$K = 2^n - 1$ (the all-ones vector, corresponding to the universe). The reduction seems complete!

But there is a "bug" in this reduction: Binary integer addition is different from set union in that it has *carry*. For example, $3 + 5 + 7 = 15$ in bit-vector form is $0011 + 0101 + 0111 = 1111$; but the corresponding sets $\{3, 4\}$,$\{2, 4\}$, and $\{2, 3, 4\}$ are not disjoint, neither is their union $\{1, 2, 3, 4\}$. There is a simple and clever way around this problem: Think of these vectors as integers not in base 2, *but in base* $n + 1$. That is, set S_i becomes integer $w_i = \sum_{j \in S_i} (n+1)^{3m-j}$. Since now there can be no carry in any addition of up to n of these numbers, it is straightforward to argue that there is a set of these integers that adds up to $K = \sum_{j=0}^{3m-1} (n+1)^j$ if and only if there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

Pseudopolynomial Algorithms and Strong NP-completeness

In view of Theorem 9.10, the following result seems rather intriguing:

Proposition 9.4: Any instance of KNAPSACK can be solved in $\mathcal{O}(nW)$ time, where n is the number of items and W is the weight limit.

Proof: Define $V(w, i)$ to be the largest value attainable by selecting some among the i first items *so that their total weight is exactly* w. It is easy to see that the nW entries of the $V(w, i)$ table can be computed in order of increasing i, and with a constant number of operations per entry, as follows:

$$V(w, i+1) = \max\{V(w, i), v_{i+1} + V(w - w_{i+1}, i)\}$$

To start, $V(w, 0) = 0$ for all w. Finally, the given instance of KNAPSACK is a "yes" instance if and only if the table contains an entry greater than or equal to the goal K. □

Naturally, Proposition 9.4 does not establish that **P** = **NP** (so, keep on reading this book!). This is not a polynomial algorithm because its time bound nW is *not a polynomial function of the input:* The length of the input is something like $n \log W$. We have seen this pattern before in our first attempt at an algorithm for MAX FLOW in Section 1.2, when the time required was again polynomial in the integers appearing in the input (instead of their logarithms, which is always the correct measure). Such "pseudopolynomial" algorithms are a source not only of confusion, but of genuinely positive results (see Chapter 13 on approximation algorithms).

In relation to pseudopolynomial algorithms, it is interesting to make the following important distinction between KNAPSACK and the other problems that we showed **NP**-complete in this chapter—SAT, MAX CUT, TSP (D), CLIQUE, TRIPARTITE MATCHING, HAMILTON PATH, and many others. All these latter problems were shown **NP**-complete via reductions that constructed only *polynomially small integers.* For problems such as CLIQUE and SAT, in which integers

are only used as node names and variable indices, this is immediate. But even for TSP (D), in which one would expect numbers to play an important role as intercity distances, we only needed distances no larger than *two* to establish **NP**-completeness (recall the proof of the Corollary to Theorem 9.7)†. In contrast, in our **NP**-completeness proof for KNAPSACK we had to create *exponentially large integers* in our reduction.

If a problem remains **NP**-complete even if any instance of length n is restricted to contain integers of size at most $p(n)$, a polynomial, then we say that the problem is *strongly* **NP**-*complete*. All **NP**-complete problems that we have seen so far in this chapter, with the single exception of KNAPSACK, are strongly **NP**-complete. It is no coincidence then that, of all these problems, only KNAPSACK can be solved by a pseudopolynomial algorithm: It should be clear that strongly **NP**-complete problems *have no pseudopolynomial algorithms*, unless of course $\mathbf{P} = \mathbf{NP}$ (see Problem 9.5.31).

We end this chapter with a last interesting example: A problem which involves numbers and bearing a certain similarity to KNAPSACK, but turns out to be strongly **NP**-complete.

BIN PACKING: We are given N positive integers $a_1, a_2, \ldots, a_N$ (the *items*), and two more integers C (the *capacity*) and B (the *number of bins*). We are asked whether these numbers can be partitioned into B subsets, each of which has total sum at most C.

Theorem 9.11: BIN PACKING is **NP**-complete.

Proof: We shall reduce TRIPARTITE MATCHING to it. We are given a set of boys $B = \{b_1, b_2, \ldots, b_n\}$, a set of girls $G = \{g_1, g_2, \ldots, g_n\}$, a set of homes $H = \{h_1, h_2, \ldots, h_n\}$, and a set of triples $T = \{t_1, \ldots, t_m\} \subseteq B \times G \times H$; we are asked whether there is a set of n triples in T, such that each boy, girl, and home is contained in one of the n triples.

The instance of BIN PACKING that we construct has $N = 4m$ items—one for each triple, and one for each occurrence of a boy, girl, or home to a triple. The items corresponding to the occurrences of b_1, for example, will be denoted by $b_1[1], b_1[2], \ldots, b_1[N(b_1)]$, where $N(b_1)$ is the number of occurrences of b_1 in the triples; similarly for the other boys, the girls, and the homes. The items corresponding to triples will be denoted simply t_j.

The sizes of these items are shown in Figure 9.11. M is a very large number, say $100n$. Notice that one of the occurrences of each boy, girl, and home (arbitrarily the first) has different size than the rest; it is this occurrence that will participate in the matching. The capacity C of each bin is $40M^4+15$—

† To put it otherwise, these problems would remain **NP**-complete if numbers in their instances were represented *in unary*—even such wasteful representation would increase the size of the instance by a only polynomial amount, and thus the reduction would still be a valid one.

just enough to fit a triple and one occurrence of each of its three members *as long as either all three or none of the three are a first occurrence.* There are m bins, as many as triples.

Item	Size
first occurrence of a boy $b_i[1]$	$10M^4 + iM + 1$
other occurrences of a boy $b_i[q], q > 1$	$11M^4 + iM + 1$
first occurrence of a girl $g_j[1]$	$10M^4 + jM^2 + 2$
other occurrences of a girl $g_j[q], q > 1$	$11M^4 + jM^2 + 2$
first occurrence of a home $h_k[1]$	$10M^4 + kM^3 + 4$
other occurrences of a home $h_k[q], q > 1$	$8M^4 + kM^3 + 4$
triple $(b_i, g_j, h_k) \in T$	$10M^4 + 8 - iM - jM^2 - kM^3$

Figure 9.11. The items in BIN PACKING.

Suppose that there is a way to fit these items into m bins. Notice immediately that the sum of all items is mC (the total capacity of all bins), and thus all bins must be exactly full. Consider one bin. It must contain four items (proof: all item sizes are between $\frac{1}{5}$ and $\frac{1}{3}$ of the bin capacity). Since the sum of the items modulo M must be 15 ($C \bmod M = 15$), and there is only one way of creating 15 by choosing four numbers (with repetitions allowed) out of 1, 2, 4, and 8 (these are the residues of all item sizes, see Figure 9.11), the bin must contain a triple that contributes 8 mod M, say (b_i, g_j, h_k), and occurrences of a boy $b_{i'}$, a girl $g_{j'}$, and a home $h_{k'}$, contributing 1, 2, and 4 mod 15, respectively. Since the sum modulo M^2 must be 15 as well, we must have $(i' - i) \cdot M + 15 = 15 \bmod M^2$, and thus $i = i'$. Similarly, taking the sum modulo M^3 we get $j = j'$, and modulo M^4 we get $k = k'$. Thus, each bin contains a triple $t = (b_i, g_j, h_k)$, together with one occurrence of b_i, one of g_j, and one of h_k. Furthermore, either all three occurrences are first occurrences, or none of them are—otherwise $40M^4$ cannot be achieved. Hence, there are n bins that contain only first occurrences; *the n triples in these bins form a tripartite matching.*

Conversely, if a tripartite matching exists, we can fit all items into the m bins by matching each triple with occurrences of its members, making sure that the triples in the matching get first occurrences of all three members. The proof is complete. □

Notice that the numbers constructed in this reduction are polynomially large —$\mathcal{O}(|x|^4)$, where x is the original instance of TRIPARTITE MATCHING. Hence, BIN PACKING is strongly **NP**-complete: Any pseudopolynomial algo-

rithm for BIN PACKING would yield a polynomial algorithm for TRIPARTITE MATCHING, implying $\mathbf{P} = \mathbf{NP}$.

BIN PACKING is a useful point of departure for reductions to problems in which numbers appear to play a central role, but which, unlike KNAPSACK (at least as far as we know), are strongly **NP**-complete.