1. A naïve greedy algorithm would place as many words as possible onto each line. This creates sub-optimal results when a long word occurs near the end of a line, forcing an early line break. Since there is no way for the line-breaking algorithm to move the long word earlier, we can only reduce the unwanted end-of-line spaces by moving words from earlier lines forward. This suggests that the decision about whether to break after a particular word depends on what words follow it in the paragraph.

Therefore, let $OPT[i]$ be the minimum cost for neatly printing words $i \ldots n$.

$$
COST[i,j] = \begin{cases} 
\infty & \text{if } (j-i) + \sum_{k=i}^{j} l_k > M \\
0 & \text{if } j = n \\
(M - (i-j) - \sum_{k=i}^{j} l_k)^2 & \text{otherwise}
\end{cases}
$$

$$
OPT[i] = \min_{i \leq j \leq n} COST[i,j] + OPT[i+j]
$$

To actually do the printing, rather than just calculating the optimal cost, we can use an additional array to store the $j$ value for each $OPT[i]$. 
PRINT-NEATLY($M, n, l$)
1 for $i \leftarrow n$ to 1
2 do $length \leftarrow l_i$
3 $j \leftarrow i$
4 $OPT[i] \leftarrow \infty$
5 $last[i] \leftarrow i$
6 while $length \leq M$
7 do if $j = n$
8 then $OPT[i] \leftarrow 0$
9 $last[i] \leftarrow n$
10 exit while
11 else $cost \leftarrow OPT[j] + (M - length)^2$
12 if $cost < OPT[i]$
13 then $OPT[i] \leftarrow cost$
14 $last[i] \leftarrow j$
15 $j \leftarrow j + 1$
16 $length \leftarrow length + l_j + 1$
17 $i \leftarrow 1$
18 while $i < n$
19 do Print words $i$ through $next[i]$
20 Print new line
21 $i \leftarrow next[i] + 1$

The outer loop performs $n$ iterations, and the inner loop will perform at most $M/2$, giving an overall time complexity of $O(nM)$. The $OPT$ and $next$ arrays both have $n$ elements, and therefore require $O(n)$ space.

(10 points)

2. (a) Since $OPT_k[u,v]$ is the shortest path using at most $k$ edges, we can use $OPT_{k-1}$ to look for a path to a neighbor of $v$, and then add the weight of the edge to $v$. Assuming $OPT_k[u,v] = \infty$ if no such path exists, $w_{uv} = \infty$ if no such edge exists, and $OPT_k[u, u] = w_{uu} = 0$, we have:

$$OPT_k[u, v] = \min_t w_{ut} + OPT_{k-1}[t, v]$$

Turning this into an $O(n^4)$ time algorithm is trivial:

SHORTEST-PATH($V, w$)
1 $OPT_1 \leftarrow w$
2 for $k \leftarrow 2$ to $n$
3 do $OPT_k \leftarrow$ Matrix which is 0 on the diagonal and $\infty$ elsewhere
4 for $u \in V$
5 do for $v \in V$
6 do for $t \in V$
7 do if $w_{ut} + OPT_{k-1}[t, v] < OPT_k[u, v]$
8 then $OPT_k[u, v] \leftarrow w_{ut} + OPT_{k-1}[t, v]$
(5 points)
(b) Now, instead of building a k-edge path by adding one edge to a \((k - 1)\)-edge path, we add two \(\lceil k/2 \rceil\)-edge paths.

\[
OPT_k[u, v] = \min_t OPT_{\lceil k/2 \rceil}[u, t] + OPT_{\lceil k/2 \rceil}[t, v]
\]

(Yes, this does potentially create problems if \(k\) is odd, but we will construct our algorithm so that this never happens.)

**Faster-Shortest-Path\((V, w)\)**

1.  
   \(OPT_1 \leftarrow w\)
2.  
   \(k \leftarrow 2\)
3. **repeat**
4.    
   \(OPT_k \leftarrow \text{Matrix which is 0 on the diagonal and \(\infty\) elsewhere}\)
5.    
   for \(u \in V\)
6.    
   do for \(v \in V\)
7.    
   do for \(t \in V\)
8.    
   do if \(OPT_{k/2}[u, t] + OPT_{k/2}[t, v] < OPT_k[u, v]\)
9.    
   then \(OPT_k[u, v] \leftarrow OPT_{k/2}[u, t] + OPT_{k/2}[t, v]\)
10.  
   \(k \leftarrow k \times 2\)
11. until \(k > n\)
12.  
   \(OPT_n \leftarrow OPT_k\)

The three inner loops still run \(n\) times each, but the outer loop now takes at most \(\log_2 n\) iterations, giving a total run time of \(O(n^3 \log n)\).

(5 points)

(c) Given an \(OPT_n\) matrix, we can find the shortest path from \(u\) to \(v\) by repeatedly selecting nodes \(t\) such that \(OPT[u, t] + OPT[t, v] = OPT[u, v]\). This can be done by starting with \(u\), selecting an appropriate \(t\) from its neighbors, and then finding the shortest path from \(t\) to \(v\).

(5 points)

3. (a) Consider this graph:

```
   1  b
 a  2  c
   2
```

Starting with \(S = \{a\}\), the algorithm will first add \(b\) to \(S\), setting \(d_b = 1\). Next, it will add \(c\), setting \(d_c = -1\). Then the algorithm will terminate, as \(V \setminus S = \emptyset\). However, the shortest path from \(a\) to \(b\) goes via \(c\) with length 1.

(5 points)

(b) Consider this graph:

```
   1  1
 x y
 w  1
   2
```

3
Starting with \( S = \{w\} \), the algorithm will first add \( x \), then \( y \), and then \( z \), concluding that \( d_z = 3 \), even though there is a shorter path \([a, d]\) of length 2.

\((5 \text{ points})\)

4. (a) INSERT(3)
   
   \[
   \begin{array}{c}
   3 \\
   \end{array}
   \]

(b) INSERT(10)
   
   \[
   \begin{array}{c}
   3 \\
   | \\
   \_ \\
   | \\
   10 \\
   \end{array}
   \]

(c) INSERT(5)
   
   \[
   \begin{array}{c}
   5 \\
   | \\
   \_ \\
   | \\
   3 \\
   | \\
   \_ \\
   | \\
   10 \\
   \end{array}
   \]

(d) INSERT(8)
   
   \[
   \begin{array}{c}
   3 \\
   | \\
   \_ \\
   | \\
   5 \\
   | \\
   \_ \\
   | \\
   10 \\
   | \\
   \_ \\
   | \\
   8 \\
   \end{array}
   \]

(e) EXTRACT-MIN
   
   \[
   \begin{array}{c}
   10 \\
   | \\
   \_ \\
   | \\
   5 \\
   | \\
   \_ \\
   | \\
   8 \\
   \end{array}
   \]

(f) INSERT(6)
   
   \[
   \begin{array}{c}
   5 \\
   | \\
   \_ \\
   | \\
   6 \\
   | \\
   \_ \\
   | \\
   8 \\
   | \\
   \_ \\
   | \\
   10 \\
   \end{array}
   \]

(g) EXTRACT-MIN

\[
\begin{array}{c}
\end{array}
\]
5. There are a few valid ways to do this, depending on whether you interpret “larger tree” to mean size or depth. Here, I assume depth. If size had been used instead, the trees would be different starting with step (j).

(a) UNION(1,2)

```
  2   3   4   5   6   7   8   9   10
   /   /   /   /   /   /   /   /
  1   2   3   4   5   6   7   8   9   10
```

(b) UNION(3,4)

```
  2   4   5   6   7   8   9   10
   /   /   /   /   /   /   /
  1   2   3   4   5   6   7   8   9   10
```

(c) UNION(5,6)

```
  2   4   6   7   8   9   10
   /   /   /   /   /   /
  1   2   3   4   5   6   7   8   9   10
```

(10 points)
(d) UNION(3,5)

```
  6   7   8   9   10
 /     /     /     /
2     4     5     3
 /     /
1     1
```

(e) UNION(1,3)

```
  2   3   4   5   6   7   8   9   10
 /     /     /     /     /     /     /     /
1     1     1     1     2     3     4     5
```

(f) UNION(7,8)

```
  6
 /     /     /     /     /     /     /     /
2     3     4     5     7     8     9     10
 /     /     /     /     /     /     /
1     1     1     1     2     3     4
```

(g) UNION(9,10)

```
  6
 /     /     /     /     /     /     /
2     3     4     5     7     8     9
 /     /     /     /     /     /
1     1     1     1     2     3
```

(h) UNION(7,9)

```
  6
 /     /     /     /     /     /     /     /
2     3     4     5     7     8     9
 /     /     /     /     /     /     /
1     1     1     1     2     3
```

6
(i) FIND(3)

```
      6
     / \
    2   3
   / \ / \n  1  4  5
```

(j) UNION(1,10)

```
   10
  /   /
 /     \
6      8
```

(k) FIND(7)

```
   10
  /   /
 /     \
6      7
```

(l) FIND(3)

```
   10
  /   /
 /     \
3      8
```

(m) FIND(4)

```
   10
  /   /
 /     \
3      7
```

7
(n) FIND(5)

(10 points)