1. Consider the **Subset Sum** problem: given a set of positive integers \( X = \{x_1, \ldots, x_n\} \) and a positive integer \( T \), we ask if there exists a subset \( X' \subseteq X \) such that
\[
\sum_{x \in X'} x = T.
\]
**Subset Sum** is NP-complete. Note that as stated this is a decision problem. The answer is yes or no rather than a list of numbers. Show that if P=NP then you can use the polynomial time solution to the decision problem to give a polynomial time solution to the problem of actually finding the subset. Present pseudo-code describing the algorithm and justify that it will have polynomial running time by giving an upper bound for its execution time under the assumption that the decision problem can be solved in \( O(p_d(n)) \) time.

2. The diagram shown in Figure 1 is a non-deterministic Turing machine that solves the subset sum problem in non-deterministic polynomial time. I will use this machine as a motivating example when presenting the proof of the Cook-Levin Theorem showing that the 3-CNF satisfaction problem is NP-complete. Accordingly, I thought it might be useful for you to explore this Turing machine a bit before we discuss the proof.

The input this machine expects is a value \( N \) encoded in binary and surrounded by a pair of dollar signs (\$) followed by a list of integer values also encoded in binary but separated from one another and terminated by number signs (#). Thus, the input \$1001$101#100#11#10# would represent the subset-sum question “Is there a sublist of the numbers (5, 4, 3, 2) that adds up to 9?”

In this figure, a notation of the form 0/1/# → 0/1/#, \( L \) on a transition is meant as a shorthand meaning that the transition can be taken on any of the inputs 0, 1 or # and that the symbols written if the transition is taken are 0, 1, and # respectively. That is, this particular example says that on any of the symbols listed, the machine can move one square to the left while leaving the previous tape cell unchanged.

This machine is fairly simple, but it does not employ the most obvious algorithm to solve the problem. Unfortunately, like all too many programmers before me, I failed to include good comments explaining the algorithm the Turing machine shown in the figure uses.

The “obvious” algorithm is to first randomly cross out some list of the numbers provided by replacing their digits with number signs (my non-obvious algorithm starts by performing exactly this process). Then, the obvious way to verify that the random guesses made generated a correct solution would be to repeatedly subtract one from the number between the dollar signs and from one of the numbers that were not crossed out initially to verify that this process leads to a situation in which all of the non-crossed out numbers get reduced to zeros at the same point that the number between the dollar signs reaches zero.
Figure 1: A NTM that decides Subset-sum
(a) Help me out! Generate the missing documentation. That is, please give a brief, informal description of the algorithm implemented by the Turing machine in Fig. 1 and justify its correctness.

(b) Analyze the running time of my Turing machine and give a bound on its worst-case running time as a function of the size of its input string.

(c) Explain why I used this Turing machine as my lecture example rather than the “obvious” one. It turns out that although the obvious one might have been easier to implement, it would not have been an appropriate example.

3. Complete problem 7.26 in Sipser:

Let \( \phi \) be a 3CNF-formula. An \( \neq \)-assignment to the variable of \( \phi \) is one where each clause contains two literals with unequal truth values. In other words, an \( \neq \)-assignment satisfies \( \phi \) without assigning three true literals in any clause.

(a) Show that the negation of any \( \neq \)-assignment to \( \phi \) is also an \( \neq \)-assignment.

(b) Let \( \neq SAT \) be the collection of 3CNF-formulas that have \( \neq \)-assignments. Show that we obtain a polynomial time reduction from \( 3SAT \) to \( \neq SAT \) by replacing each clause \( c_i \)

\[
(y_1 \lor y_2 \lor y_3)
\]

with the two clauses

\[
(y_1 \lor y_2 \lor z_i) \text{ and } (\overline{z} \lor y_3 \lor v)
\]

where \( z_i \) is a new variable for each clause \( c_i \) and \( v \) is a single additional new variable.

(c) Conclude that \( \neq SAT \) is NP-complete. (Hint: Given parts (a) and (b) this step should be trivial.)

Note: This problem depends on material we will not cover in class until Monday 11/27.