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.

Note. You may (and should) assume $T, x_i \in \mathbb{R}_{\geq 0}$. The problem appears (computationally) hard otherwise.

Exercise 11.8.1. 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. Assuming $x_i \leq \epsilon T$ for all i, we can take a greedy approach that exploits the "smallness" of each x_i to guarantee the construction of a subset-sum that does not overshoot the target range $[(1 - \epsilon)T, (1 + \epsilon)T]$.

approx-subset-sum($\{x_1, ..., x_n\}, \epsilon, T$):

/ returns a nonempty subset $S \subseteq \{x_1, ..., x_n\}$ such that $\sum S \in [(1 - \epsilon)T, (1 + \epsilon)T]$; returns the empty set if no such subset exists. */*

1. Let S be an empty set.
2. For i from 1 to n do
   A. $S \leftarrow S \cup \{x_i\}$.
   B. If $\sum S \geq (1 - \epsilon)T$ then return S.
3. Return $\emptyset$.

Note that since $T > 0$, $S \neq \emptyset$ is necessary when we successfully return a subset sum S; hence we distinguish between "successful" and "unsuccessful" outputs by whether the algorithm returns a nonempty or empty set.

Runtime. The algorithm above has asymptotic runtime in $O(n)$, which gives us a polynomial-time solution. ■

Correctness. We prove correctness by showing (i) if we successfully return a nonempty subset S, then $\sum S$ is in the target range and (ii) if we do not successfully return a nonempty subset, it is indeed impossible for a subset of $\{x_1, ..., x_n\}$ to sum to the target range.

(i) First, we assume that the algorithm successfully returns some nonempty subset $S = \{x_1, ..., x_i\}$ for some $i \in [1, n]$. Our return condition guarantees that $\sum S \geq (1 - \epsilon)T$; to complete the proof, we show that $\sum S \leq (1 + \epsilon)T$.

We already know that $\sum S \setminus \{x_i\} < (1 - \epsilon)T$, otherwise, we would have already returned $S = \{x_1, ..., x_{i-1}\}$ in a previous iteration. But $x_i \leq \epsilon T$, so

$$\sum S = x_i + \sum S \setminus \{x_i\} \leq \epsilon\, T + (1 - \epsilon)T = T \leq (1 + \epsilon)T$$

(ii) Now assume that we return an empty subset, indicating that the algorithm did not find a successful subset. Then for all i, $\sum_{j \in [1,i]} x_j < (1 - \epsilon)T$; in particular, $\sum_{j \in [1,n]} x_j < (1 - \epsilon)T$, where all $x_j > 0$. Clearly, this means that no subset of $\{x_1, ..., x_n\}$ will be large enough to reach the target range.

■

□

Exercise 11.8.2. 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. We can leverage the fact that if each $x_i > \epsilon T$, then any feasible subset summing to at most T can contain *at most* $\frac{1}{\epsilon}$ items. Hence, we can afford to enumerate all subsets of size up to $\frac{1}{\epsilon}$.

approx-subset-sum-big($\{x_1, \ldots, x_n\}, \epsilon, T$):

/* *returns a subset S whose sum is in $[(1-\epsilon)T,\ (1+\epsilon)T]$ if possible; otherwise returns $\emptyset$.* */

1. For k from 0 to $\lfloor 1/\epsilon \rfloor$ do
   - A. Enumerate all subsets of $\{x_1, \ldots, x_n\}$ of size exactly k, and call each such subset S_k.
   - B. If $\sum S_k \in [(1-\epsilon)T,\ (1+\epsilon)T]$, then return S_k.
2. Return $\emptyset$.

Runtime. Since each $x_i > \epsilon T$, any subset of size $> \frac{1}{\epsilon}$ would exceed T. Thus we can simply check all subsets of size at most $\frac{1}{\epsilon}$. The number of such subsets is

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{\lfloor 1/\epsilon \rfloor},$$

which is $O(n^{1/\epsilon})$. This is polynomial time for fixed ϵ. ■

Correctness. If there exists a subset summing to exactly T, then any such subset must have size at most $\frac{1}{\epsilon}$. We enumerate all such subsets such that if a feasible subset S exists, we will find one with sum in $[(1-\epsilon)T,\ (1+\epsilon)T]$. Indeed, notice that $\sum S \le T \le (1+\epsilon)T$, and $\sum S \ge T \ge (1-\epsilon)T$ trivially if it equals T.

If we return the empty set, then no combination of up to $\frac{1}{\epsilon}$ items falls within the target interval. In particular, no subset can sum exactly to T.

Thus the procedure satisfies the $(1 \pm \epsilon)$-approximation requirement. ■

□

Exercise 11.8.3. Now give a polynomial time $(1 \pm \epsilon)$-approximation algorithm for subset sum in the general setting (with both big and small inputs).

Solution. We combine the ideas from 11.8.1 and 11.8.2. Split the input into two sets:

$$\mathcal{B} = \{\, x_i \mid x_i > \epsilon T \}, \quad \mathcal{S} = \{\, x_i \mid x_i \leq \epsilon T \}.$$

`approx-subset-sum-general(`$\{x_1, \dots, x_n\}, \epsilon, T$`):`

/ returns a subset S whose sum is in $[(1-\epsilon)T,\ (1+\epsilon)T]$ if possible; otherwise returns $\emptyset$. */*

1. Enumerate all subsets of $\mathcal{B}$ of size up to $\lfloor 1/\epsilon \rfloor$.
2. For each such subset $B \subseteq \mathcal{B}$:
   A. Let $C \leftarrow T - \sum B$ be the remaining capacity for the small items.
   B. $S_{\text{small}} \leftarrow$ `approx-subset-sum(`$\mathcal{S}, \epsilon, C$`)`
   C. If $\sum B + \sum S_{\text{small}} \in [(1-\epsilon)T,\ (1+\epsilon)T]$
      1. Return $B \cup S_{\text{small}}$.
3. Return $\emptyset$.

Runtime. We only enumerate subsets of $\mathcal{B}$ up to size $\frac{1}{\epsilon}$, which is at most $O(n^{1/\epsilon})$. For each such subset, we run `approx-subset-sum(`$\mathcal{S}, \epsilon, C$`)` in $O(n)$. Thus for fixed ϵ, the algorithm is polynomial in n and $(1/\epsilon)$. ■

Correctness. We consider two cases:

Case 1: Suppose there exists a subset

$$S^* \subseteq \{x_1, \dots, x_n\} \quad \text{with} \quad \sum_{x \in S^*} x = T.$$

Partition S^* into big and small items by letting

$$B^* = S^* \cap \mathcal{B} \quad \text{and} \quad S^*_{\text{small}} = S^* \cap \mathcal{S},$$

so that $S^* = B^* \cup S^*_{\text{small}}$. When the algorithm enumerates subsets of $\mathcal{B}$, it will eventually consider a subset B corresponding to B^*. Define the residual capacity

$$C = T - \sum_{x \in B} x.$$

Then, calling $S_{\text{small}} =$ `approx-subset-sum(`$\mathcal{S}, \epsilon, C$`)` yields a subset of $\mathcal{S}$ whose sum is within a factor of $(1 \pm \epsilon)$ of C. Consequently, the combined sum $\sum [B \cup S_{\text{small}}]$ falls within the interval $[(1-\epsilon)T,\ (1+\epsilon)T]$.

Case 2: If no subset of the input sums to T, then the algorithm returns $\emptyset$. This outcome is correct, as it does not falsely claim the existence of a feasible subset. ■

□