1. (a) We adapt the linear program given at the bottom of page 860 of the text. This LP is associated with a max flow problem given by a directed graph $G = (V,E)$ in which each edge (u,v) in E has a nonnegatve capacity $c(u,v) \geqslant 0$. We set $c(u,v) = 0$ if (u,v) is not an edge, and in particular $c(u,u) = 0$ for all $u \in V$. A flow is a nonnegative real-valued function $f : V \times V \to \mathbb{R}$ that satisfies the capacity constraint and flow conservation. A maximum flow is a flow that satisfies these constraints and maximizes the flow value.

   We use the variable f_{uv} to stand for $f(u,v)$, for each pair (u,v) of vertices. For this exercise, the linear program is the same as that in the text, except that the node constraint for a node $u \in V - \{s,t\}$ is modified so that the total flow out of u is equal to $(1-\epsilon_u)$ times the total flow into u.

   Here is the resulting modified linear program:

   - maximize: $\sum_{v\in V} f_{sv} - \sum_{v\in V} f_{vs}$
   - subject to:

   $$\begin{array}{rcll} f_{uv} & \leqslant & c(u,v) & \text{for each } u,v \in V, \\ (1-\epsilon_u)\sum_{v\in V} f_{vu} & = & \sum_{v\in V} f_{uv} & \text{for each } u \in V - \{s,t\}, \\ f_{uv} & \geqslant & 0 & \text{for each } u,v \in V. \end{array}$$

   Every valid flow in the network yields a feasible solution to the linear program because flow values satisfy each constraint in the linear program. Hence, the maximum value of the objective function is at least as large as the maximum flow.

   Conversely, every feasible solution to the linear program yields a valid flow in the network because every constraint on the flow is represented by some linear inequality. Hence, the maximum value of the objective function is no larger than the maximum flow.

   (b) Assume that $V = \{0,1,\dots,|V|-1\}$. We code a pair (u,v) of vertices using the pairing function $\langle u,v\rangle = u|V| + v + 1$. Thus for $1 \leqslant i \leqslant n = |V|^2$ the variable x_i plays the role of the variable f_{uv} in part (a), where $i = \langle u,v\rangle$.

   To recover the pair (u,v) from i we use the inverse pairing functions

   $$\text{left}(i) = \big\lfloor (i-1)/|V| \big\rfloor \qquad\qquad \text{right}(i) = i - 1 - \text{left}(i)|V|$$

   So if $i = \langle u,v\rangle$, then $u = \text{left}(i)$ and $v = \text{right}(i)$.

   The software at our disposal will minimize $c \cdot x$ subject to constraints. We observe that in general maximizing a function g is the same as minimizing $-g$, so we can translate the maximizing problem from part (a) to the problem of minimizing $\sum_{v\in V} f_{vs} - \sum_{v\in V} f_{sv}$.

   Thus we can define the vector c as follows: for $1 \leqslant i \leqslant n = |V|^2$,

   $$c_i = \begin{cases} 1 & \text{if right}(i) = s, \\ -1 & \text{if left}(i) = s \text{ and right}(i) \neq s, \\ 0 & \text{otherwise.} \end{cases}$$

   Next we define an $[m \times n]$ matrix A and an $[m \times 1]$ vector b such that the constraints $Ax \geqslant b$ are equivalent to the constraints given in the solution to part (a). For this we keep in mind that in general $y \leqslant z$ is equivalent to $-y \geqslant -z$ and $y = z$ is equivalent to the conjunction of $y \geqslant z$ and $-y \geqslant -z$.

   We partition the rows of A into blocks, and the rows of b into similar blocks, where each pair of blocks is responsible for a different set of constrains, as follows.

   - Rows $1,\dots,n = |V|^2$ ensure $x_i \geqslant 0$ for each i, and hence $f_{uv} \geqslant 0$ for all $u,v \in V$. Thus if $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant n$ then $b_i = 0$ and

   $$A_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

- Rows $n+1 \leqslant i \leqslant 2n$ ensure $-x_\ell \geqslant -c(\text{left}(\ell), \text{right}(\ell))$ for each i, where $\ell = i - n$, and hence $f_{uv} \leqslant c(u,v)$ for all $u, v \in V$. Thus for $n+1 \leqslant i \leqslant 2n$ and for $1 \leqslant j \leqslant n$ let $b_i = -c(\text{left}(i-n), \text{right}(i-n))$ and

$$A_{ij} = \begin{cases} -1 & \text{if } i - n = j, \\ 0 & \text{otherwise.} \end{cases}$$

- Rows $2n+1 \leqslant i \leqslant 2n+|V|$ ensure for each $v \in V$ that $(1-\epsilon_v)\sum_{u \in V} f_{uv} \geqslant \sum_{u \in V} f_{vu}$. (Here we have interchanged the roles of v and u in this set of constraints because of the way we defined the pairing function: note that $v = \text{right}(i-2n)$ runs through all of V as i runs from $2n+1$ to $2n+|V|$.) For $2n+1 \leqslant i \leqslant 2n+|V|$ we define $v = v_i = \text{right}(i-2n)$. Thus for these values of i and for $1 \leqslant j \leqslant n$ we set $b_i = 0$ and define

$$A_{ij} = \begin{cases} 1 - \epsilon_{v_i} & \text{if right}(j) = v_i, \\ -1 & \text{if left}(j) = v_i \text{ and right}(j) \neq v_i, \\ 0 & \text{otherwise.} \end{cases}$$

- Rows $2n+|V|+1 \leqslant i \leqslant 2n+2|V|$ ensure for each $v \in V$ that $-(1-\epsilon_v)\sum_{u \in V} f_{uv} \geqslant -\sum_{u \in V} f_{vu}$. The definition of A_{ij} for these rows is the same as for the previous case, except the top row has $\epsilon_{v_i} - 1$ and the middle row has $+1$.

Thus the resulting matrix A has $m = 2|V|^2 + 2|V|$ rows and $n = |V|^2$ columns, and the constraints $Ax \geqslant b$ are equivalent to the constraints given in part (a).

2. Let D be a decision problem in *NP*. For any input x, we write "$D(x)$" to denote the answer (TRUE or FALSE) of problem D on input x. Then by definition (see page 1064 of the text), there is a two-input polynomial-time algorithm A and constants c and d such that for all $x \in \{0,1\}^*$

$$D(x) \iff \text{there exists a certificate } y \text{ with } |y| \leqslant d|x|^c \text{ such that } A(x,y) = \text{TRUE} \tag{1}$$

Thus for each input x of length n the total number of possible strings that are potential certificates is the number of binary strings of length at most dn^c, which is at most 2^{dn^c}.

Here is an algorithm $A'(x)$ for solving decision problem $D(x)$:

```
Algorithm $A'(x)$:
    $n \leftarrow |x|$
    for every string $y$ of length at most $dn^c$:
        **if** $A(x,y) = 1$:
            **return** TRUE    # and exit
    **return** FALSE
```

It is immediate from (1) that $A'(x) = D(x)$ for all inputs x.

We can upper bound the runtime of A' as follows. The number of iterations of the loop is at most the number of strings y of length at most dn^c, which is at most 2^{dn^c} (see above). The time required for each iteration is dominated by the time required to compute $A(x,y)$, where $|y| \leqslant dn^c$. Since the composition of two polynomials is again a polynomial, the time required to compute $A(x,y)$ is bounded by some polynomial $q(n)$. Thus the total time required for $A'(x)$ is $\mathcal{O}\left(q(n)2^{dn^c}\right) = \mathcal{O}\left(2^{p(n)}\right)$ for some polynomial $p(n)$, as required.

3. First we show that FP $\in NP$. Given a YES instance $(L, s_1, \ldots, s_n, k)$ of FP, a certificate is a partition $P_1, \ldots, P_k$ of $\{1, \ldots, n\}$ satisfying the conditions specified in the definition of FP. A verifier can easily check in polynomial time that the certificate satisfies all of these conditions.

   To show that FP is *NP*-hard, we show that Subset-Sum $\leqslant_p$ FP. (We know from lectures that Subset-Sum in *NP*-hard.)

   Our task is to give a polytime algorithm which transforms a sequence $I = (x_1, \ldots, x_\ell, t)$ of positive integers (in binary), describing an instance of Subset-Sum, to an instance $I' = (L, s_1, \ldots, s_n, k)$ of FP such that I is a YES instance of Subset-Sum iff I' is a YES instance of FP.

   Let $u = x_1 + x_2 + \cdots + x_\ell$.

   Case 1: If $u = 2t$, then let $L = t, k = 2, n = \ell$, and $s_1, \ldots, s_n = x_1, \ldots, x_\ell$.

   Case 1: If $u < 2t$, then let $L = t, k = 2, n = \ell + 1$, and $s_1, \ldots, s_n = x_1, \ldots, x_\ell, x_{\ell+1}$, where $x_{\ell+1} = 2t - u$.

   Case 2: If $u > 2t$, then let $L = u - t, k = 2, n = \ell + 1$, and $s_1, \ldots, s_n = x_1, \ldots, x_\ell, x_{\ell+1}$, where $x_{\ell+1} = 2(u - t) - u = u - 2t$.

   Clearly, $L, s_1, \ldots, s_n, k$ can be computed in polytime from $x_1, \ldots, x_\ell, t$ because all arithmetic comparisons and operations can be performed in polytime for binary integers.

   Suppose I is a YES instance of Subset-Sum. Then there exists $S \subseteq \{1, \ldots, \ell\}$ such that $\sum_{i \in S} x_i = t$.

   Case 1: If $u = 2t$, then $L = t$. Let $P_1 = S$ and $P_2 = \{1, \ldots, n\} - S$. Then $\sum_{i \in P_1} s_i = \sum_{i \in S} x_i = t \leqslant L$ and $\sum_{i \in P_2} s_i = \sum_{i=1}^{\ell} x_i - \sum_{i \in S} x_i = u - t = t \leqslant L$.

   Case 2: If $u < 2t$, then $L = t$. Let $P_1 = S$ and $P_2 = \{1, \ldots, n\} - S$. Then $\sum_{i \in P_1} s_i = \sum_{i \in S} x_i = t \leqslant L$ and $\sum_{i \in P_2} s_i = \sum_{i=1}^{\ell+1} x_i - \sum_{i \in S} x_i = x_{\ell+1} + u - t = (2t - u) + u - t = t \leqslant L$.

   Case 3: If $u > 2t$, then $L = u - t$. Let $P_1 = \{1, \ldots, \ell\} - S$ and $P_2 = S \cup \{\ell + 1\}$. Then $\sum_{i \in P_1} x_i = u - t \leqslant L$ and $\sum_{i \in P_2} x_i = (\sum_{i \in S} x_1) + x_{\ell+1} = t + (u - 2t) = u - t \leqslant L$.

   In all cases, I' is a YES instance of FP, as required.

   Conversely, suppose that I' is a YES instance of FP. Then there exists a partition P_1, P_2 of $\{1, \ldots, n\}$ such that $\sum_{i \in P_1} x_i \leqslant L$ and $\sum_{i \in P_2} x_i \leqslant L$.

   Case 1: If $u = 2t$, then $L = t$ and $\sum_{i \in P_1} x_i + \sum_{i \in P_2} x_i = x_1 + \cdots + x_\ell = u = 2t = 2L$, so both P_1 and P_2 must have sum *exactly* equal to L. Let $S = P_1$. Then $\sum_{i \in S} x_i = \sum_{i \in P_1} x_i = t$.

   Case 1: If $u < 2t$, then $L = t$ and $\sum_{i \in P_1} x_i + \sum_{i \in P_2} x_i = x_1 + \cdots + x_\ell + x_{\ell+1} = u + (2t - u) = 2t = 2L$, so both P_1 and P_2 must have sum *exactly* equal to L. Let $S = P_1$ or $S = P_2$, whichever set does *not* contain $x_{\ell+1}$. Then $\sum_{i \in S} x_i = L = t$.

   Case 2: If $u > 2t$, then $L = u - t$ and $\sum_{i \in P_1} x_i + \sum_{i \in P_2} x_i = x_1 + \cdots + x_\ell + x_{\ell+1} = u + (u - 2t) = 2(u - t) = 2L$, so both P_1 and P_2 must have sum *exactly* equal to L. Let $S = P_1 - \{x_{\ell+1}\}$ or $S = P_2 - \{x_{\ell+1}\}$ (depending on which packet contains $x_{\ell+1}$). Then $\sum_{i \in S} x_i = L - (u - 2t) = (u - t) - (u - 2t) = t$.

   In all cases, I is a YES instance of Subset-Sum, as required.

   (Note: Many other reductions were possible. For example, we could set $L = 2u$, $s_1, \ldots, s_n = x_1, \ldots, x_\ell, (u + t), (2u - t)$, and $k = 2$.)

4. Here is the MinPackets optimization problem;

   - *Input:* Positive integers $L, x_1, \ldots, x_n$.
   - *Output:* Packets $P_1, \ldots, P_k$ such that $P_1 \cup \cdots \cup P_k = \{1, \ldots, n\}$, $\forall i \neq j, P_i \cap P_j = \varnothing$, $\forall j, \sum_{i \in P_j} x_i \leqslant L$, and the number of packets (k) is as small as possible.

Now suppose that FP is an algorithm that decides the FP decision problem. The following algorithm shows that the MINPACKETS optimization problem is polynomial time reducible to the FP decision problem.

```
MP($L, x_1, \ldots, x_n$):
    **if** $x_1 > L$ **or** ... **or** $x_n > L$:   **return** $\varnothing$   *[no solution possible]*
    Perform binary search in the range $[1 \ldots n]$ to find a value $k$ such that
        $\text{FP}(L, x_1, \ldots, x_n, k) = $ TRUE but $\text{FP}(L, x_1, \ldots, x_n, k+1) = $ FALSE.
    $P_1 \leftarrow \varnothing, \ldots, P_k \leftarrow \varnothing$          *[current packets]*
    $t_1 \leftarrow 0, \ldots, t_k \leftarrow 0$          *[$t_i = \sum_{j \in P_i} s_j$]*
    **for** $i \leftarrow 1, \ldots, n$:
        **for** $j \leftarrow 1, \ldots, k$:        *[try putting $x_i$ in $P_j$]*
            **if** $\text{FP}(L, x_{i+1}, \ldots, x_n, t_1, \ldots, t_{j-1}, t_j + x_i, t_{j+1}, \ldots, t_k, k)$:
                $P_j \leftarrow P_j \cup \{i\}$
                $t_j \leftarrow t_j + x_i$
                $j \leftarrow k + 1$      *[break out of the loop on $j$]*
    **return** $P_1, \ldots, P_k$
```

Correctness: After ensuring that a solution is possible, the algorithm find the smallest value of k—there must be one in the range $[1 \ldots n]$ because it is always possible to put each request in its own separate packet ($\text{FP}(L, x_1, \ldots, x_n, n)$ is always TRUE).

Then, the algorithm examines each request in turn, and tries to put it in each packet in turn—one of them must work. The loop invariant for the loop on i is that $\text{FP}(L, x_i, \ldots, x_n, t_1, \ldots, t_k, k) =$ TRUE where $t_i = \sum_{j \in P_i} s_j$ for each i. This means that at the end, $P_1, \ldots, P_k$ is a valid partition of the requests into k packets.

Runtime: Let $t(m)$ be the running time of FP on inputs whose total size is m (i.e., m accounts for all of the bits required to write L, each of $x_1, \ldots, x_n$, and k). Then MP makes $\log_2 n + kn$ calls to FP on inputs of size $\mathcal{O}(m)$, in addition to a linear amount of work (at most) for each iteration of the inner loop. Hence, the total running time of MP is $\mathcal{O}\left(m^2 t(m) + m^2\right)$.