Solution to Problem Q1:

Note: We assume that no word is longer than will fit into a line, i.e., $l_i \leq M$ for all i.

First, we'll make some definitions so that we can state the problem more uniformly. Special cases about the last line and worries about whether a sequence of words fits in a line will be handled in these definitions, so that we can forget about them when framing our overall strategy.

- Define $\mathit{extras}[i, j] = M - j + i - \sum_{k=i}^{j} l_k$ to be the number of extra spaces at the end of a line containing words i through j. Note that *extras* may be negative.
- Now define the cost of including a line containing words i through j in the sum we want to minimize:

$$lc[i, j] = \begin{cases} \infty & \text{if } \mathit{extras}[i, j] < 0 \text{ (i.e., words } i, \ldots, j \text{ don't fit)}, \\ 0 & \text{if } j = n \text{ and } \mathit{extras}[i, j] \geq 0 \text{ (last line costs 0)}, \\ (\mathit{extras}[i, j])^3 & \text{otherwise}. \end{cases}$$

By making the line cost infinite when the words don't fit on it, we prevent such an arrangement from being part of a minimal sum, and by making the cost 0 for the last line (if the words fit), we prevent the arrangement of the last line from influencing the sum being minimized.

We want to minimize the sum of lc over all lines of the paragraph.

Our subproblems are how to optimally arrange words $1, \ldots, j$, where $j = 1, \ldots, n$.

Consider an optimal arrangement of words $1, \ldots, j$. Suppose we know that the last line, which ends in word j, begins with word i. The preceding lines, therefore, contain words $1, \ldots, i-1$. In fact, they must contain an optimal arrangement of words $1, \ldots, i-1$. (The usual type of cut-and-paste argument applies.)

Let $c[j]$ be the cost of an optimal arrangement of words $1, \ldots, j$. If we know that the last line contains words $i, \ldots, j$, then $c[j] = c[i-1] + lc[i, j]$. As a base case, when we're computing $c[1]$, we need $c[0]$. If we set $c[0] = 0$, then $c[1] = lc[1, 1]$, which is what we want.

But of course we have to figure out which word begins the last line for the subproblem of words $1, \ldots, j$. So we try all possibilities for word i, and we pick the one that gives the lowest cost. Here, i ranges from 1 to j. Thus, we can define $c[j]$ recursively by

$$c[j] = \begin{cases} 0 & \text{if } j = 0 , \\ \min\limits_{1 \le i \le j} (c[i-1] + lc[i, j]) & \text{if } j > 0 . \end{cases}$$

Note that the way we defined lc ensures that

- all choices made will fit on the line (since an arrangement with $lc = \infty$ cannot be chosen as the minimum), and
- the cost of putting words $i, \ldots, j$ on the last line will not be 0 unless this really is the last line of the paragraph ($j = n$) or words $i \ldots j$ fill the entire line.

We can compute a table of c values from left to right, since each value depends only on earlier values.

To keep track of what words go on what lines, we can keep a parallel p table that points to where each c value came from. When $c[j]$ is computed, if $c[j]$ is based on the value of $c[k-1]$, set $p[j] = k$. Then after $c[n]$ is computed, we can trace the pointers to see where to break the lines. The last line starts at word $p[n]$ and goes through word n. The previous line starts at word $p[p[n]]$ and goes through word $p[n] - 1$, etc.

In pseudocode, here's how we construct the tables:

```
PRINT-NEATLY(l, n, M)
let extras[1..n, 1..n], lc[1..n, 1..n], and c[0..n] be new arrays
// Compute extras[i, j] for 1 ≤ i ≤ j ≤ n.
for i = 1 to n
    extras[i, i] = M − l_i
    for j = i + 1 to n
        extras[i, j] = extras[i, j − 1] − l_j − 1
// Compute lc[i, j] for 1 ≤ i ≤ j ≤ n.
for i = 1 to n
    for j = i to n
        if extras[i, j] < 0
            lc[i, j] = ∞


        elseif j == n and extras[i, j] ≥ 0
            lc[i, j] = 0
        else lc[i, j] = (extras[i, j])^3
// Compute c[j] and p[j] for 1 ≤ j ≤ n.
c[0] = 0
for j = 1 to n
    c[j] = ∞
    for i = 1 to j
        if c[i − 1] + lc[i, j] < c[j]
            c[j] = c[i − 1] + lc[i, j]
            p[j] = i
return c and p
```

Quite clearly, both the time and space are $\Theta(n^2)$.

In fact, we can do a bit better: we can get both the time and space down to $\Theta(nM)$. The key observation is that at most $\lceil M/2 \rceil$ words can fit on a line. (Each word is at least one character long, and there's a space between words.) Since a line with words $i, \ldots, j$ contains $j - i + 1$ words, if $j - i + 1 > \lceil M/2 \rceil$ then we know that $lc[i, j] = \infty$. We need only compute and store $extras[i, j]$ and $lc[i, j]$ for $j - i + 1 \leq \lceil M/2 \rceil$. And the inner **for** loop header in the computation of $c[j]$ and $p[j]$ can run from $\max(1, j - \lceil M/2 \rceil + 1)$ to j.

We can reduce the space even further to $\Theta(n)$. We do so by not storing the *lc* and *extras* tables, and instead computing the value of $lc[i, j]$ as needed in the last loop. The idea is that we could compute $lc[i, j]$ in $O(1)$ time if we knew the value of $extras[i, j]$. And if we scan for the minimum value in *descending* order of i, we can compute that as $extras[i, j] = extras[i + 1, j] - l_i - 1$. (Initially, $extras[j, j] = M - l_j$.) This improvement reduces the space to $\Theta(n)$, since now the only tables we store are c and p.

Here's how we print which words are on which line. The printed output of GIVE-LINES(p, j) is a sequence of triples (k, i, j), indicating that words $i, \ldots, j$ are printed on line k. The return value is the line number k.

GIVE-LINES(p, j)

$i = p[j]$
if $i == 1$
    $k = 1$
else $k = $ GIVE-LINES$(p, i-1) + 1$
print (k, i, j)
return k

The initial call is GIVE-LINES(p, n). Since the value of j decreases in each recursive call, GIVE-LINES takes a total of $O(n)$ time.