CS Theory (Fall '25) Assigned: Nov 18, 2025

Homework 5

Instructors: *William Pires and Toniann Pitassi* Due: Nov 30, 2025 at 11am

Collaboration: Collaboration with other students in the class homework is allowed. However, you must write up solutions by yourself and understand everything that you hand in. You may also consult other reference materials, but you may not seek out answers from other sources or use AI tools. **You are required to list who you collaborated with on each problem, including any TAs or students you discussed the problems with in office hours. Also list any reference materials consulted other than the lectures and textbook for our class.** See the course webpage for more details.

Formatting: Write the solution to each part of each problem on a separate page. Be sure to correctly indicate which page each problem appears on in GradeScope. Please do not write your name on any page of the submission; we are using anonymous grading in GradeScope. If we can't find your solution to any problems because it was not properly tagged with the page, or if handwriting is not legible, you will receive 20 percent on these problems.

Grading: There are 60 total possible points. For all problems you have the option of answering "Don't know" on any parts of the question, and you will receive 20 percent of the total marks for those parts.

0 Exercises

Here are some recommended exercises. You should not turn in your solutions for these, and we will not grade them.

1. Sipser, 3rd edition, problem 7.9
2. Sipser, 3rd edition, problem 7.22
3. Sipser 3rd edition, problem 7.23 (solution in book)
4. Sipser, 3rd edition, problem 7.40 (solution in book)
5. Let Factor be the function that takes as input a natural number x in decimal notation, and outputs the prime factorization of x, also in decimal. For example, if the input is 13, then the output would be 13, and if the input is 12, the output would be 2,2,3. (You can output the prime factors in any order.) Prove that if P = NP then there is a polynomial-time algorithm for Factor. Note that NP is a class of *languages* (or decision problems), so factoring is not in NP. Therefore, you should give an algorithm for Factor, assuming that every language in NP is solvable in polynomial-time.
6. Try to prove that the following languages are NP-complete. (Sipser, Section 7.5 covers all of these in detail but try to do them yourself first for practice.) Clique, Independent Set, HamPath, Vertex-Cover, Subset-Sum.

Solution.

1. The following algorithm decides $TRIANGLE$:

Algorithm 1 A polytime algorithm for $TRIANGLES$

Input: $\langle G \rangle$, where G is a graph on n vertices

1: **for** Each triplet of verticles v_1, v_2, v_3 **do**
2: Check v_1, v_2, v_3 forms a triangle.
3: If yes accept.
4: **end for**
5: Reject.

The input size is $|\langle G \rangle| = O(n^2)$ and checking if a fixed (v_1, v_2, v_3) forms a triangle takes polynomial time in n. Since there's $\binom{n}{3} = O(n^3)$ triplets to check, the algorithm takes polynomial time. It's easy to see that the algorithm accepts if and only if G has a triangle.

2. First, $DOUBLE\text{-}SAT$ is in NP because there is a polynomial time verifier.

Algorithm 2 A polytime verifier for $DOUBLE\text{-}SAT$

Input: $\langle \langle \phi \rangle, c \rangle$, where ϕ is a CNF formula with n variables and m clauses.

1: Check $c = (\alpha_1, \alpha_2)$ where assignments to the variables of ϕ.
2: Check that α_1, α_2 are **different** assignments. If not reject.
3: Check that α_1 satisfies ϕ. If not, reject. ▷ I.e. for each clause C of ϕ, check α_1 set some literal in C to true.
4: Check that α_2 satisfies ϕ. If not, reject.
5: Accept.

We now prove this is a correct polytime verifier for Double-SAT.

We have that $|\langle \phi \rangle| = O(nm)$. First, we check c consists of 2 assignments to the variables (n bits string) and that they are different. This takes poly(n) time. One line (3) we check α_1 satisfies ϕ: For each of the m clauses of ϕ we check the assignment satisfies it, so this can be done in time poly(nm). Similarly line (4) takes time polynomial in n and m. Hence, the verifier runs in time poly$(|\phi|)$.

If $\langle \phi \rangle \in DOUBLE-SAT$ then there exists two assignments α_1^*, α_2^* which are different and both satisfy ϕ. When $c = (\alpha_1^*, \alpha_2^*)$ the verifier will accept.

If $\langle \phi \rangle \notin DOUBLE-SAT$ then for every choice of certificate c we will reject. Indeed, if c isn't two different assignments (α_1, α_2) to the variables we reject on line (1) or (2). Otherwise since $\langle \phi \rangle \notin DOUBLE-SAT$, there can only be at most one assignment that satisfies ϕ so at least one of α_1 or α_2 doesn't satisfy ϕ and the verifier rejects on line (3) or (4).

Now, we give a mapping reduction from 3-SAT to $DOUBLE\text{-}SAT$:

Algorithm 3 Mapping reduction f from SAT to $DOUBLE\text{-}SAT$

Input: ϕ, a Boolean formula containing variables $x_1, \ldots, x_k$.

1: Let x_{k+1} be a variable which is not in ϕ.
2: Let $\phi' := \phi \wedge (x_{k+1} \vee \overline{x_{k+1}})$.
3: Output ϕ'.

First, f can be computed in polynomial time since we just add the clause $(x_{k+1} \vee \overline{x_{k+1}})$ to ϕ. This can easily be done in time $O(|\phi|)$.

We show that $\phi \in$ 3-SAT if and only if $\phi' \in DOUBLE\text{-}SAT$.

In the first direction, suppose $\phi \in 3{-}SAT$. Then, there exists a satisfying assignment $\alpha \in \{0,1\}^n$ for $\{x_1, \ldots, x_k\}$. ϕ' will be satisfied using α and assigning x_{k+1} to *either* true or false [a]. This means ϕ' has two valid assignments ($\alpha \circ 0$ and $\alpha \circ 1$), so $\phi' \in DOUBLE\text{-}SAT$.

In the other direction, suppose $\phi' \in DOUBLE\text{-}SAT$. Take one of the satisfying assignments. That assignment, restricted to $\{x_1, \ldots, x_k\}$, must be a satisfying assignment for ϕ, so $\phi \in$ 3-SAT.

3. Solution in book.

4. Solution in book.

5. Suppose $P = NP$. First, we observe that the language $PRIMES$ of prime numbers satisfies $\overline{PRIMES} \in NP$ (the proof is the factors). If $P = NP$, then $\overline{PRIMES}$ is decidable in polynomial time (given x in decimal we can check x is prime in time poly(log(x))), so $PRIMES$ is too.[b]

   Let the language $PRIME\text{-}FACTOR\text{-}UNDER\text{-}\ell = \{\langle x, \ell\rangle \mid x \text{ has a prime factor} \leq \ell\}$. This language is in NP. Here's a simple NTM for it :

Algorithm 4 An NTM for $PRIME\text{-}FACTOR\text{-}UNDER\text{-}\ell$

Input: $\langle x, \ell\rangle$, where x, ℓ are numbers in decimal.

1: Non-deterministically pick a y number where $y \leq \ell$ and $1 < y < x$. ▷ We represent y in decimal
2: Check if y is prime (this can be done in polynomial time in the size of y as mentioned above). If not reject.
3: Check y divides x. If not reject.
4: Accept.

The NTM runs in polynomial in $|\langle x, \ell\rangle| = O(\log(x) + \log(\ell))$: Picking $y \leq x, \ell$ takes polynomial time. We can check y is prime in time poly(log(y)) = poly(log(x)) (since $y \leq x$.) And we can check y divides x in time poly($|\langle x\rangle|$).

It's easy to see that the NTM accepts if and only if the string y it picked is a prime factor of x and $y \leq \ell$. Hence, if $\langle x, \ell\rangle \notin PRIME\text{-}FACTOR\text{-}UNDER\text{-}\ell$, the NTM always rejects.

If $\langle x, \ell\rangle \in PRIME\text{-}FACTOR\text{-}UNDER\text{-}\ell$, then x has some prime factor z with $z \leq \ell$. When the NTM picks $y = z$ on line (1), the NTM will accept. So the NTM accepts in some path.

Therefore, if $P = NP$, $PRIME\text{-}FACTOR\text{-}UNDER\text{-}\ell$ will have a polynomial time algorithm.

Then, the following is an algorithm for Factor.

Algorithm 5 FactorAlgorithm

Input: $\langle x \rangle$, a number represented in decimal

1: Define FindPrimeFactor(l, h):
2: If $l = h$: return l.
3: If $PRIME\text{-}FACTOR\text{-}UNDER\text{-}\ell(\langle x, \lfloor \frac{l+h}{2} \rfloor \rangle)$: return FindPrimeFactor$(l, \lfloor \frac{l+h}{2} \rfloor)$.
4: Else: return FindPrimeFactor$(\lceil \frac{l+h}{2} \rceil, h)$.
5: $c \leftarrow$ FindPrimeFactor$(1, x)$.
6: If $c = 1$ or x, return "$1, x$".
7: Return "c, FactorAlgorithm(x/c)".

Basically, this algorithm implements binary search to find one factor, then recursively continues to the other factors.

We have a lot of things to show. First, we have to show that the algorithm runs in polynomial time. First, we show that FindPrimeFactor runs in polynomial time when given input (l, h), where $l, h \leq x$. We know that FindPrimeFactor can only recurse a maximum of $\log(h - l)$ times. In our case, FindPrimeFactor will recurse at most $\log x$ times, which is linear in the length of the representation of x. Each step in FindPrimeFactor takes polynomial time (here, we use our assumption that we can solve $PRIME\text{-}FACTOR\text{-}UNDER\text{-}\ell$ in poly time), so FindPrimeFactor will run in polynomial time.

Now, we show that FactorAlgorithm doesn't recurse too many times. Since each time we run it, the input is divided by at least 2, it can recurse at most $\log x$ times, which is polynomial. Since each time FactorAlgorithm runs, it takes only polynomial time, the algorithm will terminate in polynomial time.

Now, we have to show that the algorithm is actually correct. First, we show that FindPrimeFactor actually does find a prime factor of x between l and h. When FindPrimeFactor is first called with inputs $(1, x)$, there must be a prime factor between l and h. Relying on the correctness of the algorithm for $FACTOR\text{-}UNDER\text{-}\ell$, this condition holds for every execution of FindPrimeFactor. Therefore, when $l = h$, the prime factor must be l.

Now, we analyze the outer function. As a base case, if x is prime, it outputs a valid prime factorization. Assuming this inductively, for all other cases the return value of "c, FactorAlgorithm(x/c)" is a valid prime factorization because c is guaranteed to be a prime factor by the correctness of FindPrimeFactor.

6. See Sipser for all of these proofs.

[a] Such an assignment would always satisfy $(x_{k+1} \vee \overline{x_{k+1}})$, clearly would satisfy the clauses in ϕ

[b] $PRIMES$ is actually in P regardless of whether $P = NP$, but this is very nontrivial.

Please read the instructions at the end of this HW carefully before doing the problems. In particular, you may only use the 3 NP-complete problems defined there in your reductions.

1 Neural Networks (20 points)

A simple 3-layer feed-forward neural network is described as follows (see the accompanying figure below):

- The input layer consists of n input nodes, denoted $x_1, \ldots, x_n$;
- The middle layer consists of m middle (hidden) units denoted $y_1, \ldots, y_m$. Each hidden unit computes a weighted threshold function of the inputs. Specifically for each hidden unit y_j, there is a weight vector $\overline{w}_j = w_{j,1}, \ldots, w_{j,n}$, $w_{j,i} \in \{-1, 0, 1\}$ and a threshold value $t_j \in \mathbb{Z}$ and the activation of y_i is 1 if $\sum_{i=1}^{n} w_{j,i} x_i \geq$ and 0 otherwise.
- The output layer consists of one output node denoted by f. It takes as input the hidden layer activation values and has a weight vector $\overline{v} = v_1, \ldots, v_m$, $v_k \in \{-1, 0, 1\}$ and a threshold value $t \in \mathbb{Z}$ and outputs 1 if $\sum_{k=1}^{m} v_k y_k \geq t$ and 0 otherwise.

The input to N-NET consists of: (i) the first layer weight vectors $w_{j,i} \in \{-1, 0, 1\}$, $1 \leq j \leq m$, $1 \leq i \leq n$, and the associated thresholds $t_1, \ldots, t_m \in \mathbb{Z}$; (ii) the second layer weight vectors $v_1, \ldots, v_m \in \{-1, 0, 1\}$, and threshold $t \in \mathbb{Z}$. The threshold values $t_1, \ldots, t_m$ and t are all given in binary representation. The input is in the language if and only if there exists an assignment $\alpha \in \{-1, 1\}^n$ to the input variables $x_1, \ldots, x_n$ such that the output node evaluates to 1.

(a) Prove that N-NET is in NP. Note that in your proof of correctness, you will need to discuss the length of the input as a function of n, m and the threshold values.

Solution.

Consider the following algorithm, which we will show is a polynomial-time verifier for N-NET:

Algorithm 6 Verifier for N-NET

Input: $\langle N, \alpha \rangle$ where N is a 3 layer neural network and α is an input assignment for N

1: Check that N is a valid encoding as described in the problem. Determine the number n of input variables for N, and check that $\alpha \in \{-1, 1\}^n$ is a valid input into N. If any of these are false, reject.
2: For each $1 \leq j \leq m$, where m is the number of hidden units:
   2a. Calculate $z_j = \sum_{i=1}^{n} w_{j,i} \alpha_i$.
   2b. If $z_j \geq t_j$, let $y_j = 1$; otherwise let $y_j = 0$.
3: Calculate $z = \sum_{j=1}^{m} v_j y_j$.
4: If $z \geq t$, *accept*. Otherwise, *reject*.

First we show this algorithm is indeed a verifier:

- When $\langle N \rangle \in$ N-NET, there exists an input assignment α such that $N(\alpha) = 1$, therefore the

verifier will accept $\langle N, \alpha \rangle$ for this certificate α.

- When $\langle N \rangle \notin$ N-NET, N outputs 0 on any input assignment α, therefore the verifier rejects $\langle N, \alpha \rangle$ for any α.

Now we show that this verifier runs in polytime. First note that the weights $(w_{i,j}, v_j)$ and thresholds (t_j, t) have size $\leq |\langle N \rangle|$ because they are encoded in the input. Similarly, $m, n \leq |\langle N \rangle|$ because N is encoded by at least this many variables.

The algorithm performs arithmetic computations (add, multiply, compare) on integers either from the input or resulting from these computations. Each arithmetic operation takes polynomial time in terms of its operands, and thus all of these arithmetic operations take polytime in terms of the input size.

Step 2 performs $O(mn)$ operations, while steps 3 and 4 perform $O(m)$ operations. Therefore the total runtime is $O(mn)\text{poly}(|\langle N \rangle|) + O(m)\text{poly}(|\langle N \rangle|) = \text{poly}(|\langle N \rangle|)$.

Therefore N-NET $\in NP$.

(b) Prove that N-NET is NP-hard, and therefore by (a) and (b) it is NP-complete.

Solution.

We will prove NP-hardness through showing a polytime mapping reduction from 3-SAT. Fix some $N_0 \in$ N-NET, and define the following function:

Algorithm 7 Polytime mapping reduction 3-SAT $\leq_p$ N-NET

Input: $\langle \Phi \rangle$ where Φ is a boolean formula with at most 3 literals per clause

1: Remove the clauses in Φ that contain both a variable and its complement. If Φ is now empty, *output* $\langle N_0 \rangle$.
2: Let n be the number of variables in Φ. Let m be the number of clauses in Φ.
3: For each clause C_j, where $1 \leq j \leq m$, construct the hidden unit y_j:
   1. For each variable x_i, where $1 \leq i \leq n$:
      (a) If x_i is in C_j, set the corresponding incoming weight as $w_{j,i} = 1$.
      (b) If $\overline{x_i}$ is in c_j, set the weight as $w_{j,i} = -1$.
      (c) Otherwise, let $w_{j,i} = 0$.
   2. Set the threshold as $t_j = 2 - |C_j|$, where $|C_j|$ denotes the number of literals in C_j.
   3. Set the outgoing weight as $v_j = 1$.
4: Let the final threshold be $t = m$.
5: *Output* the encoding of the weights and thresholds, $\langle \{w_{i,j}, t_j, v_j \mid 1 \leq i \leq n,\ 1 \leq j \leq m\}, t \rangle$.

We first show that f can be computed in polytime. Let m be the number of clauses and n the number of variables in boolean formula Φ, so that $|\Phi| = \text{poly}(m, n)$. For each of the m clauses, we set its n corresponding (incoming) weights by scanning for the corresponding variable in that clause. This

takes $O(mn)$ steps. We also assign m threshold values, m (outgoing) weights, and a final threshold value, which takes $O(m+n)$ steps. Since in total this takes $O(mn)$ time, the function is computable in polynomial time in terms of the size of Φ.

Before proving f is a mapping reduction, we first make two useful observations:

1. Removing the clauses containing some $x_i \vee \overline{x_i}$ preserves (un)satisfiability of Φ, because any disjunction containing $x_i \vee \overline{x_i}$ is always satisfied. Therefore we will prove correctness with respect to Φ that do not contain such clauses.

2. In the network encoded by $f(\langle\Phi\rangle)$, the weight $w_{j,i}$ is nonzero exactly when x_i or $\overline{x_i}$ is in the jth clause C_j. Therefore if the network's input is $\alpha \in \{-1,1\}^n$, the sum $\sum_i w_{j,i}\alpha_i$ is over exactly $|C_j|$ nonzero terms, which are $+1$ when the literal is satisfied and -1 when the literal is not. That is, the sum equals the number of true literals minus the number of false literals in clause C_j (under the assignment α). Thus α is a satisfying assignment for clause C_j if and only if the hidden unit y_j is activated on input α.

Now we show that f is a mapping reduction.

- When $\langle\Phi\rangle \in$ 3-SAT, it has a satisfying argument $x \in \{0,1\}^n$ such that $\Phi(x) = 1$. Let α be the same as x except 0 becomes -1. If α is inputted into the network encoded by $f(\langle\Phi\rangle)$, each hidden unit y_j is 1 by the observation from earlier. Then the output node has sum $m \geq m$ and the network outputs 1. Thus $f(\langle\Phi\rangle) \in$ N-NET.

- When $f(\langle\Phi\rangle) \in$ N-NET, it has an input assignment $\alpha \in \{-1,1\}^n$ so that the network's output is 1. Then each of the m hidden units must have value $y_j = 1$ so that the output sum is at least m. Furthermore, each $y_j = 1$ implies, by the earlier observation, that α is a satisfying assignment for each clause C_j in Φ. Since Φ is a conjunction of these clauses, it is satisfied by α. Thus $\langle\Phi\rangle \in$ 3-SAT.

Since f is a computable function such that $\langle\Phi\rangle \in$ 3-SAT $\iff$ $f(\langle\Phi\rangle) \in$ N-NET, N-NET must be NP-hard like 3-SAT. We showed N-NET $\in NP$ in part (a), and thus we conclude N-NET is NP-complete.

2 Fun with Math (25 points)

In the game of ChangePro, the input is a target number t, and a multiset S of coins where each coin has an associated positive integer value (e.g., $S = \{2,3,4,4,5,7,7,10,25\}$). The goal is to write an expression using the source numbers in S and the operations + and $\times$ that evaluates to the target number, t. The expression doesn't have to use all source numbers, but can only use each number as many times as it appears.

(a) Give a reasonable representation of the input to ChangePro as a string x over some finite alphabet. Specify your alphabet and encoding of the input. What is the length of x in terms of t and the numbers in S ?

Solution.

One encoding for multiset $S = \{s_1, s_2, \ldots, s_n\}$ and target t over the input alphabet $\Sigma = \{0, 1, \#\}$ could be the string

$$x = \langle S, t \rangle = \langle s_1 \rangle_{bin} \# \langle s_2 \rangle_{bin} \# \cdots \# \langle s_n \rangle_{bin} \# \# \langle t \rangle_{bin}$$

where $\langle \cdot \rangle_{bin} : \mathbb{N} \to \{0, 1\}^*$ maps a positive integer value to its binary representation. In this case the length of x is

$$n + 1 + \sum_{i=1}^{n} \log_2 |s_i| + \log_2 |t|.$$

(b) Using your representation, prove that ChangePro is in NP.

Solution.

We can construct the following polytime verifier for ChangePro

Algorithm 8 Verifier for ChangePro

Input: $\langle \langle S, t \rangle, E \rangle$ where S is a multiset, t is a positive integer, and E is an arithmetic expression

1: Check that S is a multiset of positive integer values, t is a positive integer value, and E is an expression consisting only of the source numbers in S and operations $+$ and $\times$.
2: Evaluate the expression E. If it equals t, accept. Otherwise reject.

It is straightforward to check this is a verifier. If $\langle S, t \rangle \in$ ChangePro, then there exists a valid expression E that evaluates to t using only the values in S, addition, and multiplication, When this E is the certificate, $\langle \langle S, t \rangle, E \rangle$ is accepted by the verifier. If $\langle S, t \rangle \notin$ ChangePro, then no expression E that uses only values in S, addition, and multiplication will evaluate to t. Therefore $\langle \langle S, t \rangle, E \rangle$ is rejected for all E.
Furthermore, we can check that the verifier runs in polynomial time in terms of the size of $\langle S, t \rangle$. There are at most $|S|$ numbers in the expression E, and thus $O(|S|)$ operations in the expression. Each operation takes polynomial time in terms of its operands' sizes, which are either values in S or the results of previous addition/multiplication, and thus polynomial in terms of $|S|$. Since the final value of E has polynomial size in terms of $|S|$, comparing it to t is also a polytime operation. Thus the verifier runs in poly($|\langle S, t \rangle|$) time.

(c) Prove that ChangePro is NP-hard, and therefore by (b) and (c) it is NP-complete.

Solution.

We show the following is a polytime mapping reduction from SUBSET-SUM to ChangePro.

Algorithm 9 Mapping reduction f

Input: $\langle S, t\rangle$ where S is multiset of positive integers and t is positive integer

1: Construct the multiset S' by multiplying every number in $S = \{s_1, \ldots, s_n\}$ by $t+1$. That is, let $S' = \{(t+1)s_1, \ldots, (t+1)s_n\}$.
2: Output $\langle S', (t+1)t\rangle$.

Note that this function runs in polytime in terms of S and t because multiplying every value by $t+1$ takes polynomial time.
Now we verify correctness of the mapping reduction.

- If $\langle S, t\rangle \in$ SUBSET-SUM, there exists some sub multiset $R \subset S$ so that $\sum_{r\in R} r = t$. Then $(t+1)R \subset S'$, and $\sum_{r'\in(t+1)R} r' = \sum_{r\in R}(t+1)r = (t+1)t$. Thus $f(\langle S, t\rangle) = \langle S', (t+1)t\rangle \in$ ChangePro.
- If $\langle S, t\rangle \notin$ SUBSET-SUM, no subset sums to t, and therefore no subset of $S' = (t+1)S$ sums to $(t+1)t$. Furthermore, any two elements in S' are of form $(t+1)x, (t+1)y$ for $x, y \in S$. So $(t+1)x \cdot (t+1)y \geq (t+1)^2 > (t+1)t$. Note that we use the fact that x, y, t are positive integers by definition. Since no expression evaluates to the target with multiplication and addition, $f(\langle S, t\rangle) \notin$ ChangePro.

We conclude that ChangePro is NP-hard. Combined with part (b) which shows ChangePro $\in$ NP, this proves ChangePro is NP-complete

3 NP-Completeness (15 points)

Let L be the language consisting of all strings $w \in \{0,1\}^*$ such that w contains at least 3 consecutive 1's. Prove that L is NP-complete if and only if P = NP.

Solution.

First note that L is regular since it can be expressed as the regex $\{0,1\}^*111\{0,1\}^*$. Therefore L is in P: a polytime decider for L would be to run a DFA (which terminates in finite steps) to determine if a string is accepted.

($\implies$) Assume L is NP-complete. By definition of NP-completeness, any language L' in NP has a polytime (mapping) reduction f to L, that is $L' \leq_p L$. Let D be a polytime decider for L. We can build a polytime decider for L' as follows:

Algorithm 10 A polytime decider for L'

Input: x

1: Compute $f(x)$ ▷ This takes time poly($|x|$)
2: Run D on $f(x)$. ▷ This takes time poly($|f(x)|$) = poly($|x|$).
3: If D accepted, accepts.
4: If D rejected, reject.

The above takes polynomial time. Since $x \in L' \iff f(x) \in L$ we have that on line 2 D accepts iff $x \in L'$. Hence the above correctly decides L' and does so in polynomial time. So $L \in$ P. Hence NP $\subseteq$ P and we have P = NP.

($\Longleftarrow$) Assume P = NP. Therefore the NP-complete language CLIQUE is in P and thus has a polytime decider D. Under these assumptions, we can show that CLIQUE polytime reduces to L. Consider the following function f

Algorithm 11 Mapping reduction f for CLIQUE $\leq_p L$

Input: $\langle G, k\rangle$ where G is an undirected graph and k is a positive integer

1: (If the input is not of the proper format, output 0.)
2: Run D on $\langle G, k\rangle$.
3: If D accepted, *output* 111.
4: if D rejected, *output* 0.

First note that f can be computed in polynomial time in terms of $|\langle G, k\rangle|$ because D runs in polynomial time in terms of $|\langle G, k\rangle|$.

Furthermore, $\langle G, k\rangle \in$ CLIQUE iff $\langle G, k\rangle$ is accepted by D. So if $\langle G, k\rangle \in$ CLIQUE, $f(\langle G, k\rangle) = 111 \in L$. If $\langle G, k\rangle \notin$ CLIQUE, $f(\langle G, k\rangle) = 0 \notin L$.

We have shown CLIQUE $\leq_p L$, and therefore L is NP-hard because CLIQUE is. Since $L \in$ P $\subset$ NP, we conclude that L is NP-complete.

Instructions on level of detail required for algorithms, reductions and proofs You do not need to discuss the mechanics of an actual TM (tapes, states, the transition function). See chapter 7 of the book for examples. Make sure your algorithm is clear and well defined (e.g., what is the input to the program, what are the main subroutines/loops/steps; specify the data structure(s) being used (variables, arrays, etc). You can give a few sentences to describe the ideas behind your algorithm, **but we expect your algorithm to be properly written pseudocode (with lines of codes, clear indentation etc...). Do not give us a big paragraph of text as your algorithm.**

We highly recommend that you number the lines of your algorithm to be able to refer to them easily.

Some clarifications:

Unless instructed, you should think of every numbers as being encoded in binary.

For Problems 1 and 2, when proving $L \in$ NP. You can either give an NTM or a verifier.

a) You have to prove your NTM / Verifier is running in polynomial time (use asymptotic notation).

b) You have to prove that if $x \in L$ your NTM accepts in some computation path, or that there is a certificate that makes your Verifier accepts. You should explain what the computation path / certificate is.

c) You have to prove that if $x \notin L$ your NTM rejects all computation paths, or that there are no certificates that make your Verifier accept.

For Problems 1 and 2, when proving L is NP-hard.

a) Give a polytime mapping reduction f from A to L where A is NP-hard (see below for which problems A you are allowed to use). Give this reduction as an algorithm in pseudo-code.

b) Prove that f can be computed in polynomial time.

c) You have to prove that if $x \in A$, then $f(x) \in L$.

d) You have to prove that if $x \notin A$, then $f(x) \notin L$.

NP-Hard problems you're allowed to use on the HW. You are only allowed to use the following NP-hard problems. Do not use other problems, even if we mentioned them in class or discussed in the book.

- In the following, S is a multiset of k positive integers[1], given in binary.

  SUBSET-SUM $:= \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\},$ and for some $\{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\},$ we have $\sum y_i = t\}$

- CLIQUE $= \{\langle G, k\rangle \mid$ G is an undirected graph with a k-clique $\}$.

- 3-SAT $= \{\langle \Phi\rangle \mid$ Φ satisfiable Boolean formula with at most 3 literals per clause$\}$.

[1]We say multiset (and not set) because the same number can appear multiple times in S, for instance you can have $S = \{2, 4, 4, 7, 7\}$