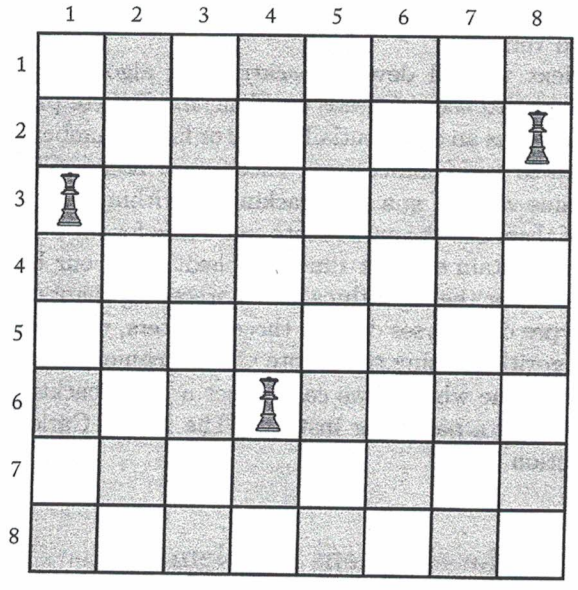

Figure 5.6
The queen in row 6 is being threatened in its left diagonal by the queen in row 3 and in its right diagonal by the queen in row 2.

▶ Algorithm 5.1

The Backtracking Algorithm for the *n*-Queens Problem

Problem: Position n queens on a chessboard so that no two are in the same row, column, or diagonal.

Inputs: positive integer n.

Outputs: all possible ways n queens can be placed on an $n \times n$ chessboard so that no two queens threaten each other. Each output consists of an array of integers *col* indexed from 1 to n, where *col*[*i*] is the column where the queen in the ith row is placed.

```
void queens (index i)
{
  index j;

  if (promising(i))
     if (i == n)
        cout << col[1] through col [n];
     else
        for (j = 1; j <= n; j++){      // See if queen in
           col[i + 1] = j;             // (i + 1)st row can be
           queens(i + 1);              // positioned in each of
                                       // the n columns.
        }
}
```

```
bool promising (index i)
{
   index k;
   bool switch;

   k = 1;
   switch = true;                    // Check if any queen threatens
   while (k < i &&  switch){  // queen in the ith row.
      if (col[i] == col[k] || abs(col[i] - col[k]) == i - k)
         switch = false;
      k++;
   }
   return switch;
}
```

When an algorithm consists of more than one routine, we do not order the routines according to the rules of any particular programming language. Rather, we just present the main routine first. In Algorithm 5.1, that routine is *queens*. Following the convention discussed in Section 2.1, n and *col* are not inputs to the recursive routine queens. If the algorithm were implemented by defining n and *col* globally, the top-level call to *queens* would be

```
queens(0);
```

Algorithm 5.1 produces *all* solutions to the n-Queens problem because that is how we stated the problem. We stated the problem this way to eliminate the need to exit when a solution is found. This makes the algorithm less cluttered. In general, the problems in this chapter can be stated to require one, several, or all solutions. In practice, the one that is done depends on the needs of the application. Most of our algorithms are written to produce all solutions. It is a simple modification to make the algorithms stop after finding one solution.

It is difficult to analyze Algorithm 5.1 theoretically. To do this, we have to determine the number of nodes checked as a function of n, the number of queens. We can get an upper bound on the number of nodes in the pruned state space tree by counting the number of nodes in the entire state space tree. This latter tree contains 1 node at level 0, n nodes at level 1, n^2 nodes at level 2, ..., and n^n nodes at level n. The total number of nodes is

$$1 + n + n^2 + n^3 + \cdots + n^n = \frac{n^{n+1} - 1}{n - 1}.$$