The manuscript below corresponds to my talk presented to the mathematics Colloquium, but with references added later. The references are by no means complete, particularly with regard to books that I could not conveniently locate. I want to thank Dan Burghelea for inviting me to give a talk on P = NP (as part of a series of lectures on the Clay Millenium Problems at Ohio State University) as this is not something that I would normally do.

CLAY MILLENIUM PROBLEM:
P = NP

by
Harvey M. Friedman
Mathematics Colloquium
Ohio State University
October 20, 2005

The equation P = NP concerns algorithms for deciding membership in sets. The consensus is that P ≠ NP, although some prominent experts guess otherwise.

P = NP is one of many questions of the form: If there is an algorithm for determining membership in a given set using specified limited resources, is there automatically such an algorithm using yet more limited specified resources?

P = NP sits in discrete complexity theory - based on sets of finite strings in a finite alphabet of letters. Sets of integers, sets of rationals, and sets of finite sequences of such, are all treated as special cases of sets of finite strings.

Discrete complexity theory is dependent on a formal treatment of algorithms - initiated in the 1930's by Alan Turing.

The official problem description [Co] contains much detailed information that will not be mentioned here.

1. TURING MACHINES.
2. SOME R.E. COMPLETENESS.
3. TIME, SPACE, NONDETERMINISM.
4. SOME NP COMPLETENESS.
5. OPINIONS.

1. TURING MACHINES.

There are many sources that discuss Turing machines. E.g., [Tu36], [Tu37], [Ro67], [Co71], [Da73], [AHU74], [HU79], [GJ79], [Od89], [Pa94], [Da00], [CLR01], [Si05], [Co], and many dozens of others.

Turing machines form the most common primitive model of computation used in theoretical studies. Micro Center does not have Turing machines for sale.

TM's process symbols from a finite alphabet. When given a finite string from a finite alphabet as input, it grinds away, perhaps halting after finitely many steps, or perhaps going on forever.

TM's have a two way infinite tape divided into squares indexed by the integers, and a reading head that always sits on exactly one square of tape.

Every TM comes with a finite set of distinct symbols $a_0,\ldots,a_k,b_0,\ldots,b_n$, $k,n \geq 0$. $\{a_0,\ldots,a_k\}$ is the input alphabet. $b_0,\ldots,b_n$ are the auxiliary symbols. b_0 is the "blank".

Every TM also comes with a finite set of distinct states $q_0,\ldots,q_m$, $m \geq 2$. q_0 is the starting state, q_1 is "accept", and q_2 is "reject".

Every TM comes already programmed. The program consists of a finite list of instructions that tell the reading head what to do.

Each instruction takes the following form:

If the reading head sees symbol x and TM is in state q, then replace x with symbol x', put TM into state q', and move the reading head one square to the left (right).

Note that x = x' and q = q' are allowed.

TM's are deterministic - there is exactly one such instruction for each symbol and state other than q_1,q_2. (Nondeterministic Turing machines are called NDTM's).

For nonempty A, A* is the set of all finite strings from A (empty string allowed).

A finite input string □ □ $\{a_0,..., a_k\}^*$, is given.

The TM is starts by writing □ on the tape starting with square 1. The remaining squares have the blank, b_0. The reading head is placed on square 1. The starting state is q_0.

Instructions are successively followed. If final state is q_1 then TM "accepts" the input string □. If final state is q_2 then TM "reject" □. If there is no final state (q_1 or q_2) then TM neither accepts nor rejects □.

DEFINITION. Let S □ A*. S is recursively enumerable (r.e.) if and only if there is a TM with input alphabet A, such that S = {x □ A*: x is accepted by the TM}.

DEFINITION. Let S □ A*. S is recursive if and only if there is a TM with input alphabet A, such that S = {x □ A*: x is accepted by the TM} and every x □ A* is accepted or rejected by the TM.

THEOREM 1.1. Every recursive subset of A* is r.e., but not vice versa.

THEOREM 1.2. S □ A* is recursive if and only if S and A*\S are r.e.

The above two Theorems are discussed in virtually any place that discusses Turing machines without resource bounds (before computational complexity theory with resource bounds).

Adaptation to **Z** is by identifying nonzero integers with their base 2 expansion as a string in {0,1}, and 0 with the empty string. Related identifications work for **Q**, and for finite sequences from **Z** or **Q**.

The TM model is a crude specialized model that can be augmented in many ways. More tapes, more reading heads, etc. RAM machines are more like what they sell at Micro Center than TM's.

Early work was devoted to simulating more powerful models by the TM model. The recursive and r.e. sets don't change.

CHURCH'S THESIS. Any reasonable model of discrete computation can be simulated in the TM model.

The notions of recursive and r.e. subsets of A* are remarkably robust.

Promising efforts to "prove" Church's Thesis exist.

The Turing machines themselves can be identified with finite strings in a fixed finite alphabet. Thus we can consider

1. The set of all TM's that halt at the empty input string.
2. The set of all TM's that don't halt at the empty input string.
3. The set of all TM's that halt at every input string.

THEOREM 1.3. Set 1 is r.e. but not recursive. Set 2 is co-r.e. but not r.e. Set 3 is not a Boolean combination of r.e. sets.

Again, the above two Theorems are discussed in virtually any place that discusses Turing machines without resource bounds (before computational complexity theory with resource bounds).

How "hard" can membership in an r.e. set be?

To make sense of this, we need "recursive functions".

Any TM provides a partial function f:A* □ A* as follows. If TM does not halt with input x □ A*, then f(x) is undefined. If TM does halt with input x □ A* then, after halting, read the input symbols from left to right, skipping over the auxiliary symbols.

These partial f:A* □ A* are called the "partial recursive functions". These f:A* □ A* (i.e., dom(f) = A*) are called the "recursive functions".

Let S,T □ A*. We write S $\square_r$ T if and only if there exists a recursive f:A* □ A* such that

$$x \square S \square f(x) \square T.$$

THEOREM 1.4. There exists an r.e. $T \subseteq A^*$ such that for all r.e. $S \subseteq A^*$, $S \leq_r T$. Set 1 above has this property. There exists a nonrecursive r.e. $T \subseteq A^*$ without this property.

Yet again, the above two Theorems are discussed in virtually any place that discusses Turing machines without resource bounds (before computational complexity theory with resource bounds).

We call this property r.e. completeness. It obviously implies nonrecursiveness (undecidability).

THEOREM 1.5. Let $S,T \subseteq A^*$ be complete r.e. There exists a recursive permutation $f:A^* \rightarrow A^*$ such that $f[S] = T$.

See [My55], [Ro67], [Od89].

2. SOME R.E. COMPLETENESS.

The most familiar sets of strings, integers, rationals, finite sequences of integers, or rationals, are easily recursive.

But here is a simple example of a set of integers not known to be recursive.

$$S = \{n-m: n,m \text{ prime}\}.$$

By standard techniques, S is r.e.

Here is an example from group theory.

Let E be a finite set of group equations in distinct letters $g_1,...,g_n$. We ask:

for all groups G and $g_1,...,g_n \in G$
obeying these equations,
do we have $g_1 = ... = g_n = e$?

This question asks if the group presented by the equations is trivial.

THEOREM 2.1. This triviality problem is complete r.e.

Note that the data can live naturally in {0,1,•,-,e,;,=,(,)}. Here strings of 0's and 1's for the generators, • for the multiplication, - for the inverse, e for the identity, and ; to separate the equations.

See [No55], [Bo59], [BDL73].

Here is a closely related example from topology.

One can construct from a group presentation of G, a 2-dimensional simplicial complex whose fundamental group is G. Since a simplicial complex is simply connected if and only if its fundamental group is trivial, we have

> it is not recursive (undecidable) whether a given 2-dimensional simplicial complex is simply connected.

See [Ha73]. For further results along these lines, see, e.g., [Mar58], [BCL73], and [BHP68].

We now discuss the existence of zeros.

THEOREM 2.2. There is an algorithm for deciding whether a given polynomial in several variables with integer coefficients has a zero in the reals.

See [Ta51], [Re92a], [Re92b], [Re92c], [BPR03].

THEOREM 2.3. There is no algorithm for deciding whether a given polynomial in several variables with integer coefficients has a zero in the integers. This decision problem is complete r.e.

The proof was completed in 1970. This is sometimes called the MRDP theorem, as [Ma70] relied substantially on earlier efforts of three others. See [DPR61], [Ma70], and expositions in [Da73], [Ma93]. Also see [Da77]. Also see the discussion in [Ma93] concerning attempts to limit the number of variables and the degree needed for this negative result. Our knowledge is extremely poor in this regard.

WIDE OPEN. Is there an algorithm for deciding whether a given polynomial in several variables with integer coefficients has a zero in the rationals?

Now let W be the least class of functions in one variable, that can be written in terms of 1,+,-,•,sin and variables.

THEOREM 2.4. There is no algorithm for deciding whether a given function from W has a zero in the reals. This problem is complete r.e.

Ranges of polynomials are particularly relevant.

THEOREM 2.5. Let A □ N. A is r.e. iff A = $P[Z^k]$ □ N for some polynomial P of k variables with integer coefficients.

THEOREM 2.6. If P has k variables and degree □ 2, with integer coefficients, then $P[Z^k]$ is recursive.

This follows from [Sie72] on quadratic Diophantine equations.

Even Theorem 2.3 is wide open for 2 variable cubics.

3. TIME, SPACE, NONDETERMINISM.

There are many sources that discuss Turing machines with resource bounds. The key phrase to search for is "complexity theory" or "computational complexity theory". One can be more specific with "discrete computational complexity theory", as there is a continuous theory that is a horse of a different color. For discrete complexity theory, see [Co71], [AHU74], [HU79], [GJ79], [Pa94], [CLR01], [Si05], [Co], and many other places.

A TM is said to run in polynomial time if and only if there exists $n \geq 1$ such that at all input strings x,

TM cannot make > $(|x|+2)^n$ moves

where $|x|$ is the length of x.

(We make this definition in this form so that we don't have to change it for the nondeterministic case. In this deterministic case, this simply means that at all x, TM makes □ $(|x|+2)^n$ moves).

A TM is said to run in polynomial space if and only if there exists $n \geq 1$ such that at all input strings x,

TM cannot visit > $(|x|+2)^n$ squares of tape.

(We make this definition in this form so that we don't have to change it for the nondeterministic case. In this deterministic case, this simply means that at all x, TM visits □ $(|x|+2)^n$ squares).

Let S □ A*. We say that S is decidable in polynomial time if and only if there is a TM running in polynomial time, with input alphabet A, where S = {x □ A*: x is accepted by TM}.

We say that S is decidable in polynomial space if and only if there is a TM running in polynomial space, with input alphabet A, such that S = {x □ A*: x is accepted by TM}.

WIDE OPEN. If S is decidable in polynomial space then is S decidable in polynomial time?

P consists of all decision problems solvable in polynomial time. PSPACE consists of all decision problems solvable in polynomial space.

$$P = PSPACE?$$

is wide open.

Recall the TM instructions

If the reading head sees symbol x and the TM is in state q, then replace x with symbol x′, put TM into state q′, and move the reading head one square to the left (right).

We required: exactly one such instruction for each symbol and state, other than q_1,q_2.

In a NDTM, we only require that there be at least one such instruction for each symbol and state, other than q_1,q_2.

A NDTM is initialized in the same way as a TM with input string x, but it can choose which of possibly many applicable instructions to follow at each stage.

It is an old classical result that if we use NDTM's instead of TM's in our definition of recursive and r.e. sets, then we will have the same recursive and r.e. sets.

However, once resource bounds are placed on computation, nondeterminism versus determinism becomes a very difficult issue that, in many contexts, remains entirely unresolved.

NDTM runs in polynomial time if and only if there exists $n \geq 1$ such that at all input strings x,

$$\text{NDTM cannot make} > (|x|+2)^n \text{ moves.}$$

NDTM runs in polynomial space if and only if there exists $n \geq 1$ such that at all input strings x,

$$\text{NDTM cannot visit} > (|x|+2)^n \text{ squares of tape.}$$

NDTM accepts the input string x iff it can reach state q_1 following some path of instructions.

NDTM rejects the input string x if and only if it can reach state q_2 following some path of instructions.

Let $S \subseteq A^*$. We say that S is decidable in nondeterministic polynomial time if and only if there is an NDTM running in polynomial time with input alphabet A such that $S = \{x \in A^*: x \text{ is accepted by NDTM}\}$.

We say that S is decidable in nondeterministic polynomial space if and only if there is an NDTM running in polynomial space with input alphabet A such that $S = \{x \in A^*: x \text{ is accepted by NDTM}\}$.

NP consists of all decision problems solved in nondeterministic polynomial time.

NPSPACE consists of all decision problems solved in nondeterministic polynomial space.

THEOREM 3.1. $P \subseteq NP \subseteq PSPACE = NPSPACE$.

See [Sa70], [HU79], [Pa94], [Si05].

Most experts believe that the two inclusions are proper. In particular, that $P \neq NP$.

I think most experts believe that it is substantially easier to show $P \neq PSPACE$ than $P \neq NP$, although even this seems so hard that almost nobody admits they are working on it.

How hard is it to solve an NP problem? To make sense of this, we need polynomial time functions.

Any TM running in polynomial time provides a function f:A* □ A* as follows. Let x □ A*. Run the TM until halting. Then read the input symbols from left to right, skipping over the auxiliary symbols.

These are the polynomial time computable functions.

Let S,T □ A*. We write S $\square_p$ T if and only if there exists a polynomial time f:A* □ A* such that

x □ S □ f(x) □ T.

THEOREM 3.2. There exists an NP T □ A* such that for all NP S □ A*, S $\square_p$ T.

We call this property NP completeness.

Useful characterization of NP:

THEOREM 3.3. Let S □ A*. Then S □ NP if and only if there exists T □ A*, T □ P, and n ≥ 1, such that S = {x: (□y)((x,y) □ T □ |y| □ $(|x|+2)^n$)}.

See [Co].

THEOREM 3.4. If some NP complete problem is in P, then all NP problems are in P.

So the NP complete problems stand and fall together.

THEOREM 3.5. Every NP, or even PSPACE problem can be solved in deterministic exponential time. I.e., O(2^{n^k}), for some k.

This is a straightforward counting argument, where the exponential deterministic algorithm searches over the relevant configurations.

4. SOME NP COMPLETENESS.

nSAT, n ≥ 1.

$\pm p_{11}$ ... $\pm p_{1n}$
$\pm p_{21}$ ... $\pm p_{2n}$
...
$\pm p_{k1}$ ... $\pm p_{kn}$.

In the above, there are k clauses, each of length n. Repetitions of p's are of course expected.

Each p_{ij} is p_x for some bit string x.

Is there a truth assignment to the p's so that all clauses come out true?

THEOREM 4.1. nSAT is NP complete if n ≥ 3. 2SAT is in P.

See [Co71], [AHU74], [HU79], [GJ79], [Pa94], [CLR01], [Si05], [Co]..

We now turn to graph theory.

G = (V,E), where V is a finite set of vertices, and E is a set of unordered pairs from V (edges).

k-colorability: Is there a coloring of the vertices of G with at most k colors?

THEOREM 4.2. kCOL is NP complete if k ≥ 3. 2COL is in P.

See [Ka72], [CLR01].

CLIQUE: Does G contain a clique of size k?

Here k is NOT fixed, and is part of the data. Obviously we can assume that k is at most the number of vertices of G, and so it is unimportant whether k is given in unary or binary.

THEOREM 4.3. CLIQUE is NP complete. For any fixed clique size k, it is in P.

See [Ka72], [GJ79], [CLR01].

SUBISO: Given two graphs G,H, is G isomorphic to a subgraph of H?

THEOREM 4.4. SUBISO is NP complete.

See [Co71].

TRAVSALES. Given m cities, with positive integer distance function between each unordered pair of cities, and positive integer n. Is there a tour of the cities where the total distance traveled is □ n?

Here n and distances are given in binary.

THEOREM 4.5. TRAVSALES is NP complete.

See [GJ79], [CLR01].

GRAPHISO. Given two graphs, are they isomorphic?

THEOREM 4.6. GRAPHISO is in NP. Not known if it is NP complete, or if it is in P.

THEOREM 4.7. GRAPHISO for planar graphs is in P. For all $k \geq 1$, GRAPHISO for graphs of valence □ k is in P.

See [HW74] and [Lu82].

Now for some number theory.

THEOREM 4.8. “n is prime”, n given in binary, is in P.

See [AKS04] and [Di04].

QUADCONG. Given integers $a,b,c > 0$, is there an integer $0 < x < c$ such that $x^2 \equiv a \bmod b$?

Here a,b,c given in binary.

THEOREM 4.9. QUADCONG is NP complete. In P for c = .

See [MA78].

QUADEQN. Given integers $a,b,c > 0$, are there integers $x,y > 0$ such that $ax^2 + by = c$?

THEOREM 4.10. QUADEQN is NP complete.

See [MA78].

SUBSUM. Given integers $a_1,\ldots, a_n,b > 0$, is b the sum of distinct a’s?

THEOREM 4.11. SUBSUM is NP complete.

See [Ka72], [CLR01].

SUBPROD. Given integers $a_1,...,a_n,b > 0$, is b the product of distinct a's?

THEOREM 4.12. SUBPROD is NP complete.

See [Ya78].

We now come to factoring. Here we do not have a decision problem.

FACTORING. Given integer $n > 0$, find a nontrivial factor of n if it exists.

THEOREM 4.13. If $P = NP$ then factoring can be done in polynomial time.

"Factoring in polynomial time" means: there exists a factoring function computable in polynomial time. I.e., an f:Z Z such that if $n > 0$ is composite then $f(n) > 0$ is a nontrivial factor of n.

WIDE OPEN. Can factoring be done in polynomial time? Is factoring NP complete (in an appropriate sense)?

A suitably practical algorithm for factoring would destroy a number of fashionable cryptographic protocols.

See [CLR01], p. 834-836.

5. OPINIONS.

William Gasarch published an opinion poll on $P = NP$ in [Ga02]. Gasarch reported on 100 responses.

The median guess as to when it would be solved was around 2050.

61 thought $P \neq NP$.
9 thought $P = NP$.
4 thought it could not be resolved in ZFC.
3 stated that it could be resolved using very concrete methods.
22 offered no opinion.

Gasarch writes in [Ga02]:

“Of those who answered, most thought P ≠ NP. Most of the 9 who thought that P = NP were respectable members of the community. All of them recognized that their opinion is a minority viewpoint. A few even said they took it just to be contrary. However, the fact that 22 people did not venture an opinion indicates more uncertainty on this then one would have thought”.

Here are some quotes published by Gasarch [Ga02] from some very well known people, plus me.

Bela Bollobas: 2020, P = NP. I think that in this respect I am on the loony fringe of the mathematical community. I I think (not too strongly) that P = NP and this will be proved within twenty years. Some years ago, Charles Read and I worked on it quite a bit, and we even had a celebratory dinner in a good restaurant before we found an absolutely fatal mistake. I would not be astonished if very clever geometric and combinatorial techniques gave the result, without discovering revolutionary new tools. A bit like Tim Gowers’s solutions to major sixty-year old questions of Banach. ... Sadly, we haven’t returned to the P vs NP question since that unfortunate experience fifteen years ago. The danger of wasting a year for no return is rather off-putting.

John Conway: 2030, P ≠ NP. In my opinion this shouldn’t really be a hard problem; it’s just that we came late to this theory, and haven’t yet developed any techniques for proving computations to be hard. Eventually, it will just be a footnote in the books.

Yuri Gurevich: 2060, P = NP. The positive solution will not make the standard NP problems worst-case feasible in the practical sense of the word.

Richard Karp: P ≠ NP. My intuitive belief is that P is unequal to NP, but the only supporting arguments I can offer are the failure of all efforts to place specific NP-complete problems in P by constructing polynomial-time algorithms. I believe that the traditional proof techniques will not suffice. Something entirely novel will be required. My hunch is that the problem will be solved by a young researcher who is not encumbered by too much conventional wisdom about how to attack the problem.

Donald Knuth: It will be solved by either 2048 or 4096. I am currently somewhat pessimistic. The outcome will be the truly worst case scenario: namely that someone will prove "P = NP because there are only finitely many obstructions to the opposite hypothesis"; hence there will exist a polynomial time solution to SAT but we will never know its complexity!

Laszlo Lovasz: 2017, P ≠ NP. Probably some new math modeling the information flow through a Boolean circuit. With luck, something like algebraic topology or algebraic geometry will be used.

Bob Tarjan: In my view, there is no way to even make intelligent guesses about the answer to any of these questions. If I had to bet now, I would bet that P is not equal to NP. I estimate the half-life of this problem at 25-50 more years, but I wouldn't bet on it being solved before 2100. Its solution will require unforeseen new techniques.

Jeff Ullman: 2100, P ≠ NP. I think the problem is comparable to some of the great problems of mathematics that lasted hundreds of years, e.g., the 4-color theorem. Thus, I'd guess 100 years. I'd bet we don't have the techniques, or even names for the techniques today. Again, that would be analogous to the situation for many of the great open problems of mathematics 30 years after they were posed.

Avi Wigderson: I think this project is a bit premature. I think we know too little of what is relevant to even guess answers to your questions... The only thing I can definitely say, is that it is one of the most important and interesting questions ever asked by humans, and more people and resources should participate in filling up the holes that would allow better guesses of answers to your questions.

Harvey Friedman: 2050, P ≠ NP. Detailed combinatorial work on easier problems, leading up to the full result. P = PSPACE will be refuted first.

REFERENCES

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison Wesley.

[AKS04] M. Agrawal, N. Kayal, N. Saxena, "PRIMES is in P." Annals of Mathematics 160(2): 781-793.

[Bo59] W.W. Boone, The word problem, Annals of Mathematics, 70, 207-265.

[BCL73] W.W. Boone, F.B. Cannonito, and R.C. Lyndon, Word Problems, North-Holland.

[BHP68] W.W. Boone, W. Haken, and V. Poenaru, On recursively unsolvable problems in topology and their classification, in: Contributions to Mathematical Logic, ed. H.A. Schmidt et al, North-Holland, 37-74.

[CLR01] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press.

[Co71] S.A. Cook, "The complexity of theorem-proving procedures," Proc. 3rd Ann. ACM Symp. on Theory of Computing, Assoc. for Computing Machinery, New York, 151-158.

[Co] S.A. Cook, P vs NP Problem, Clay Mathematics Institute, http://www.claymath.org/millennium/P vs NP/

[Da73] M. Davis, Computability and Unsolvability, Dover.

[Da77] M. Davis, Unsolvable problems, in: Handbook of Mathematical Logic, ed. J. Barwise, Studies in Logic and the Foundations of Mathematics, North-Holland.

[Da00] The Universal Computer: The Road from Leibniz to Turing, W. W. Norton and Company.

[Di04] Primality Testing in Polynomial Time: From Randomized Algorithms to "PRIMES Is in P", in Lecture Notes in Computer Science, Vol. 3000.

[DMR76] M. Davis, Y. Matiyasevich, J. Robinson, Hilbert's Tenth Problem, Diophantine equations: positive aspects of a negative solution, in: Mathematical Developments Arising from Hilbert's Problems, volume 28 of Proceedings of Symposia in Pure Mathematics, 323-378, Providence, Rhode island, American Mathematical Society.

[DPR61] M.D. Davis, H. Putnam, and J. Robinson, The decision problem for exponential Diophantine equations, Annals of Mathematics, Second series, 74(3):45-436.

[Ga02] W. Gasarch, The P=?NP Poll, in: SIGACT News Complexity Theory Column 36.

[GJ79] M.R. Garey and D.S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman.

[Ha73] W. Haken, Connections between topological and group theoretical decision problems, in [BCL73], 427-441.

[HU79] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison Wesley.

[Ka72] R.M. Karp, Reducibility among combinatorial problems, in R.E. Miller and J.W. Thatcher (eds.), Complexity of Computer Computations, Plenum Press, New York, 85-103.

[Le84] L. Levin, Universal sorting problems, Problems of Information Transmission 9, 1973, 265-266. See also Annals of the History of Computing, 6, 384-400.

[Lu82] E.M. Luks, Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time, J. Comput. System Sci., 25, 42-48.

[Ma70] Y.V. Matiyasevich, Enumerable sets are Diophantine, Soviet Mathematics, Doklady, 11(2), 354-358.

[Ma93] Y.V. Matiyasevich, Hilbert’s Tenth Problem, MIT Press.

[Mar58] A.A. Markov, Unsolvability of the problem of homeomorphy in: Proceedings of the International Congress of Mathematicians, Cambrdige University Press, 300-306.

[My55] J. Myhill, Creative sets, Zeitschrift fur mathematische Logik and Grundlagen der Mathematik, vol. 1, 97-108.

[MA78] K. Manders and L. Adelman, NP complete decision problems for binary quadratics, J. of Computing and Systems Sciences, **16**, 168-184.

[No55] P.S. Novikoff, On the algorithmic unsolvability of the word problem in group theory, Dokl. Akad. Nauk SSSR Mathematiceskii Institut Trudy, 44, Moscow.

[Od89] P. Odifreddi, Classical Recursion Theory by P. Odifreddi (North-Holland, Volume I, 1989, Volume II, 1999).

[Pa94] C.H. Papadimitriou, Computational Complexity, Addison Wesley.

[Re92a] J. Renegar, On the computational complexity and geometry of the first-order theory of the reals. I, J. Symbolic Computation, 13 (1992), no. 3, 255-299.

[Re92b] J. Renegar, On the computational complexity and geometry of the first-order theory of the reals. II, J. Symbolic Computation, 13 (1992), no. 3, 301-327.

[Re92c} J. Renegar, On the computational complexity and geometry of the first-order theory of the reals. III. Quantifier elimination. J. Symbolic Computation, 13 (1992), no. 3, 329-352.

[BPR03] S. Basu, R. Pollack, M.F. Roy, Algorithms in Real Algebraic Geometry, in: Algorithms and Computation in Mathematics, vol. 10, Springer Verlag.

[Ri68] D. Richardson, Some undecidable problems involving elementary functions of a real variable, Journal of Symbolic Logic, 33(4), 514-520.

[Ro67] H. Rogers Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill.

[Sa70] W.J. Savitch, Relationship between nondeterministic and deterministic tape complexities, J. of Computer and Systems Sciences **4**:2, 177-192.

[Si05] M. Sipser, Introduction to the Theory of Computation, Course Technology.

[Sie72] C.L. Siegel, Zur Theorie der quadratischen Formen, Nachrichten der Akademie der Wissenschaften in Göttingen, II, Mathematisch-Physikalische Klasse, no. 3:21-46.

[Ta51] A. Tarski, A decision method for elementary algebra and geometry, 2nd revised edition (Berkeley and Los Angles).

[Tu36] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proceedings of the London Mathematical Society, Second Series, 42:230-265.

[Tu37] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, A correction, Proceedings of the London Mathematical Society, Second Series, 43:544-546.

[Ya78] A.C. Yao, private communication, reference from [GJ79].