```
1     /**
2      * Performs the standard binary search
3      * using one comparison per level.
4      * @exception ItemNotFound if appropriate.
5      * @return index where item is found.
6      */
7     public static int binarySearch( Comparable [ ] a,
8                                     Comparable x ) throws ItemNotFound
9     {
10        if( a.length == 0 )
11            throw new ItemNotFound( "BinarySearch fails" );
12
13        int low = 0;
14        int high = a.length - 1;
15        int mid;
16
17        while( low < high )
18        {
19            mid = ( low + high ) / 2;
20
21            if( a[ mid ].compares( x ) < 0 )
22                low = mid + 1;
23            else
24                high = mid;
25        }
26
27        if( a[ low ].compares( x ) == 0 )
28            return low;
29
30        throw new ItemNotFound( "BinarySearch fails" );
31    }
32 }
```

Figure 5.12 Binary search using two-way comparisons

1. Each access must be very expensive compared to a typical instruction. For example, the array might be on a disk instead of in memory, and each comparison requires a disk access.
2. The data must not only be sorted; it must also be fairly uniformly distributed. For example, a phone book is fairly uniformly distributed. If the input items are {1, 2, 4, 8, 16, …}, then the distribution is not uniform.

These assumptions are quite restrictive, so you might never use an interpolation search. But it is interesting to see that there is more than one way to solve a problem and that no algorithm, not even the classic binary search, is the best in all situations.

The idea of the interpolation search is that we are willing to spend more time to make an accurate guess regarding where the item might be. The binary search