KØBENHAVNS UNIVERSITET

Computability and Complexity: Problem Set 1

Due: ~~Thursday February 19~~ Tuesday February 24 at 23:59 AoE.
Submission: Please submit your solutions via *Absalon* as a PDF file. State your name and e-mail address at the top of the first page. Solutions should be written in LaTeX or some other math-aware typesetting system with reasonable margins on all sides (at least 2.5 cm). Please try to be precise and to the point in your solutions and refrain from vague statements. Never just state an an answer, but make sure to also explain your reasoning. *Write so that a fellow student of yours can read, understand, and verify your solutions.* In addition to what is said below, the general rules for problem sets stated on the course webpage always apply.
Collaboration: Discussions of ideas in groups of two to three people are allowed—and indeed, encouraged—but you should always write up your solutions completely on your own, from start to finish, and you should understand all aspects of them fully. It is not allowed to compose draft solutions together and then continue editing individually, or to share any text, formulas, or pseudocode. Also, no such material may be downloaded from or generated via the internet to be used in draft or final solutions. Submitted solutions will be checked for plagiarism. You should also clearly acknowledge any collaboration. State close to the top of the first page of your problem set solutions if you have been collaborating with someone and if so with whom. *Note that collaboration is on a per problem set basis, so you should not discuss different problems on the same problem set with different people.*
Reference material: Some of the problems are "classic" and hence it might be possible to find solutions on the internet, in textbooks or in research papers. It is not allowed to use such material in any way unless explicitly stated otherwise. Anything said during the lectures or in the lecture notes, or any material found in Arora–Barak, should be fair game, though, unless you are specifically asked to show something that we claimed without proof in class. All definitions should be as given in class or in Arora–Barak and cannot be substituted by versions from other sources. It is hard to pin down 100% watertight, formal rules on what all of this means—when in doubt, ask the main instructor.
Grading: A total score of 70 points will be enough for grade 02, 105 points for grade 4, 140 points for grade 7, 175 points for grade 10, and 210 points for grade 12 on this problem set. Please note that problems are not necessarily presented in order of difficulty. Unless otherwise stated, every subproblem can be solved independently of other subproblems. Any revised versions of the problem set with clarifications and/or corrections will be posted on the course webpage `jakobnordstrom.se/teaching/CoCo26/`.
Questions: Please do not hesitate to ask the instructors or TA if any problem statement is unclear, but please make sure to send private messages when using Absalon—sometimes specific enough questions could give away the solution to your fellow students, and we want all of you to benefit from working on, and learning from, the problems. Good luck!

1 (10 p) The Arora–Barak textbook defines NP to be the set of languages L with the following property: There is a polynomial-time (deterministic) Turing machine M and a polynomial p such that $x \in L$ holds if and only if there is a witness y of length *exactly* $p(|x|)$ for which $M(x, y) = 1$. An attentive listener will have noticed that in the lectures we were in fact a bit more relaxed, and stipulated that the witness y should be of length *at most* $p(|x|)$, but might be shorter for some x.

Show that these two different definitions of NP are equivalent. That is, prove formally that the two definitions yield exactly the same set of languages in NP. (This is not hard, but please be careful so that you do not run into problems with any annoying details.)

2 (20 p) Let OneNegSat be the language of satisfiable CNF formulas in which each clause has at most one negated literal. Prove that OneNegSat is in P.

3 (20 p) Assuming that P = NP, describe a polynomial-time algorithm that gets a graph G as input and outputs a proper 3-colouring of G if the graph is 3-colourable and otherwise outputs the message "the graph is not 3-colourable."

(Note that the algorithm should solve the search problem of actually producing a 3-colouring and not just the decision problem of determining whether such a colouring exists. You are free to use any fact that follows immediately from the assumption P = NP, but please explain clearly when and how you use such facts.)

4 (40 p) When we established the Cook-Levin theorem in class, one of the lemmas we proved (or, at least, hand-waved a proof for) was that any Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be represented as a CNF formula. Can you use the construction in our proof of this lemma to write down such CNF representations of the following functions? For a full score, can you make the representation more concise than what a naive application of the lemma would give? (If a more concise representation is possible, then just giving such a representation is sufficient for a full score, as long as you explain how you found it and why it is correct.)

4a The *even* function

$$EVEN(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } \sum_{i=1}^{4} x_i \equiv 0 \pmod 2, \\ 0 & \text{otherwise.} \end{cases}$$

4b The *majority* function

$$MAJ(x_1, x_2, x_3, x_4, x_5) = \begin{cases} 1 & \text{if } \sum_{i=1}^{5} x_i \geq 3, \\ 0 & \text{otherwise.} \end{cases}$$

4c The *not-all-equal* function

$$NEQ(x_1, x_2, x_3, x_4, x_5, x_6) = \begin{cases} 1 & \text{if there are } i, j \in [6] \text{ such that } x_i = 1 \text{ and } x_j = 0, \\ 0 & \text{otherwise.} \end{cases}$$

5 (60 p) The reason we were not done with the Cook-Levin theorem after having proven the lemma mentioned in Problem 4 is that this lemma said that for a Boolean function $f : \{0,1\}^n \to \{0,1\}$ the size of the CNF representation could be as large as $n \cdot 2^n$. This means that a reduction that would write a CNF representation of the function $f_x^L(y)$ indicating whether the witness y makes the Turing machine verifier accept x would not necessarily run in polynomial time.

Can you prove that such a pessimistic size estimate is in fact necessary by exhibiting an exponential lower bound for the CNF representation of some Boolean function? Concretely, if we generalize the functions in Problem 4 to n bits (where we can also say that n is always odd for the *majority* function to make sense), can you prove an exponential lower bound $\exp(\Omega(n))$ for any of these functions? What is the best lower bound you can get?

(Note that we are looking for a formal proof here of any lower bounds for a full score, but partial results or convincing intuitive arguments can give partial credits.)

6 (50 p) Given a (multi)set $A = \{a_1, a_2, \ldots, a_m\}$ of integer terms and a target sum T, does there exist a subset $S \subseteq [m]$ such that $\sum_{i \in S} a_i = T$? We have learned in class that this problem, known as SubsetSum, is NP-complete. In this problem, we want to look more closely at the reduction establishing NP-hardness and study what happens when we tinker with this reduction.

6a Recall the reduction we saw from 3-Sat to SubsetSum constructed as follows: We are given a 3-CNF formula F with m clauses $C_1, \ldots, C_m$ over n variables $x_1, \ldots, x_n$. We build from this F a SubsetSum instance with $2(n+m)$ integer terms and target sum as follows, where all numbers below have $n+m$ decimal digits each:

- For each variable x_i, construct numbers A_i^T and A_i^F such that:
  - the ith digit of A_i^T and A_i^F is equal to 1;
  - for $n+1 \leq j \leq n+m$, the jth digit of A_i^T is equal to 1 if the clause C_{j-n} contains the literal x_i;
  - for $n+1 \leq j \leq n+m$, the jth digit of A_i^F is equal to 1 if C_{j-n} contains $\overline{x}_i$, and
  - all other digits of A_i^T and A_i^F are 0.
- For each clause C_j , construct numbers B_j^1 and B_j^2 such that
  - the $(n+j)$th digit of B_j^1 is equal to 1;
  - the $(n+j)$th digit of B_j^2 is equal to 2; and
  - all other digits of B_j^1 and B_j^2 are 0.
- The target sum T has
  - jth digit equal to 1 for $1 \leq j \leq n$ and
  - jth digit equal to 4 for $n+1 \leq j \leq n+m$.

Since we discussed this only briefly in class, write down a detailed proof establishing that the above is a correct reduction from 3-Sat to SubsetSum that proves the NP-hardness of the latter problem. That is, argue that the reduction (i) is polynomial-time computable, (ii) maps yes-instances to yes-instances, and (iii) maps no-instances to no-instances.

6b Given a 3-CNF formula F with m clauses over n variables, run the same reduction as in problem 6a except that the numbers B_j^1 and B_j^2 are omitted and the target sum T has all digits equal to 1. Formulas that map into satisfiable instances of SubsetSum under this modified reduction have a very specific form of satisfying assignments. Describe what such assignments look like.

6c Consider the language HackedSat consisting of 3-CNF formulas that map to satisifiable SubsetSum instances under the reduction in problem 6b. What is the complexity of deciding this language? Is it in NP? In P? Or NP-complete? For full credit, provide either a polynomial-time algorithm or a reduction from some problem proven NP-complete in chapter 2 in Arora-Barak or during the lectures.

7 (50 p) Recall that as discussed in class, we can agree on some fixed, standardized encoding of Turing machines in the binary alphabet $\{0, 1\}$. This allows us to view each Turing machine as an integer, namely the number whose binary expansion is the encoding of the Turing machine in question. We can also also agree that integers that do not correspond to Turing machines under this translation are interpreted as the Turing machine that immediate halts regardless of input. Given this convention, any number x encodes a Turing machine M_x, and we can define a function $g : \mathbb{N} \to \mathbb{N}$ by

$$g(x) = \begin{cases} s & \text{if } M_x \text{ takes } s < \infty \text{ steps before halting given the empty string as input;} \\ 0 & \text{if } M_x \text{ does not halt given the empty string as input.} \end{cases}$$

Note that given that we have fixed the encoding of Turing machines into binary strings, this is certainly a well-defined mathematical function that maps any non-negative integer x into some non-negative integer $y = g(x)$.

Even though the function $g(x)$ exists, *computing* it is another matter. In this problem, we want to show that $g(x)$ is not computable in a very strong sense. Namely, your task is to prove that $g(x)$ grows faster than any computable function. That is, show that there cannot exist any monotonically increasing function $h : \mathbb{N} \to \mathbb{N}$ and any Turing machine M_h such that $g(x) = \mathrm{O}(h(x))$ and M_h computes $h(x)$ when given x as input.

8 (60 p) We claimed in class that if M is a Turing machine that decides a language L in time $T(n)$, then there is an *oblivious* Turing machine M_{obl} that also decides L and runs in time $\mathrm{O}\big(T(n)^2\big)$. Your task is to prove (a slightly restricted version of) this claim.

Here are some useful facts that can be used freely:

1. We can assume that M has two tapes, just as usual, but that our oblivious Turing machine M_{obl} to be constructed has four tapes, where we think of the third tape as an "input copy tape" and the fourth tape as a "timer tape". For the purposes of this problem, let us refer to such a Turing machine as an "extended Turing machine".

2. We can assume that the upper bound on the running time $T(n)$ is a *time-constructible function*, meaning that given an input string x of length $n = |x|$ it is possible to compute the number $T(n)$ and write it in binary to the work tape in time $\mathrm{O}(T(n))$.

3. Adding or substracting some fixed constant K from a number written on the work tape can also be done efficiently.

4. The Turing machine can detect when it reaches the first position on the work tape and cannot move further right. We can also think of the work tape as being pre-filled with a special blank symbol so that the Turning machine can detect when it has reached a tape cell where the tape head has never been before.

5. If the alphabet of the Turing machine M is Σ, then we can let the alphabet of M_{obl} be $\Sigma^* = \{\sigma, \sigma^* \mid \sigma \in \Sigma\}$, where we can think of σ^* as encoding some useful information such as "the symbol written in this tape cell is σ and the tape head of M would be here right now."

8a Given a Turing machine M as above that runs in time at most $T(n) = \Omega\big(n\big)$ for any input x of size $n = |x|$, where $T(n)$ is a time-constructible function, describe how to construct an extended Turing machine M' that accepts the same language as M but runs in time *exactly* $T'(n)$ for all inputs x of size $n = |x|$, where $T'(n) = \mathrm{O}(T(n))$ is some well-chosen function.

8b Explain how, by using the extended alphabet $\Sigma^* = \{\sigma, \sigma^* \mid \sigma \in \Sigma\}$ described above, one single transition (computation step) of M can be simulated by a constant number of passes back and forth over the input and work tapes on an extended Turing machine M'. That is, at end of the passes the extended Turing machine tape heads should be in the first position of the tapes, the work tape should have changed as after a single write during a transition of M, and special symbols σ^* on the input copy tape and work tape should indicate the new positions of the tape heads for M.

8c Given a Turing machine M as above that runs in time at most $T(n) = \Omega\big(n\big)$ for any input x of size $n = |x|$ and for $T(n)$ being a time-constructible function, describe how to construct an extended Turing machine M_{obl} that decides the same language as does M; runs in time $\mathrm{O}\big(T(n)^2\big)$; and is oblivious.

9 (80 p) Your task in this problem is to produce a complete, self-contained proof of (the vanilla version of) Ladner's theorem that was discussed briefly during one of the lectures. The goal of this endeavour is (at least) twofold:

- To have you work out the proof in detail and make sure you understand it.
- To train your skills in mathematical writing.

When you write the proof, you can freely consult the lecture notes as well as the relevant material in Arora–Barak, but you need to fill in all missing details. Also, the resulting write-up should stand on its own without referring to the lecture notes, Arora–Barak, or any other source.

Your write-up should be accessible to a student who has studied and fully understood the material during the first two weeks of lectures of this course but does not necessarily know more than that. (However, you do not need to explain again the material in our first lectures, but can assume that these lectures have been fully digested.)

You are free to structure your proof as you like, except that all of the ingredients listed below should be explicitly addressed somewhere in your proof. (You can take care of them in whatever order you find appropriate, however. Please do not refer to the labelled subproblems in your write-up, since it should be a stand-alone text, but make sure that it is easy to find where in your solution the different items are dealt with.)

9a Define

$$\textsc{Sat}_P = \left\{ \psi 01^{n^{P(n)}} \;\middle|\; \psi \in \textsc{CnfSat} \text{ and } n = |\psi| \right\}$$

as the language of satisfiable CNF formulas padded by a suitable number of ones at the end as determined by the function P, which we assume to be polynomial-time computable.

9b Prove that if $P(n) = \mathrm{O}(1)$, then $\textsc{Sat}_P$ is NP-complete.

9c Prove that if $P(n) = \Omega(n/\log n)$, then $\textsc{Sat}_P \in \mathsf{P}$.

9d Give a complete description of the algorithm computing $H(n)$ (as in the lecture notes) and prove that H is well-defined in that the algorithm terminates and computes some specific function.

9e Prove that not only does the algorithm terminate, but it can be made to run in time polynomial in n. (Note that there are a number of issues needing clarification here, such as, for instance, how to solve instances of $\textsc{CnfSat}$ efficiently enough.)

9f Prove that $\textsc{Sat}_H \in \mathsf{P}$ if and only if $H(n) = \mathrm{O}(1)$.

9g Prove that if $\textsc{Sat}_H \notin \mathsf{P}$, then $H(n) \to \infty$ as $n \to \infty$.

9h Assuming that $\mathsf{P} \neq \mathsf{NP}$, prove that $\textsc{Sat}_H$ does not lie in P but also cannot be NP-complete.