April 2021, UM.
Lectures in computational complexity. Very brief lecture notes.

Lecture 1. April 6.

1.1. The model of computation. We discusses the computational model called *a Turing machine*. There exists various versions of this model. We assume that a Turing machine contains one tape (infinite to the right). The Church–Turing thesis claims that every function that can be calculated by an "effective method" is also computable by a Turing machine.

The Church–Turing thesis is not a mathematical statement since it refers to an informal intuitive idea of arbitrary "effective methods." Formally, we say that a partial function $f : \{0,1\}^* \to \{0,1\}^*$ is *computable* if there is a Turing machine with the following property: if the machine gets as input (at the beginning of the tape) a string x from the domain of f, then its produces $f(x)$ (also at the beginning fo the tape) and stops; if the machine gets as input a string x that is not in the domain of f, then it never stops.

The *time* of run of a Turing machine (on a given input) is the number of elementary operations ("steps") before it stops. The *space* (used memory) of run of a Turing machine (also on a given input) is the number of cells on the tape visited by the machine at least once before it stops.

Exercise 1.1 (optional, not necessary for the final exam)**.** Construct Turing machines computing the following problems:

1. Return 1 for the inputs $x \in \{0,1\}^*$ that are palindromes and return 0 for non-palindromes.
2. Return 1 for the inputs $x \in \mathbb{N}$ (given by its binary expansion) that are divisibly by 3 and return 0 otherwise.
3. Compute the sum $a+b$ for a pair of numbers $a, b \in \mathbb{N}$ given by their binary expansions.
4. Compute the produce $a \cdot b$ for a pair of numbers $a, b \in \mathbb{N}$ given in the unary numeral system.
5. Compute the produce $a \cdot b$ for a pair of numbers $a, b \in \mathbb{N}$ given by their binary expansions.

For a more detailed discussion of Turing machines see [2]; definitions of a multi-tape Turing machine can be found also in [1, 3].

1.2. Hierarchy theorem.
Let us fix a computable function $t : \mathbb{N} \to \mathbb{N}$ and define a partial function $F : \{0,1\}^* \to \{0,1\}^*$ as follows:

$$F(\underbrace{11\ldots1}_{n}) = \begin{cases} 0, & \text{if the } n\text{-th Turing machine stops on the input } \underbrace{11\ldots1}_{n} \\ & \text{in at most } t(n) \text{ steps and prints 1} \\ 1, & \text{otherwise.} \end{cases}$$

Theorem 1.1.
(a) Function $F(x)$ is computable.
(b) Function $F(x)$ cannot be computed in time $t(n)$.

Definition 1.1. A function $t : \mathbb{N} \to \mathbb{N}$ is *time-constructible* is there exists a Turing machine that on every input $\underbrace{11\ldots1}_{n}$ returns the binary expansion of $t(n)$ and stops in time $O(t(n))$.

Now we can prove a stronger version of Theorem 1.1.

Theorem 1.2. (a) Let $t(n) > n$. Then $F(x)$ is computable in time poly(n) (*in the class we discussed that that for Turing machines with one tape the time bound* $O(t(n)^4)$ *is enough; with more accurate considerations the constant* 4 *could be reduced*).

(b) Let us assume that function $t(n)$ is in the definition of $F(x)$ is time-constructible. Then $F(x)$ cannot be computed in time $t(n)$. Moreover, there is no Turing machine that computes $F(x)$ for all x with a finite number of exceptions ("errors") and stops in time $\leq t(n)$ for all x from the domain of $F(x)$.

Definition 1.2. Let $f : \mathbb{N} \to \mathbb{N}$ be a total function. A language $L \subset \{0,1\}^*$ belongs to the class DTIME(f) is there is a Turing machine that decides L in time $O(f(n))$. In other words, there exists a Turing machine that

- returns 1 for inputs $x \in L$,
- returns 0 for inputs $x \notin L$,
- stops in time $O(f(n))$ on all inputs of length n.

Theorem 1.3 (time hierarchy)**.** (a) If $f(n) \leq g(n)$ for all n than

$$\text{DTIME}(f(n)) \subset \text{DTIME}(g(n)).$$

(b) Let $t : \mathbb{N} \to \mathbb{N}$ be total time-constructible function and $t(n) \geq n$ for all n. Then

$$\text{DTIME}(t(n)) \neq \text{DTIME}(t^4(n))$$

The constant 4 in this theorem can be improved. For tighter versions of the time hierarchy theorem see [1, 2, 3].

Definition 1.3. Let $f : \mathbb{N} \to \mathbb{N}$ be a total function. A language $L \subset \{0,1\}^*$ belongs to the class DSPACE(f) is there is a Turing machine that decides L on space $O(f(n))$. In other words, there exists a Turing machine that

- returns 1 for inputs $x \in L$,
- returns 0 for inputs $x \notin L$,

- uses $O(f(n))$ on the tape for all inputs of length n.

Definition 1.4. A function $s : \mathbb{N} \to \mathbb{N}$ is space-constructible is there exists a Turing machine that on every input $\underbrace{11\ldots1}_{n}$ returns the binary expansion of $t(n)$ and stops using space (memory) $O(s(n))$.

Theorem 1.4 (space hierarchy)**.** (a) If $f(n) \leq g(n)$ for all n than

$$\mathrm{DSPACE}(f(n)) \subset \mathrm{DSPACE}(g(n)).$$

(b) Let $s : \mathbb{N} \to \mathbb{N}$ be total space-constructible function and $s(n) \geq n$ for all n. Then

$$\mathrm{DSPACE}(s(n)) \neq \mathrm{DSPACE}(s^4(n))$$

Again, the constant 4 in this theorem can be improved significantly. For a *much* tighter version of the space hierarchy theorem see [1, 2, 3].

Definition 1.5. We use the following standard notation:

- $\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{DTIME}(n^k)$,
- $\mathrm{E} = \bigcup_{c=1}^{\infty} \mathrm{DTIME}(2^{cn})$,
- $\mathrm{EXP} = \bigcup_{k=1}^{\infty} \mathrm{DTIME}(2^{n^k})$,
- $\mathrm{PSPACE} = \bigcup_{k=1}^{\infty} \mathrm{DSPACE}(n^k)$.

It is not difficult to verify the we have the following relations between the complexity classes:

$$\mathrm{P} \subset \mathrm{NP} \subset \mathrm{PSPACE} \subset \mathrm{EXP}.$$

We known that it least one of these three inclusions must be strict.

Exercise 1.2. Are the following languages decidable or undecidable: the set of all $\underbrace{11\ldots1}_{n}$ such that the n-th Turing machine

(a) stopes on the input $\underbrace{11\ldots1}_{n}$;

(b) stopes on the input $\underbrace{11\ldots1}_{n}$ in time n^{10};

(c) stopes on all input $x \in \{0,1\}^*$ in time $|x|^{10}$?

Exercise 1.3. Explain why the following functions are time and space constructible:

- $f(n) = n$,
- $f(n) = n^2$,
- $f(n) = n^{10}$,
- $f(n) = 2^n$.

Exercise 1.4. Prove that

- $\mathrm{DTIME}(2^n) = \mathrm{DTIME}(2^{n+10})$,
- $\mathrm{DTIME}(2^n) \neq \mathrm{DTIME}(2^{10n})$.

Exercise 1.5. Assume that P = PSPACE. Prove that PSPACE $\neq$ EXP.

Lecture 2. April 13.

2.1. Nondeterministic computations. In the class we defined *nondeterministic Turing machines* (with one tape). We say that language L is recognized by a nondeterministic Turing machine M in time $t(n)$ if

- for every $x \in L$, given input x machine M can do nondeterministic choices in such a way that it stop in time $\leq t(|x|)$ and return the answer 1, i.e., there exists at least one possible branch of nondeterministic computation that terminates in at most $t(|x|)$ steps and gives the positive answer;
- for every $x \notin L$ machine never stops (for any branch of computation).

In a similar way we define languages that can be recognized by a nondeterministic Turing *in space* $s(n)$ for a given function $s(n)$.

Exercise 2.1. Let $t(n)$ be a time constructible function and $t(n) \geq n$ for all n. Assume that a nondeterministic Turing machine M accepts language L in time $t(n)$. Then there exists another nondeterministic Turing machine M' such that

- for every $x \in L$, given input x machine M' can stop in time $\leq O(t(|x|))$ and return the answer 1, i.e., there exists at least one possible branch of nondeterministic computation that terminates in at most $t(|x|)$ steps and gives the positive answer;
- for every $x \notin L$, given input x machine M' stops in time $\leq O(t(|x|))$ and returns the answer 0 for all nondeterministic choices.

Definition 2.1. Let $f : \mathbb{N} \to \mathbb{N}$ be a total function. A language L belongs to the class $\mathrm{NTIME}(f)$ if there exists a nondeterministic Turing machine M that recognizes L in time $O(t(n))$.

We define

$$\mathrm{NP} = \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k).$$

Definition 2.2. Let $f : \mathbb{N} \to \mathbb{N}$ be a total function. A language L belongs to the class NSPACE(f) if there exists a nondeterministic Turing machine M that recognizes L in space $O(s(n))$.

We define

$$\text{NPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(n^k).$$

Definition 2.3. We say that a language L has a *poly-time checkable system of certificates* if there exists a deterministic Turing machine M and two polynomials p_1 and p_2 such that

- M stops in time $p_1(n)$ on every input of length n,
- $x \in L$ if and only if there exists $y \in \{0,1\}^{p_2(|x|)}$ such that $M(x, y) = 1$.

Theorem 2.1. A language belongs to NP if and only if it has poly-time checkable system of certificates.

Exercise 2.2. Assume that P = NP. Prove that there exists an algorithm that finds a satisfying assignment for a satisfiable Boolean formula φ in polynomial time.

Definition 2.4 (reminder)**.** A language $L' \subset \{0,1\}^*$ is *polynomial-time Karp reducible* to a language $L' \subset \{0,1\}^*$ (or just *polynomial-time reducible*), denoted by $L \leq_P L'$, if there is a polynomial-time computable function

$$f : \{0,1\}^* \to \{0,1\}^*$$

such that for every $x \in \{0,1\}^*$ we have $x \in L$ if and only if $f(x) \in L'$.

A language L is *hard* for a class $\mathcal{C}$ (e.g., for the class NP or for the class PSPACE) if every $L' \in \mathcal{C}$ is polynomial-time Karp reducible to L. A language L is *complete* for a class $\mathcal{C}$ if L is hard for $\mathcal{C}$ and $L \in \mathcal{C}$.

Exercise 2.3. Prove that every PSPACE-hard language is NP-hard.

Exercise 2.4. Assume that every NP-hard language is PSPACE-hard. Prove that NP = PSPACE.

2.2. Languages of intermediate complexity.

Theorem 2.2 (Ladner)**.** If P $\neq$ NP then there exists a language $L \in \mathrm{NP} \setminus \mathrm{P}$ that is not NP-complete.

We did not prove Ladner's theorem in the class. Two proofs of this theorem can be found in [4]. In the class we discussed a few examples of problems that might have intermediate complexity (they are known to be in NP but not known to be in P or be NP-complete).

2.3. Nondeterministic and deterministic space complexity.

Theorem 2.3 (a simple version of Savitch's theorem)**.** Let $s(n) \geq n$ be a space-constructible function. Then $\mathrm{NSPACE}(f(n)) \subset \mathrm{DSPACE}(f^2(n))$.

From Savitch's theorem it follows that NPSPACE = PSPACE. We proved this theorem in the class. The proof can be found in [1, 2, 3].

Exercise 2.5. We denote

$$\mathrm{EXPSPACE} = \bigcup_{k=1}^{\infty} \mathrm{DSPACE}(2^{n^k})$$

and

$$\mathrm{NEXPSPACE} = \bigcup_{k=1}^{\infty} \mathrm{NSPACE}(2^{n^k}).$$

Prove that NEXPSPACE = EXPSPACE.

2.4. A complete problem for PSPACE**.**

Theorem 2.4. The problem QBF (Quantified Boolean Formula) is PSPACE-complete.

We proved this theorem in the class. The proof can be found in [1, 2, 3].

Lecture 3. April 120.

3.1. Randomized algorithms. In the class we defined *randomized* (or *probabilistic*) Turing machines — machines that can, if necessary, generate random bits during the computation (every time such a machine makes a random choice, it takes one of the two alternative ways to continue the computation, and each of the two alternatives is chosen with probability 1/2). For every branch of a randomized computation, its probability is the product of probabilities for each elementary probabilistic choice along this branch (this means that a branch of computation with k random choices has probability $1/2^k$). See [1, 3] for possible definitions of a probabilistic Turing machine. For a probabilistic Turing machine we define in the usual way *time* and *space* used by computation. Observe that

a probabilistic Turing machine (similar to a non-determinstic Turing machine) can run in different time and on different space on one and the same input, as the computation depends on the random choices.

We define two classes of computational complexity associated with polynomial time probabilistic algorithms.

Definition 3.1 (bounded-error probabilistic polynomial time, BPP)**.** We say that $L \in$ BPP if there is a probabilistic Turing machine M such that

- if $x \in L$ then $\text{Prob}[M(x) = 1] > 2/3$
- if $x \notin L$ then $\text{Prob}[M(x) = 1] < 1/3$
- there is a polynomial $p(n)$ such that for every $x \in \{0,1\}^*$ the computation $M(x)$ always (for all random choices during the computation) stops in at most $p(|x|)$ steps

The complexity class BPP will not change if we use instead of the thresholds $2/3$ and $1/3$ the numbers 0.99 and 0.01 respectively, or any other numbers $1-\epsilon$ and ϵ such that $0 < \epsilon < 1/2$.

Definition 3.2 (randomized polynomial time, RP)**.** We say that $L \in$ RP if there is a probabilistic Turing machine M such that

- if $x \in L$ then $\text{Prob}[M(x) = 1] = 1$
- if $x \notin L$ then $\text{Prob}[M(x) = 1] < 1/2$
- there is a polynomial $p(n)$ such that for every $x \in \{0,1\}^*$ the computation $M(x)$ always (for all random choices during the computation) stops in at most $p(|x|)$ steps.

The complexity class RPP will not change if we use instead the thresholds $1/2$ any other numbers $0 < \epsilon < 1$. In the class we proved the following properties of the complexity classes involving randomness:

Proposition 3.1. (a) P $\subset$ RP $\subset$ BPP $\subset$ PSPACE. (b) RP $\subset$ NP.

Definition 3.3 (zero-error probabilistic polynomial time, ZPP)**.** We say that $L \in$ ZPP if there is a probabilistic Turing machine M such that

- if $x \in L$ then $\text{Prob}[M(x) = 1] = 1$,
- if $x \notin L$ then $\text{Prob}[M(x) = 0] = 1$,
- there is a polynomial $p(n)$ such that for every $x \in \{0,1\}^*$ the *expectation* of the number of steps in the computation $M(x)$ is at most $p(|x|)$.

Exercise 3.1. (a) Assume that *expectation* of the number of steps in the computation $M(x)$ is T. Prove that

$$\text{Prob}[M(x) \text{ stops in } > 10T \text{ steps}] < 1/10.$$

(b) Prove that ZPP $\subset$ RP $\cap$ coRP (by definition, $L \in$ coRP if the complement of the language L belongs to RP).

3.2. Computation with oracle. In the class we defined an oracle Turing machine. We say that $A \leq_T^p B$ (A is s Turing reducible to B) if there is a poly-time oracle Turing machine that recognizes A given access to oracle B.

Simple properties of Turing reduction:

- $A \leq_T^p A$
- $A \leq_T^p \bar{A}$ (where $\bar{A}$ denotes the complement of A)
- if $A \leq_T^p B$ and $B \leq_T^p C$ then $A \leq_T^p C$
- if $A \leq_T^p B$ and $B \in \mathrm{P}$ then $A \in \mathrm{P}$
- if $A \in \mathrm{P}$ then for all B we have $A \leq_T^p B$

For every oracle A we can define the classes of computational complexity for Turing machines having access to oracle A: $\mathrm{DTIME}^A(f)$, $\mathrm{DSPACE}^A(f)$, $\mathrm{NTIME}^A(f)$, $\mathrm{NSPACE}^A(f)$ as well as P^A, NP^A, PSPACE^A, EXP^A.

Many statements of computational complexity *relativize*, i.e., they remain valid for computations with any oracle:

- $\mathrm{DTIME}^A(f) \subsetneqq \mathrm{DTIME}^A(f^4)$ for every time-constructible function f such that $f(n) \geq n$
- $\mathrm{DSPACE}^A(f) \subsetneqq \mathrm{DSPACE}^A(f^4)$ for every space-constructible function f such that $f(n) \geq n$
- $\mathrm{P}^A \subset \mathrm{NP}^A \subset \mathrm{PSPACE}^A \subset \mathrm{EXP}^A$
- $\mathrm{NPSPACE}^A = \mathrm{PSPACE}^A$

However, some important properties do not relativize, as the following two theorems show.

Theorem 3.1. There exists an oracle A such that $\mathrm{P}^A = \mathrm{NP}^A$.

Theorem 3.2. There exists an oracle B such that $\mathrm{P}^B \neq \mathrm{NP}^B$.

Sketch of proof of Theorem 3.1: Let $A = QBF$ (or any other PSPACE-complete language). Then both P^A and PSPACE^A are equal to PSPACE. Since NP^A is always sandwiched between P^A and PSPACE^A, we obtain $\mathrm{NP}^A = \mathrm{PSPACE}$. Therefore, $\mathrm{P}^A = \mathrm{NP}^A$.

Sketch of proof of Theorem 3.2: First of all, we define B. For each $n_k = 10^k$ we choose strings $x_k \in \{0,1\}^{n_k}$ and $y_k \in \{0,1\}^{2n_k}$ such that $C(x_k) \geq |x_k|$ and $C(y_k) \geq |y_k|$ (incompressible strings in the sense of Kolmogorov complexity). We let $B = \{x_k y_k \mid k = 1, 2, \ldots\}$ (here $x_k y_k$ is the concatenation of x_k and y_k). Thus, B contains one string of length $3 \cdot 10^k$ for each number $k > 0$ and no strings of any other length. Further, we let $L = \{x_k \mid k = 1, 2, \ldots\}$.

Lemma 3.1. $L \in \mathrm{NP}^B$.

Proof: Indeed, $x \in L$ if and only if there exists a y of length $2|x|$ such that $xy \in B$. To check that x belongs to L we need to guess a certificate y and make one query to oracle B. □

Lemma 3.2. (a) $|x_1 \dots x_k| \leq 10 + 100 + \dots + 10^k < (2/10)|x_{k+1}|$.
(b) $|y_1 \dots y_k| \leq 2 \cdot 10 + 2 \cdot 100 + \dots + 2 \cdot 10^k < (4/10)|x_{k+1}|$.

Proof: This follows from the definition of x_i and y_i. □

Lemma 3.3. For every polynomial time oracle deterministic Turing machine M and for all large enough k, for every input x of length n_k, the computation $M^B(x)$ cannot make queries z to the oracle for any z of length $3n_j$ for $j > k$.

Proof: The queries of length n_j for $j > k$ are too long: a polynomial time machine has no time to print them on the oracle tape.

Lemma 3.4. For every polynomial time oracle deterministic Turing machine M and for all large enough k, for every input x of length n_k, the computation $M^B(x)$ cannot make a query to the oracle $z = x_k y_k$.

Proof: For the sake of contradiction we assume that the computation $M^B(x)$ queries from the oracle $z = x_k y_k$.

From the previous lemma we know that M cannot make queries $z \stackrel{?}{\in} B$ with $z = x_j y_j$ for any $j > k$. Thus, all queries of $M^B(x)$ to the oracle can be of the following types:

- $z = x_j y_j$ for $j < k$ (these queries have positive answers),
- $z = x_k y_k$ (this query has a positive answer, if it ever happen),
- all other queries (these queries have negative answers).

To simulate the computation of $M^B(x)$ without oracle B, we need to know the input string x, the strings $x_j y_j$ for any $j < k$, and the moment T when the machine queries $z = x_k y_k$ for the first time. Due to Lemma 3.2, we can describe all this information with less than

$$|x| + |x_1 \dots x_k| + |y_1 \dots y_k| + \log T < (1 + 2/10 + 6/10)n_k + O(\log n_k)$$

($O(\log T)$ is logarithmic since the time of computation is polynomial in the length of the input).

It follows that we can compute y_k given much less than $2n_k$ bits of information. This contradicts the assumption that y_k is incompressible. □

Lemma 3.5. A polynomial time oracle deterministic Turing machine M having access to oracle B cannot compute the characteristic function of L.

Proof: Assume for the sake of contradiction that such a machine exists. We will show that for all large enough k, strings x_k are compressible.

We know from Lemma 3.3 and Lemma 3.4 that given an input x of length n_k, the computation $M^B(x)$ can make the following queries to the oracle:

- $z = x_j y_j$ for any $j < k$ (that must have positive answers),
- and other queries (that must have negative answers).

We can simulate this computation without oracle B given only the strings x_i and y_i for $i < k$. Therefore, we need only

$$|x_1 \dots x_k| + |y_1 \dots y_k| < (2/10 + 6/10)n_k$$

bits of information to reproduce the computations $M^B(x)$ for *every* x of length n_k. This is enough to find the unique string of length n_k for which $M^B(x)$ returns the positive answer. Hence, Kolmogorov complexity of x_k is less than n_k, and we get a contradiction. □

The statement of the theorem follows immediately from Lemma 3.5. □

Exercise 3.2 (optional)**.** Construct an oracle A such that $\mathrm{P}^A \neq \mathrm{BPP}^A$.

Exercise 3.3. Prove that $\mathrm{BPP}^A = \mathrm{EXP}^A \Rightarrow \mathrm{P}^A \neq \mathrm{PSPACE}^A$.

3.3. Interactive proofs and zero knowledge proofs.

Definition 3.4. We say that a language L belongs to the class IP (L has *an interactive proof system*), if there exists a poly-time randomized Turing machine V (*Verifier*) and an arbitrary Turing machine P (*Prover*) communicating with each other such that

- for each $x \in L$

  $$\text{Prob[result of communication of } V \text{ and } P \text{ on input } x \text{ is 1}] > 2/3,$$

  and
- for each $x \notin L$, for every prover P'

  $$\text{Prob[result of communication of } V \text{ and } P' \text{ on input } x \text{ is 1}] < 1/3.$$

Remark: The constants 2/3 and 1/3 in the definition above can be changed to 0.99 and 0.01 respectively, or even to any reals $1 - \varepsilon$ and ε (for $\varepsilon < 1/2$). These modifications will not affect the defined class IP (all variants of the definition are equivalent to each other).

Some simple properties:

- BPP $\subset$ IP (the class BPP corresponds to the «interactive protocols» where a poly-time randomized Verifier does not ask any question to the Prover and performs all the computations without assistance).
- NP $\subset$ IP (the class NP corresponds to the «interactive protocols» where a poly-time Verifier does not use randomness).

Proposition 3.2. The language

$$nonIso := \{(G_1, G_2) \ : \ \text{graphs } G_1 \text{ and } G_2 \text{ are } \textbf{not} \text{ isomorphic}\}$$

belongs to IP.

(Proven in the class. A proof can be found in [1, 2, 3].)

Exercise 3.4. Prove that the language of quadratic non-residues

$$NQR = \{(k, p) \mid p \text{ is prime, and there is no } m \text{ such that } m^2 = k \ mod \ p\}$$

belongs to IP.

Theorem 3.3. IP = PSPACE.

(We did not prove this theorem in the class. If you are interested, you can read a proof in [5] or in [1, 2, 3].)

Remark: It is known that the equality IP = PSPACE is not «relativizable», i.e., there exists an oracle A such that $\text{IP}^A \neq \text{PSPACE}^A$.

Zero knowledge proof: In the class we discussed a protocol of a *zero-knowledge* interactive proof for the problem of *Graph isomorphism*

$$Iso := \{(G_1, G_2) \ : \ \text{graphs } G_1 \text{ and } G_2 \text{ are isomorphic}\}.$$

This language belongs to NP since there is a trivial protocol where Prover simply reveals an isomorphism between given graphs. This trivial protocol discloses an isomorphism between the graphs, so Verifier can later use this information to convince other parties that the graphs are indeed isomorphic. So the trivial protocol is *not zero-knowledge*.

We discussed a subtler protocol (randomized and substantially interactive) where Prover convinces Verifier that two given graphs are isomorphic, but Verifier get no other information. In particular, Verifiers learns nothing about any specific isomorphism between these graphs. Even being convinced that G_1 and G_2 are isomorphic, in the future Verifier cannot reproduce this «proof» and convince any third party (without new intervention of Prover).

References

[1] Sylvain Perifel. *Complexité algorithmique.* Ellipses, 2014.

[2] Michael Sipser. *Introduction to the Theory of Computation.* Cengage Learning. 3rd ed. 2012.

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

[4] Lance Fortnow. Two Proofs of Ladner's Theorem. http://oldblog.computationalcomplexity.org/media/ladner.pdf

[5] Alexander Shen, *IP= SPACE: simplified proof.* Journal of the ACM (JACM) 39, no. 4 (1992): 878-880.

Further reading

[6] Lance Fortnow and Steve Homer. A Short History of Computational Complexity. Bulletin of the EATCS, 80, 2003, pp. 95-133.

`http://people.cs.uchicago.edu/~fortnow/beatcs/column80.pdf`

[7] Lance Fortnow. The status of the P versus NP problem. Communications of the ACM, 52(9), 2009, pp. 78-86.

`http://people.cs.uchicago.edu/~fortnow/papers/pnp-cacm.pdf`

[8] Scott Aaronson's Shtetl Optimized blog: Reasons to believe. (10 justifications for the belief that P $\neq$ NP)

`https://www.scottaaronson.com/blog/?p=122`

[9] Russell Impagliazzo. A Personal View of Average Case Complexity. Proceedings of the 10th Annual IEEE Conference Structure in Complexity Theory, 1995, pp. 134-147. (5 possible worlds of complexity)

`http://www.cs.mun.ca/~kol/courses/6743-w15/papers/russell-fiveworlds.pdf`