real number line structure to the study of the consistency; many mathematical concepts can be defined precisely using only set theory concepts). There are three ways to find the number of subsets of a set $S=\{a_0, a_1, \ldots, a_{n-1}\}$. The Tower of Hanoi is $O(2^n)$, as the expression of the trend we see would be $2^0+2^1+2^2+2^3+2^{n-1}=2^n-1$ that takes exponential running time. The second way is to translate between the binary representation of the rank and the subset when 1 means the corresponding element is in the subset, and 0 means the element is not in the subset, see below:

$$\text{We take } S=\{a_0, a_1, a_2\}.$$

Subsets of a given set:

0 000 { } the empty subset
1 001 $\{a_0\}$
2 010 $\{a_1\}$
3 011 $\{a_0, a_1\}$
4 100 $\{a_2\}$
5 101 $\{a_0, a_2\}$
6 110 $\{a_1, a_2\}$
7 111 $\{a_0, a_1, a_2\}$

As a result, we get exponential running time that will rising meteorically if we will add n elements to this set. And the third way, let us consider a set with 5 elements:

$$S=\{1, 2, 3, 4, 5\}.$$

What is the number of all possible and proper subsets of a given set with these 5 elements? There are 2^n subsets and 2^n-1 proper subsets that means that the number of all subsets of a set is 2^5 and the number of proper subsets is 2^5-1. To determine the Big-O runtime complexity, we do not need to look at how many recursive calls are made (iterating over all possible subsets of a set) since we will not deal with Fibonacci trees, it will be used only the task of the recursive Fibonacci number calculation that is $O(2^n)$, as the certain patterns in the recurrence relation lead to exponential results too (exponential time grows much faster than polynomial time). Therefore, we will get this using a new time complexity that works without a return (we capture one of the NP tasks in a sequence of matrix loops that runs in polynomial time and hack its secret arrangement without recursion). You need to read the paper at https://doi.org/10.3844/jcssp.2020.1610.1624 that is published recently and gets O(log n) complexity instead of $O(n^2)$ before continuing this reading since we will use this O(log n) complexity to solve this exponential task in polynomial running time (read this paper instead of the Methodology section, you can start reading at once from the end to clarify faster how it works, more exactly, see Lemma 21 and then other lemmas).

Let's continue if you have read. We will solve this NP problem using the new matrix model of computation concept and prove that this is a perfect path for its solution.

Lemma 1.0. The use of the NP task 2^n partitioning into 2^2 particles is a key for this NP task solution.

Proof. We need to generate a set of matrices to find the number of all subsets of this set with five elements that is $S=\{1, 2, 3, 4, 5\}$ (see above), where are 2^5 subsets that are equal to $2^2 \cdot 2^2 \cdot 2^1$. Each of these 2^2 particles gives one complete matrix. The matrices look like these matrix loops:

$$\text{Loop1,2,3}=\begin{pmatrix}2 & 2\\ 8 & 8\end{pmatrix}, \begin{pmatrix}2 & 2\\ 8 & 8\end{pmatrix}, \begin{pmatrix}2\\ 8\end{pmatrix},$$

where we have inserted these previous 2^2 particles of our partitioned task that is $2^5=2^2 \cdot 2^2 \cdot 2^1$ in an array as one of the options of this array to find the number of all possible subsets of a given above set, and the set of these matrices represents this decomposition of $2^5=2^2 \cdot 2^2 \cdot 2^1$, and each of them works to find these 2^2 particles, note that matrix Loop1,2,3 is not complete since the number of elements of the given set is odd, therefore, Loop 1,2,3 not works completely and carried over this incomplete matrix that is $\begin{pmatrix}2\\ 8\end{pmatrix}$ to the following loops; we are moving ahead only (without using backtracking to find all subsets), i.e., we do not need to iterate recursively (return), we take the result obtained by the first matrix loop and drag it to another matrix loop till we get to finish (terminate), and as we move ahead through the matrix loops, we cut the work at least in half and are closer to finding the last result, that is how we proceed, and further, we receive this:

$$\text{Loop2,3}=\begin{pmatrix}4 & 4\\ 6 & 6\end{pmatrix}, \begin{pmatrix}2\\ 8\end{pmatrix},$$

and finally, we have

$$\text{Loop3,0}=\begin{pmatrix}16 & 2\\ 84 & 98\end{pmatrix},$$

where the total $a=LE=2^5=(2\cdot2)\cdot(2\cdot2)\cdot2=(4\cdot4)\cdot2=16\cdot2=32=a_3$.

Remark 1.0. There is a reference map of these matrices that is:

$$\begin{pmatrix}L & E\\ I & T\end{pmatrix},$$

In the first loop, $L=E=2$, $I=T=sbasis-L=sbasis-E=8$, $sbasis=10$, $LE=4=a_1$, in the second loop, $L=E=4=a_1$, $I=T=sbasis-L=sbasis-E=6$,

sbasis=10, LE=16=a_2, in the third loop, L=16=a_2, E=2, I=sbasis-L=84, T=sbasis-E=92, sbasis=100, LE=32=a_3. We are interested only in the (a) options values, as all these elements of a given set are inserted on a position of (a)=LE options in this array after partitioning them into 2^2 equal particles, therefore, it is not necessary to determine the values of T elements, they can be dropped since these values will not be used in the main algorithm below. We use this algorithm for these 2^2 particle's logarithmic looping:

$$T(n)=(E:2\text{-}(E:2:(sbasis:(I\text{-}L))))\cdot sbasis$$

that provides the following

- Loop1,2,3 gives a_1=20:2-(20:2:(100:(80-20)))=4=2^2 - for Loop1,2 that goes to (4·4)·2
- Loop2,3 gives a_2=40:2-(40:2:(100:(60-40)))=16=4^2 - for Loop2 that goes to 16·2
- Loop3,0 gives a_3=200:2-(200:2:(10000:(8400-1600)))=32=16·2 - for Loop3

That means that the number of all subsets of a set is 32, including the empty subset, and the number of proper subsets is 32-1=31.

Remark 2.0. Keep in mind that we not only do not return to the matrix loop, where we already have received the result, we find the value of (a)=LE only for one complete matrix of each matrix loop since all complete matrices of each matrix loop are the same (they are copies).

Remark 3.0. Compare these steps with the following:

S={1, 2, 3, 4, 5}.

Subsets of a given above set:

{ }, {1}, {2}, {3}, {4}, {5}, {12}, {13}, {14}, {15}, {23}, {24}, {25}, {34}, {35}, {45}, {123}, {124}, {125}, {134}, {135}, {145}, {234}, {235}, {245}, {345}, {1234}, {1235}, {1245}, {1345}, {2345}, {{12345}.

Imagine how many returns (repeating moves) you will need to make to find all subsets of a set when the number of elements in a set is 20, 30, 80, etc.

Let's go further.

Lemma 2.0. The partitioning of the NP task 2^n into 2^2 particles remains a key for this NP task solution when the exponent of 2^n grows.

Proof. Suppose we need to find all subsets of a set with eight elements that is S={1, 2, 3, 4, 5, 6, 7, 8}, where are 2^8 subsets. The number of elements in this set is even, therefore, all matrices of the matrix Loop1,2.3.4 are complete. Further we have:

$$Loop1,2,3,4 = \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}, \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}, \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}, \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix},$$

then

$$Loop3,4 = \begin{pmatrix} 4 & 4 \\ 6 & 6 \end{pmatrix}, \begin{pmatrix} 4 & 4 \\ 6 & 6 \end{pmatrix},$$

and finally,

$$Loop4,0 = \begin{pmatrix} 16 & 16 \\ 84 & 84 \end{pmatrix}.$$

That is, there are 256 subsets in a given set.

Remark 1.1. Let us take a look at a visual model of this task that gives the scheme below. We have the following:

(2·2) (2·2) (2·2) (2·2) — Loop1,2,3,4, where L=E=2, I=8=sbasis-2, a_1=2·2=4

(4 · 4) (4 · 4) — Loop3,4, where L=E=4, I=6=sbasis-4, $a_2=a_1^2$=4·4=16

(16 · 16) — Loop4,0, where L=E=16, I=84=sbasis-16, $a_3=a_2^2$=16·16=256

256

The number of all subsets of a set that is 2^8 is equal to 256.

Theorem 1.0.

Regardless of how large the exponent of 2^n is, a sequence of matrix loops runs in polynomial time solving this exponential-time task, that means that an upper bound on the worst-case running time of this 2^n task is O(log n).

Corollary 1.0. We get a sequence of matrix loops that runs in polynomial time when we define the value of 2^n.

Proof

Let's go further and take a set with 30 elements, where we need to find all possible subsets of this set. The number of all subsets of this set is 2^{30}, and we have the following:

$$Loop1, \dots, 15 = \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}^{15\ complete},$$

then

$$Loop7{,}5, \dots, 15 = \begin{pmatrix} 4 & 4 \\ 6 & 6 \end{pmatrix}^{7\ complete}, \begin{pmatrix} 4 \\ 6 \end{pmatrix},$$

and then

$$Loop3{,}75, \dots, 15 = \begin{pmatrix} 16 & 16 \\ 84 & 84 \end{pmatrix}^{3\ complete}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 4 \\ 6 \end{pmatrix},$$

further,

$$Loop1{,}875, \dots, 15 = \begin{pmatrix} 256 & 256 \\ 744 & 744 \end{pmatrix}^{1\ complete}, \begin{pmatrix} 256 \\ 744 \end{pmatrix}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 4 \\ 6 \end{pmatrix},$$

and,

$$Loop1{,}875, \dots, 15 = \begin{pmatrix} 65536 & 256 \\ 34464 & dropped \end{pmatrix}, \begin{pmatrix} 16 & 4 \\ 84 & 96 \end{pmatrix},$$

and, finally,

$$Loop15, 0 = \begin{pmatrix} 16777216 & 64 \\ 83222784 & dropped \end{pmatrix}.$$

That is, there are 1 073 741 824 subsets in this set with 30 elements. We use this algorithm below for matrices of each matrix loop:

$$a{=}LE{=}(E{:}2{-}(E{:}2{:}(sbasis{:}(I{-}L)))){\cdot}sbasis$$

that takes O(log n) complexity.

Corollary 2.0. A sequence of matrix loops runs in polynomial running time when the exponent of 2^n increases and becomes larger.

Proof

As we need to estimate the asymptotic complexity of this 2^n task, let us consider, for instance, a set with 89 elements, where the number of all possible subsets is 2^{89}. We need to partition the 2^{89} into 2^2 particles, where are 44 complete and 1 incomplete matrices in the initial matrix loop (note that all incomplete matrices are carried to the following matrix loops until there are no complete matrices, then they are sequentially enclosed in additional matrix loops on the position of the (a) options in matrices), and we have the following:

$$Loop1, \dots, 44{,}5 = \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}^{44\ complete}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

then

$$Loop22{,}25, \dots, 44{,}5 = \begin{pmatrix} 4 & 4 \\ 6 & 6 \end{pmatrix}^{22\ complete}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

further,

$$Loop11{,}125,\ldots,44{,}5 = \begin{pmatrix} 16 & 16 \\ 84 & 84 \end{pmatrix}^{11\ complete}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

and

$$Loop5{,}5625,\ldots,44{,}5 = \begin{pmatrix} 256 & 256 \\ 744 & 744 \end{pmatrix}^{5\ complete}, \begin{pmatrix} 256 \\ 744 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

we have

$$Loop2{,}78125,\ldots,44{,}5 = \begin{pmatrix} 65536 & 65536 \\ 34464 & 34464 \end{pmatrix}^{2\ complete}, \begin{pmatrix} 65536 \\ 34464 \end{pmatrix}, \begin{pmatrix} 256 \\ 744 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

and then,

$$Loop1{,}390625,\ldots,44{,}5 = \begin{pmatrix} 4294967296 & 4294967296 \\ 5705032704 & 5705032704 \end{pmatrix}^{1\ complete}, \begin{pmatrix} 16777216 & 2 \\ 83222784 & 8 \end{pmatrix},$$

and finally,

$$Loop44{,}5{,}0 = \begin{pmatrix} 18446744073709551616 & 33554432 \\ 81553255926290448384 & dropped \end{pmatrix}.$$

This matrix looping gives a sequence that runs in polynomial time.

Loop1, …,44,5 gives the value of the (a_1) option for all complete matrices of this matrix loop, that is (2 · 2), and goes to the following matrix loop as the value of (L· E)=(4 · 4), and defines the number of all complete and incomplete matrices for the following matrix loop that is the Loop22,25, …,44,5 this number is (44,5:2)=22,25.

- a_1=(2:2-(2:2:(10:(8-2)))·10=4

Loop22,25, …,44,5 gives the value of the (a_2) option for all complete matrices of this matrix loop, that is (4 · 4), and goes to the following matrix loop as the value of (L · E)=(16 · 16), and defines the number of all complete and incomplete matrices for the following matrix loop that is the Loop11,125, …,44,5, i.e., (22,25:2)=11,125.

- a_2=(4:2-(4:2:(10:(6-4)))·10=16

Loop11,125, …,44,5 gives the value of the (a_3) option for all complete matrices of this matrix loop, that is (16 · 16), and goes to the following matrix loop as the value of (L· E)=(256 · 256), and defines the number of all complete and incomplete matrices of the Loop5,5625, …, 44,5, i.e., (11,125:2)=5,5625, etc.

- a_3=(16:2-(16:2:(100:(84-16)))·100=256
- a_4=(256:2-(256:2:(1000:(744-256)))·1000=65536
- a_5=(65536:2-(65536:2:(100000:(34464-65536)))·100000=4294967296
- a_6=(4294967296:2-(4294967296:2:(10000000000:(5705032704-4294967296)))·sbasis=2147483648-(2147483648:(10000000000:1410065408)))·10000000000=18446744073709551616
- a_7=(33554432:2-(33554432:2:(100000000000000000000:(81553255926290448384-18446744073709551616)))·sbasis=(16777216-(16777216:(100000000000000000000:631065118525808967 68)))·100000000000000000000=618970019642690137449562112=2^{89}

The number of all subsets of this set with 89 elements is 618970019642690137449562112.

Corollary 3.0. The sequence of matrix loops runs in polynomial time when the exponent of 2^n becomes extremely large.

Proof

Let us consider a set where the number of elements of this set is much larger than in previous sets. We take the set with 4117 elements, the number of all subsets of this set is 2^{4117} , and we have the following sequence of matrix loops:

$$Loop1,\dots,2058,5 = \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}^{2058\ complete}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

where we get the value of $a_1=(L \cdot E)= 2 \cdot 2=4$, (see the reference map of these matrices above), then,

$$Loop1029,25,\dots,2058,5 = \begin{pmatrix} 4 & 4 \\ 6 & 6 \end{pmatrix}^{1029\ complete}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

where we get the value of $a_2=a_1 \cdot a_1=16$, further,

$$Loop514,625,\dots,2058,5 = \begin{pmatrix} 16 & 16 \\ 84 & 84 \end{pmatrix}^{514\ complete}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

and this matrix loop gives the value of $a_3=a_2 \cdot a_2=256$, and then,

$$Lopp257,3125,\dots,2058,5 = \begin{pmatrix} 256 & 256 \\ 744 & 744 \end{pmatrix}^{257\ complete}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

then we get the value of $a_4=a_3 \cdot a_3=65536$, further,

$$Loop128,65625,\dots,2058,5 = \begin{pmatrix} 65536 & 65536 \\ 34464 & 34464 \end{pmatrix}^{128\ complete}, \begin{pmatrix} 65536 \\ 34464 \end{pmatrix}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

and we have the value of $a_5=a_4 \cdot a_4=4294967296$, and further,

$$Loop64,328125,\dots,2058,5 = \begin{pmatrix} 4294967296 & 4294967296 \\ 5705032704 & 5705032704 \end{pmatrix}^{64\ complete}, \begin{pmatrix} 65536 \\ 34464 \end{pmatrix}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

and we get the value of $a_6=a_5 \cdot a_5=18446744073709551616$, then,

$$Loop32,1640625,\dots,2058,5 =$$

$$= \begin{pmatrix} 18446744073709551616 & 18446744073709551616 \\ 81553255926290448384 & 81553255926290448384 \end{pmatrix}^{32\ complete}, \begin{pmatrix} 65536 \\ 34464 \end{pmatrix}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

and we have the value of $a_7=a_6 \cdot a_6=340282366920938463463374607431768211456$, further,

$$Loop16,08203125,\dots,2058,5 =$$

$$= \begin{pmatrix} 340282366920938463463374607431768211456 & 340282366920938463463374607431768211456 \\ 659717633079061536536625392568231788544 & 659717633079061536536625392568231788544 \end{pmatrix}^{16\ complete},$$

$$\begin{pmatrix} 65536 \\ 34464 \end{pmatrix}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

then we get $a_8=a_7 \cdot a_7=115792089237316195423570985008687907853269984665640564039457584007913129639936$, and,

$$Loop8,041015625,\dots,2058,5 =$$

$$= \begin{pmatrix} 115792089237316195423570985008687907853269984665640564039457584007913129639936 & 1\dots \\ 884207910762683804576429014991312092146730015334359435960542415992086870360064 & 8\dots \end{pmatrix}^{8\ complete}, \begin{pmatrix} 65536 \\ 34464 \end{pmatrix}, \begin{pmatrix} 16 \\ 84 \end{pmatrix}, \begin{pmatrix} 2 \\ 8 \end{pmatrix},$$

where we have $a_9=a_8 \cdot a_8$=1340780792994259709957402499820584612747936582059239337772356144372176403007354697680187429816690342769003185818648605085375388281194656994643364900608409 6 ..., note that the value of L element is always equal to E element and the value of I element is always equal to T element if these elements are the elements of one of the complete matrices of each matrix loop, further,

$$Loop4{,}0205078125,\ldots,2058{,}5 = \left(\begin{matrix}1340780792994259709957402499820584612747936582059239337772356144372176403007354697680187429816690342769003185818648605085375388281194656994643364900608409 6 \ldots \\ 8659219207005740290042597500179415387252063417940760662227643855627823596992645302319812570183309657230996814181351394914624611718805343005356635099391590 4 8 \ldots\end{matrix}\right)^{4\ complete},$$

$$\binom{65536}{34464},\binom{16}{84},\binom{2}{8},$$

and further, we get $a_{10}=a_9 \cdot a_9$=17976931348623159077293 0…224137216, the value of a_{10} is long enough to write it in full like the following values of a, then,

$$Loop2{,}0802534906425,\ldots,2058{,}5 = \left(\begin{matrix}1797693134862315907729330224137216 & 1797693134862315907729330224137216 \\ 1797693134862315907729330224137216 & 1797693134862315907729330224137216\end{matrix}\right)^{1\ complete},\binom{65536}{34464},\binom{16}{84},\binom{2}{8},$$

and then we have $a_{11}=a_{10} \cdot a_{10}$, and,

$$Loop1{,}005126953125 = \left(\begin{matrix}a_{11} & a_{11} \\ sbasis-a_{11} & sbasis-a_{11}\end{matrix}\right)^{1\ complete},\binom{65536}{34464},\binom{16}{84},\binom{2}{8},$$

and finally, we get the value of $a_{12}=a_{11} \cdot a_{11}$, where L=$a_{11}$, E=L=$a_{11}$, I=sbasis-L=sbasis-$a_{11}$, T=I=sbasis-$a_{11}$, then,

$$Loop2058{,}5{,}0 = \left(\begin{matrix}a_{12}=a_{11} \cdot a_{11}=(a_{11})^2 & (65536 \cdot 16 \cdot 2) \\ sbasis-a_{12} & dropped\end{matrix}\right).$$

We have the value of a_{total} that is equal to the value of ($a_{12} \cdot 65536 \cdot 16 \cdot 2$) = $2^{4\,111\,7}$, where 65536, 16 and 2 are the values from those incomplete matrices that were carried over all loops and the value of T element is dropped, we use this algorithm a=(E:2- (E:2:(sbasis:(I-L)))) to find these long values of all (a) options, and thus, regardless of how large the exponent of 2^n is, a sequence of such matrix loops runs in polynomial time.

Asymptotic analysis of the runtime of an algorithm that we use to find the value of (a) option for each complete matrix of these matrix loops is presented below.

Run-time analysis: Prove that (E:2-(E:2:(sbasis:(I-L))))·sbasis= O(log n). Let T(n) be the execution time for the input of size n, where $\lim_{n\to\infty}$, there exist positive constants and lower order terms that are not considered and can be omitted, then:

- T(n)=(E:2-(E:2:(sbasis:(I-L))))sbasis
- $T(n)=T_1(n)+T_2(n)+T_3(n)+T_4(n)+T_5(n)+T_6(n)=f(n)$
- $T_1(n)$=sbasis-L=I⇒O(n)
- $T_2(n)$=I-L⇒O(n)
- $T_3(n)$=sbasis:(I-L)⇒O(n)
- $T_4(n)$=E:2:(sbasis:(I-L))⇒O(n)
- $T_5(n)$=E:2-(E:2:(sbasis:(I-L)))⇒O(n)
- $T_6(n)$=(E:2-(E:2:(sbasis:(I-L))))·sbasis⇒can be omitted (dropped)

Let *f* and *g* be functions from positive numbers to positive numbers, where *f*(n)=(E:2-(E:2:(sbasis:(I-L))))·sbasis=O(n) and g(n)=O(log n). Prove the claim that *f*(n) is O(g(n)) if there exist positive constants c>0 and n_o>0 such that:

$$f(n) \le c*g(n) \text{ for all } n \ge n_0.$$

To prove big-O, we choose values for c and n_o and prove n>1 implies f(n)≤c*g(n):

1. Choose n_o=1,
2. Assuming n>1, find/derive a c such that:

$$\frac{f(n)}{g(n)} \le \frac{cg(n)}{g(n)} = c$$

that proves that n>1 implies f(n)≤c*gn. This means that function f(n) does not grow faster than g(n), or that function g(n) is an upper bound for f(n) for all sufficiently large $n \to \infty$.

An algorithm asymptotic running time is O(log n).

Notice. The value of sbasis is always equal to 10^n, therefore, we consider this value as an easy constant factor, and the I element is the 10's complement of the L element, therefore, it runs very quickly when we define the value of (sbasis-L).

Comparing the asymptotic running time:

An algorithm that runs in O(n) time is better than one that runs in O(2^n), and O(log n) is better than O(n).

Theorem 2.0.

It is enough to decompose n^n into the set of n^2 particles (fractions) to find the value of any n^n since there is an easy algorithm that solves the exponential-time task as the task that runs in polynomial time, i.e., we will turn (transform) NP to P using O{log n) complexity that will provide an easy solution for every n^2 particle of this set.

Proof

As any n^n can be easily decomposed (partitioned) into $n^2 \cdot n^2 \cdot \ldots \cdot n^2 \cdot n^1$ if the exponent of n^n is an odd number, and into $n^2\ n^2 \cdot \ldots\ n^2 \cdot n^2$ if the exponent of n^n is an even number, that we can find in logarithmic time using O(log n) complexity, hence we can find n^n in polynomial time, that means that P=NP. For example, it

is obvious, as we are aware, that $3^5=3^2\cdot3^2\cdot3^1$, or $5^6=5^2\cdot5^2\cdot5^2$, etc. The constant factors of this new algorithm will remain sustainable (steady) and scalable when n^n grows and goes to infinity. Let us consider the following:

Given a set (an array) of positive integers in matrix form:

$$\begin{pmatrix} n_1^n & n_2^n \\ n_3^n & n_4^n \end{pmatrix},$$

and let, for instance, $n_1=3$ and $n_2=2$, then $n_3=sbasis-n_1=7$ and $n_4=sbasis-n_2=8$, the values of the n_3 and n_4 elements of an array are the complements of the n_1 and n_2 elements, the sbasis=10, suppose we need to find the exponential values of these elements, when $n_1^n= 3^5$ and $n_2^n= 7^5$, the bases and the exponents of the n^n elements are taken arbitrarily, and the current sbasis is successive, then:

$$\begin{pmatrix} 3^5 & 2^5 \\ 7^5 & 8^5 \end{pmatrix},$$

and further, we decompose this array into this matrix of n^2 particles:

$$\begin{pmatrix} 3^2\cdot3^2\cdot3^1 & 2^2\cdot2^2\cdot2^1 \\ 7^2\cdot7^2\cdot7^1 & 8^2\cdot8^2\cdot8^1 \end{pmatrix}.$$

We got the decomposition of this n^n task into n^2 particles that transforms the exponential time to a polynomial that uses the new O(log n) complexity for $O(n^2)$, i.e., for these n^2 particles solving. Further, we make matrix loops for each of these n^2 particles that look like this:

$$\begin{pmatrix} 3 & 3 \\ 7 & 7 \end{pmatrix}\cdot\begin{pmatrix} 3 & 3 \\ 7 & 7 \end{pmatrix}\cdot\begin{pmatrix} 3 \\ 7 \end{pmatrix} \text{ and } \begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}\cdot\begin{pmatrix} 2 & 2 \\ 8 & 8 \end{pmatrix}\cdot\begin{pmatrix} 2 \\ 8 \end{pmatrix}.$$

To solve these matrix loops, we will also use this algorithm:

$$a=n_1^n\cdot n_2^n=(n_2^n:2-(n_2^n:2:(sbasis:(n_3^n-n_1^n))))\cdot sbasis,$$

The reference matrix for n^n looks like this:

$$\begin{pmatrix} {n_1}^2\cdot{n_1}^2\cdot\ldots\cdot{n_1}^2\cdot n_1 & {n_2}^2\cdot{n_2}^2\cdot\ldots\cdot{n_2}^2\cdot n_2 \\ {n_3}^2\cdot{n_3}^2\cdot\ldots\cdot{n_3}^2\cdot n_3 & {n_4}^2\cdot{n_4}^2\cdot\ldots\cdot{n_4}^2\cdot n_4 \end{pmatrix}$$

for n^n, where the exponent of this n^n is an odd, and

$$\begin{pmatrix} {n_1}^2\cdot{n_1}^2\cdot\ldots\cdot{n_1}^2\cdot{n_1}^2 & {n_2}^2\cdot{n_2}^2\cdot\ldots\cdot{n_2}^2\cdot{n_2}^2 \\ {n_3}^2\cdot{n_3}^2\cdot\ldots\cdot{n_3}^2\cdot{n_3}^2 & {n_4}^2\cdot{n_4}^2\cdot\ldots\cdot{n_4}^2\cdot{n_4}^2 \end{pmatrix}$$

for n^n, where the exponent of this n^n is an even. These all are easy to check using any random instance of n^n.

IV. Results and Discussion

The major result of this paper is that $O(2^n)=O(\log n)$, that means that P=NP. Easy to solve (to find), easy to check (to verify), don't you think? This is a study that changes our understanding of a topic. We had to go beyond the rules for this. And it is easier than you think. There is no decision problem (a yes-no question) for the NP problem anymore, and we do not need the certificate for this, since we have simplified and eliminated all that was complicated multiple times over by various wrong theories and their numerous modifications, we no longer even need the SAT. The NP tasks do not require making two recursive calls when growth doubles with each addition to the input data set, we have a new path to solve this problem in polynomial running time using a sequence of matrix loops that uses a sorted array and takes O(log n) complexity. We get the desired reduction of the exponential problem to the polynomial problem. There are some known definitions of the P vs. NP problem that will be read in a new way in the future: 'The P versus NP problem is to determine whether every language accepted by some nondeterministic algorithm in polynomial time is also accepted by some (deterministic) algorithm in polynomial time (Cook, 2000)." 'P versus NP – a gift to mathematics from computer science (Smale, 2000)." "It is interesting to recall that the motivation for the development of the theory of computation, on which theoretical computer science is based, came from purely mathematical considerations. The paradox is emerging from Cantor's set theory emphasized the need to clarify the foundations of mathematics and, under Hilbert's leadership, concentrated attention on axiomatic proof systems. The quest to understand the power and limitations of axiomatizable systems led directly to the questions about all possible formal mechanical ways of deriving proofs (sequences with desired properties). In modern terms, it led to the search for what is and is not effectively computable (Hartmanis, 1989)." "The hope that mathematical methods employed in the investigation of formal logic would lead to purely computational methods for obtaining mathematical theorems goes back to Leibniz… (Davis & Putnam,1959)." "Your definition of experiments by using point-sets is perfectly satisfactory to me, I thought, however, that it might be good to say explicitly that a computation may be part of an "observation" (Neumann, 2005)." "The most comprehensive formal systems yet set up are, on the one hand, … and, on the other, the axiom system for set theory…These two systems are so extensive that all methods of proof used in mathematics today have been formalized in them (Gödel, 1931)." "Occasionally it happens that we seek the solution under insufficient hypotheses or in an incorrectly sense (Hilbert, 2000)." "The principal technique used for demonstrating that two problems are

related is that of "reducing" one to the other, by giving a constructive transformation that maps any instance of the first problem into an equivalent instance of the second (Garey, 1979)." "…any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be reduced to the problem of determining whether a given proposition formula is a tautology (Cook, 1971)." "The class of languages recognizable by string recognition algorithms which operate in polynomial time is also invariant under a wide range of changes in the class of algorithms (Karp, 1972)." 'Due to the fact that no NP-complete problem can be solved in polynomial time… (Crescenzi & Kann, 1994)." "I offer a personal perspective on what it's about, why it's reasonable to conjecture that P≠NP is both true and provable… (Aaronson, 2011)." "…we can avoid brute - force search in many problems and obtain polynomial-time solutions. However, attempts to avoid brute force in certain other problems, including many interesting and useful ones, haven't been successful, and polynomial-time algorithms that solve them aren't known to exist (Sipser, 2012)." "As we solve larger and more complex problems with greater computational power and cleverer algorithms, the problem we cannot tackle begin to stand out (Fortnow, 2009)." "In recent years, the reducibility of computation in real environments to the standard Turing model has been brought increasingly into question (Cooper, 2004)." "The subject my talk is perhaps most directly indicated by simply asking two questions: first, is it harder to multiply than to add? and second, why? I grant I have put first of these questions rather loosely; nevertheless, I think the answer ought to be: yes. It is the second, which asks for a justification of this answer which provides the challenge (Cobham, 1965)." "Most of the computational problems that arise in practice turn out to be complete for one of a handful of complexity classes, even under very restrictive notions of reducibility (Agrawal, Allender, Impagliazzo, Pitassi, & Rudich, 2001)." "At present, when faced with a seemingly hard problem in NP, we can only hope to prove that it is not in P assuming that NP is different from P. (Goldreich, 2008)." "…an algorithm is any well-designed computational procedure that takes some value, or set of values, as input and procedures some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output (Cormen, Lieserson, & Rivast, 2009)." "It is well known that every set in P has small circuits. Adelman was recently proved the stronger result that every set accepted in polynomial time by a randomized Turing machine has small circuits (Lipton & Karp, 1980)." 'I see complexity as the intricate and exquisite interplay between computation (complexity classes) and applications (that is, problem) (Papadimitriou, 1994)." "We do not know of polynomial-time algorithms for these problems, and we cannot prove that polynomial-time algorithms exist…These are the NP-complete problems… (Kleinberg & Tardos, 2006)." "…there is a strictly ascending sequence with a minimal pair of upper bounds to the sequence…if P≠NP then there are members of NP-P that are not polynomial complete (Ladner, 1975)." "…minimal propositional logic corresponds to dependent simply typed-calculus… (Sorensen & Urzyczyn, 1998)." 'Practical problems requiring polynomial time are almost solvable in an amount of time that we can tolerate, while those that require exponential time generally cannot be solved except for small instances (Hopcroft, Motwani, & Ullman, 2001)." "Some success was had by causing the machine to systematically eliminate the redundancy; but the problem of total length increasing rapidly still remained when more complicated problems were attempted (Davis, Logemann, & Loveland, 1961)." "Gȍdel and others went on to show that various other mathematically interesting statements, besides the consistency statement, are undecidable by P, assuming it to be consistent… (Boolos, Burgess, & Jeffrey, 2007)." "There has been much work in getting the number of variables needed for an undecidability result to be small (Gasarch, 2021)."

V. Conclusion

It is possible to solve the exponential-time task in polynomial time if we forget recursion that takes $O(2^n)$ complexity and goes to regress only. As you see, it is clear that the new notion of the decision procedure for the NP problem exists. We have a completely new definition of the certificate for this NP problem that can not only be checked in polynomial time but also solved in polynomial time. And there is no case when this new uniform procedure is not valid, the algorithm terminates with a correct answer on any input instance of 2^n and does not involve seeking forever (without Halting problem, without approximation), i.e., this new certificate is consistent, therefore, we solve this NP task rapidly, accurately, and easily using this unthinkably easy computational tactic above.

Acknowledgment

I would like to express my special thanks of gratitude to the reviewers for their thoughtful comments and the effort and expertise that they contribute to review the manuscript.

Funding Information

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflict of Interest

The corresponding author has NO conflicts of interest to disclose.

Ethics

This article is original and contains unpublished material.

References Références Referencias

1. Aaronson, S. (2017). P= ?NP. (PDF)
2. Agrawal, M., Allender, E., Impagliazzo, R., Pitassi, T., & Rudich, S. (2001). Reducing the complexity of reductions. Birkhauser Verlag, Basil. Computational complexity, 10 (2001), 147-138. (PDF)
3. Arora, S., & Barak, B. (2009). Computational complexity: a modern approach. Cambridge University Press.
4. Boolos, G. S., Burgess J. P., & Jeffrey R. C. (2007). Computability and logic. Cambridge University Press.
5. Cobham, A. (1965). The intrinsic computational difficulty of functions. (PDF)
6. Cormen, I., Leiserson, C, Rivast, R., & Stein, C. (2009). Introduction to algorithms. The MIT Press. Cambridge, Massachusetts. London, England. (PDF)
7. Cook, S. A. (1971). The complexity of theorem-proving procedures. University of Toronto. (PDF)
8. Cook, S. A. (April 2000). "The P versus NP Problem". Clay Mathematics Institute. Retrieved 18 October 2006.
9. Cooper, S. B. (2004). Computability theory. University of Leeds, U. K. (PDF) Crescenzi, P., & Kann, V. (1994). A compendium of NP optimization problems. (PDF)
10. Chotchaeva, Z. A. (2020). The new matrix model of computation based purely on quite a new concept of the matrix computations for extremely quick web pages loading. Journal of Computer Science, 16(11), 1610-1624.
11. Davis, M., & Putnam, H. (1959). A computing Procedure for Quantification theory. Journal of the ACM, Volume 7.
12. Davis, M., Logemann, G., & Loveland, D. (1961). A Machine Program for theorem proving. Communications of the ACM, Volume 5, Issue 7.
13. Fortnow, L. (2009). The status of the P vs NP. Communications of the ACM, 52(9); 78-86. Garey, M. R. & Johnson, O. S. (1979). Computers and Intractability (Vol. 174). San Francisco: freeman.
14. Gasarch, W. (2002). Hilbert's Tenth Problem: Refinements and Variants. arXiv:2104.07220v2 [math.LO]
15. Gȍdel, K. (1931). On formally undecidable propositions of principia mathematics and related systems. (PDF)
16. Goldreich, O. (2010). P, NP, and NP-Completeness: The Basics of Computational Complexity. Cambridge University Press.
17. Hartmanis, J. (1989). Gödel, von Neumann, and the P=?NP problem. Bulletin of the European Association for Theoretical Computer Science. 38: 101-107.
18. Hilbert, D. (2000). Mathematical problems, AMS, Volume 37. (PDF)
19. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). Introduction to automata theory, languages, and computations. Addison-Wesley. (PDF)
20. Karp, R.M. (1972). Reducibility among Combinatorial Problems. DOI: 10.1007/978-3-540-68279-0_8, Springer, 2010.
21. Kleinberg, J., & Tardos, E. (2005). Algorithm design. Cornell University. (PDF)
22. Ladner, R. E. (1975). On the structure of Polynomial-time reducibility. Journal of the ACM. Volume 23, Issue 1.
23. Lipton, R. J., & Karp. R. M. (1980). Some connections between nonuniform complexity classes. Proceedings of the ACM, 302-309.
24. Morten Heine B. Sorensen, & Pawel Urzyczyn (1998). Lectures on the Curry-Howard Isomorphism. University of Copenhagen & Warsaw.
25. Neumann, von J. (2005). Selected letters. History of mathematics, Volume 27. (PDF)
26. Papadimitriou, C. (1993). Computational Complexity. Addison-Wesley.
27. Sipser, M. (2012). Introduction to the Theory of Computation. Cengage learning.
28. Smale, S. (2000). Mathematical problems for the next century. The mathematical intelligencer 20 (2). DOI:10.1007/BF03025291
29. Wikipedia. (2021). P versus NP problem. Wikipedia.