Solution to Problem 15-1

Taking the book's hint, we sort the points by x-coordinate, left to right, in $O(n \lg n)$ time. Let the sorted points be, left to right, $\langle p_1, p_2, p_3, \ldots, p_n \rangle$. Therefore, p_1 is the leftmost point, and p_n is the rightmost.

We define as our subproblems paths of the following form, which we call bitonic paths. A ***bitonic path*** $P_{i,j}$, where $i \le j$, includes all points $p_1, p_2, \ldots, p_j$; it starts at some point p_i, goes strictly left to point p_1, and then goes strictly right to point p_j. By "going strictly left," we mean that each point in the path has a lower x-coordinate than the previous point. Looked at another way, the indices of the sorted points form a strictly decreasing sequence. Likewise, "going strictly right" means that the indices of the sorted points form a strictly increasing sequence. Moreover, $P_{i,j}$ contains all the points $p_1, p_2, p_3, \ldots, p_j$. Note that p_j is the rightmost point in $P_{i,j}$ and is on the rightgoing subpath. The leftgoing subpath may be degenerate, consisting of just p_1.

Let us denote the euclidean distance between any two points p_i and p_j by $|p_i p_j|$. And let us denote by $b[i, j]$, for $1 \le i \le j \le n$, the length of the shortest bitonic path $P_{i,j}$. Since the leftgoing subpath may be degenerate, we can easily compute all values $b[1, j]$. The only value of $b[i, i]$ that we will need is $b[n, n]$, which is the length of the shortest bitonic tour. We have the following formulation of $b[i, j]$ for $1 \le i \le j \le n$:

$$\begin{aligned}
b[1,2] &= |p_1 p_2| \ , \\
b[i,j] &= b[i, j-1] + |p_{j-1} p_j| \quad \text{for } i < j-1 \ , \\
b[j-1, j] &= \min_{1 \le k < j-1} \{b[k, j-1] + |p_k p_j|\} \ .
\end{aligned}$$

Why are these formulas correct? Any bitonic path ending at p_2 has p_2 as its rightmost point, so it consists only of p_1 and p_2. Its length, therefore, is $|p_1 p_2|$.

Now consider a shortest bitonic path $P_{i,j}$. The point p_{j-1} is somewhere on this path. If it is on the rightgoing subpath, then it immediately preceeds p_j on this subpath. Otherwise, it is on the leftgoing subpath, and it must be the rightmost point on this subpath, so $i = j-1$. In the first case, the subpath from p_i to p_{j-1} must be a shortest bitonic path $P_{i,j-1}$, for otherwise we could use a cut-and-paste argument to come up with a shorter bitonic path than $P_{i,j}$. (This is part of our optimal substructure.) The length of $P_{i,j}$, therefore, is given by $b[i, j-1] + |p_{j-1} p_j|$. In the second case, p_j has an immediate predecessor p_k, where $k < j-1$, on the rightgoing subpath. Optimal substructure again applies: the subpath from p_k to p_{j-1} must be a shortest bitonic path $P_{k,j-1}$, for otherwise we could use cut-and-paste to come up with a shorter bitonic path than $P_{i,j}$. (We have implicitly relied on paths having the same length regardless of which direction we traverse them.) The length of $P_{i,j}$, therefore, is given by $\min_{1 \le k \le j-1} \{b[k, j-1] + |p_k p_j|\}$.

We need to compute $b[n, n]$. In an optimal bitonic tour, one of the points adjacent to p_n must be p_{n-1}, and so we have

$$b[n, n] = b[n-1, n] + |p_{n-1} p_n| \ .$$

To reconstruct the points on the shortest bitonic tour, we define $r[i, j]$ to be the immediate predecessor of p_j on the shortest bitonic path $P_{i,j}$. The pseudocode below shows how we compute $b[i, j]$ and $r[i, j]$:

```
EUCLIDEAN-TSP(p)
sort the points so that ⟨p1, p2, p3, ..., pn⟩ are in order of increasing x-coordinate
b[1, 2] ← |p1 p2|
for j ← 3 to n
    do for i ← 1 to j − 2
            do b[i, j] ← b[i, j − 1] + |p(j−1) pj|
               r[i, j] ← j − 1
       b[j − 1, j] ← ∞
       for k ← 1 to j − 2
            do q ← b[k, j − 1] + |pk pj|
               if q < b[j − 1, j]
                 then b[j − 1, j] ← q
                      r[j − 1, j] ← k
b[n, n] ← b[n − 1, n] + |p(n−1) pn|
return b and r
```

We print out the tour we found by starting at p_n, then a leftgoing subpath that includes p_{n-1}, from right to left, until we hit p_1. Then we print right-to-left the remaining subpath, which does not include p_{n-1}. For the example in Figure 15.9(b) on page 365, we wish to print the sequence $p_7, p_6, p_4, p_3, p_1, p_2, p_5$. Our code is recursive. The right-to-left subpath is printed as we go deeper into the recursion, and the left-to-right subpath is printed as we back out.

```
PRINT-TOUR(r, n)
print p_n
print p_{n-1}
k ← r[n − 1, n]
PRINT-PATH(r, k, n − 1)
print p_k
```

```
PRINT-PATH(r, i, j)
if i < j
  then k ← r[i, j]
       print p_k
       if k > 1
         then PRINT-PATH(r, i, k)
  else k ← r[j, i]
       if k > 1
         then PRINT-PATH(r, k, j)
              print p_k
```

The relative values of the parameters i and j in each call of PRINT-PATH indicate which subpath we're working on. If $i < j$, we're on the right-to-left subpath, and if $i > j$, we're on the left-to-right subpath.

The time to run EUCLIDEAN-TSP is $O(n^2)$ since the outer loop on j iterates $n-2$ times and the inner loops on i and k each run at most $n-2$ times. The sorting step at the beginning takes $O(n \lg n)$ time, which the loop times dominate. The time to run PRINT-TOUR is $O(n)$, since each point is printed just once.