sack which you are packing in preparation for a long hike in the wilderness. You have a large number of items (say, k items) of volume v_i, $i = 0, \ldots, k-1$, to fit into the knapsack, which holds a total volume V. Suppose that you are an experienced knapsack packer, and can always fit items in with no wasted space. You want to take the biggest load possible, so you want to find some subset of the k items that exactly fills the knapsack. In other words, you want to find some subset $I \subset \{1, \ldots, k\}$ such that $\sum_{i \in I} v_i = V$, if such a subset exists. This is the *general knapsack problem.* We shall further assume that V and all of the v_i are positive integers. An equivalent way to state the problem is then as follows:

The knapsack problem. Given a set $\{v_i\}$ of k positive integers and an integer V, find a k-bit integer $n = (\epsilon_{k-1}\epsilon_{k-2}\cdots\epsilon_1\epsilon_0)_2$ (where the $\epsilon_i \in \{0, 1\}$ are the binary digits of n) such $\sum_{i=0}^{k-1} \epsilon_i v_i = V$, if such an n exists.

Note that there may be no solution n or many solutions, or there might be a unique solution, depending on the k-tuple $\{v_i\}$ and the integer V.

A special case of the knapsack problem is the *superincreasing knapsack problem.* This is the case when the v_i, arranged in increasing order, have the property that each one is greater than the *sum* of all of the earlier v_i.

Example 1. The 5-tuple $(2, 3, 7, 15, 31)$ is a superincreasing sequence.

It is known that the general knapsack problem is in a very difficult class of problems, called "NP-complete" problems. This means that it is equivalent in difficulty to the notorious "traveling salesman problem." In particular, if the central conjecture in complexity theory is true, as most everyone believes it is, then there does not exist an algorithm which solves an arbitrary knapsack problem in time polynomial in k and $\log B$, where B is a bound on the size of V and the v_i.

However, the superincreasing knapsack problem is much, much easier to solve. Namely, we look down the v_i, starting with the largest, until we get to the first one that is $\leq V$. We include the corresponding i in our subset I (i.e., we take $\epsilon_i = 1$), replace V by $V - v_i$, and then continue down the list of v_i until we find one that is less than or equal to this difference. Continuing in this way, we eventually either obtain a subset of $\{v_i\}$ which sums to V, or else we exhaust all of $\{v_i\}$ without getting $V - \sum_{i \in I} v_i$ equal to 0, in which case there is no solution. We now write the algorithm in a more formal way that could be easily converted to a computer program.

The following polynomial time algorithm solves the knapsack problem for a given superincreasing k-tuple $\{v_i\}$ and integer V:

1. Set W equal to V, and set $j = k$.
2. Starting with ϵ_{j-1} and decreasing the index of ϵ, choose all of the ϵ_i equal to 0 until you get to the first i — call it i_0 — such that $v_{i_0} \leq W$. Set $\epsilon_{i_0} = 1$.
3. Replace W by $W - v_{i_0}$, set $j = i_0$, and, if $W > 0$, go back to step 2.
4. If $W = 0$, you're done. If $W > 0$, and all of the remaining v_i are $> W$, then you know there is no solution $n = (\epsilon_{k-1}\cdots\epsilon_0)_2$ to the problem. Notice that the solution (if there is one) is unique.