Exercise 11.5 Let $x_1, \ldots, x_n \in \mathbb{N}$. For each of the following problems, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.[2]

[2]You can use the solution of one subproblem to solve another, as long as there's no circular dependencies overall.

Exercise 11.5.1. The *partition problem* asks if one can partition $x_1, \ldots, x_n$ into two parts such that the sums of each part are equal.

Solution. We claim that a polynomial time solution for the partition problem would imply a polynomial time solution for SAT. To see this, we present a polynomial time reduction from subset sum, a problem known to be hard, to the partition problem.

Note (Notation)**.** Given some set $S = \{s_1, \ldots, s_n\}$, we denote the sum $\sum\limits_{s \in S} s$ with ΣS.

Consider an arbitrary instance of subset sum. That is, suppose we have a set of positive integers

$$A := \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{Z}_+$$

and a positive integer target value $T \in \mathbb{Z}_+$. Now, let $x := 2T - \Sigma A$, and define a new set $\overline{A} := A \cup \{x\}$. In the case that $T < \Sigma A/2$, notice that the value of x will end up being negative. However, if there exists some set $B = \{\beta_1, \ldots, \beta_k\} \subseteq A$ such that $\Sigma B = T$ then $A \setminus B$ sums to $\Sigma A - T \leq \Sigma A/2$, so we can simply rephrase the problem to use $(\Sigma A) - T$ as the target value.

Then, notice that

$$\begin{aligned} \Sigma\overline{A} &= \Sigma A + 2T - \Sigma A \\ &= 2T. \end{aligned}$$

Correctness. (SS $\Longrightarrow$ PP) Suppose there exists some $B \subseteq A$ such that $\Sigma B = T$. Consider the partition of $\overline{A}$ defined as $\overline{B} = B \cup \{x\}$. Then,

$$\Sigma\overline{B} = T + 2T - \Sigma A.$$

The remaining partition is then $C := \overline{A} \setminus \overline{B}$, and

$$\begin{aligned} \Sigma C &= 2T - T + 2T - \Sigma A \\ &= T + 2T - \Sigma A, \end{aligned}$$

and we can see that these two sums are equal. Hence, the partition problem is solved.

(SS $\Longleftarrow$ PP) Suppose the set $\overline{A}$ has a valid partition such that each of the two subsets sum to T. Recall that $\overline{A}$ is defined as the union of A and the singleton set $\{x\}$. By the pigeonhole principle, we know that one of these subsets of $\overline{A}$ is a subset of A, whence the subset sum problem is solved. ■

Since each step in the reduction process takes only $O(1)$ or $O(n)$ time, the entire reduction can be done in polynomial time relative to the size of A. Thus, a polynomial time solution for the partition problem implies a polynomial time solution for SAT. □

Exercise 11.5.2. The *3-partition problem* asks if one can partition $x_1, \ldots, x_n$ into 3 parts such that the sums of each part are all equal.

Solution. □

Exercise 11.5.3. The *any-k-partition problem* asks if one can partition $x_1, \ldots, x_n$ into k parts, for any integer k $\geq$ 2, such that the sums of each part are all equal.

Solution. □

Exercise 11.5.4. The *almost-partition problem* asks if one can partition $x_1, \ldots, x_n$ into two parts such that the two sums of each part differ by at most 1.

Solution. □

Exercise 11.5.5. [3]Let n be even. The *perfect partition problem* asks if one can partition $x_1, \ldots, x_n$ into two parts such that

(a) Each part has the same sum.

(b) Each part contains the same number of x_i's.

[3]IMO, this one is the trickiest.

Solution. □

Exercise 11.8 (Approximating subset sum.) Let $\epsilon \in (0,1)$ be fixed. Here we treat ϵ as a fixed constant (like $\epsilon = .1$, for 10% error); in particular, running times of the form $O(n^{O(1/\epsilon)})$ count as a polynomial.

A $(1 \pm \epsilon)$-approximation algorithm for subset sum is one that (correctly) either:

1. Returns a subset whose sum lies in the range $[(1-\epsilon)T, (1+\epsilon)T]$.
2. Declares that there is no subset that sums to (exactly) T.

Note that such an algorithm does not solve the (exact) subset sum problem.

Exercise 11.8.6. Suppose every input number x_i was "small", in the sense that $x_i \leq \epsilon T$. Give a polynomial time $(1 \pm \epsilon)$-approximation algorithm for this setting.

Solution. □

Exercise 11.8.7. Suppose every input number x_i was "big", in the sense that $x_i > \epsilon T$. Give a polynomial time $(1 \pm \epsilon)$-approximation algorithm for this setting.

Solution. □

Exercise 11.8.8. Now give a polynomial time $(1 \pm \epsilon)$-approximation algorithm for subset sum in the general setting (with both big and small inputs).

Solution. □

Exercise 12.10 Let $G = (V, E)$ be a directed graph with m edges and n vertices, where each vertex $v \in V$ is given an integer label $\ell(v) \in \mathbb{N}$. The goal is to find the length of the longest path[3] in G where the labels of the vertices are (strictly) increasing.

[3] Recall that a path is a walk that does not repeat vertices.

Exercise 12.10.9. Suppose G is a DAG. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

Solution. □

Exercise 12.10.10. Consider now the problem for general graphs. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

Solution. □

Exercise 12.10.11. Suppose instead we ask for the length of the longest path in G where G is a general graph and the labels of the vertices are weakly increasing.[4] For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

[4] A sequence $x_1, \ldots, x_k$ is weakly increasing if $x_1 \leq x_2 \leq \cdots \leq x_k$.

Solution. □