52. In practice, this algorithm is coded differently from what we show here, requiring more comparisons but being more efficient because the data structures are simpler (and the sorting is done in place). We denote the list \(a_1, a_2, \ldots, a_n\) by \(a\), with similar notations for the other lists. Also, rather than putting \(a_1\) at the end of the first sublist, we put it between the two sublists and do not have to deal with it in either sublist.

\[
\text{procedure quick}(a_1, a_2, \ldots, a_n)
\]
\[
b := \text{the empty list}
\]
\[
c := \text{the empty list}
\]
\[
temp := a_1
\]
\[
\text{for } i := 2 \text{ to } n
\]
\[
\quad \text{if } a_i < a_1 \text{ then adjjoin } a_i \text{ to the end of list } b
\]
\[
\quad \text{else adjjoin } a_i \text{ to the end of list } c
\]
\[
\quad \{ \text{notation: } m = \text{length}(b) \text{ and } k = \text{length}(c) \}
\]
\[
\quad \text{if } m \neq 0 \text{ then quick}(b_1, b_2, \ldots, b_m)
\]
\[
\quad \text{if } k \neq 0 \text{ then quick}(c_1, c_2, \ldots, c_k)
\]
\[
\quad \{ \text{now put the sorted lists back into } a \}
\]
\[
\text{for } i := 1 \text{ to } m
\]
\[
\quad a_i := b_i
\]
\[
\text{a}_{m+1} := \text{temp}
\]
\[
\text{for } i := 1 \text{ to } k
\]
\[
\quad a_{m+i+1} := c_i
\]
\[
\{ \text{the list } a \text{ is now sorted} \}
\]

54. In the best case, the initial split will require 3 comparisons and result in sublists of length 1 and 2 still to be sorted. These require 0 and 1 comparisons, respectively, and the list has been sorted. Therefore the answer is \(3 + 0 + 1 = 4\).

**SECTION 5.5 Program Correctness**

2. There are two cases. If \(x \geq 0\) initially, then nothing is executed, so \(x \geq 0\) at the end. If \(x < 0\) initially, then \(x\) is set equal to 0, so \(x = 0\) at the end; hence again \(x \geq 0\) at the end.

4. There are three cases. If \(x < y\) initially, then min is set equal to \(x\), so \((x \leq y \land \text{min} = x)\) is true. If \(x = y\) initially, then min is set equal to \(y\) (which equals \(x\)), so again \((x \leq y \land \text{min} = x)\) is true. Finally, if \(x > y\) initially, then min is set equal to \(y\), so \((x > y \land \text{min} = y)\) is true. Hence in all cases the disjunction \((x \leq y \land \text{min} = x) \lor (x > y \land \text{min} = y)\) is true.

6. There are three cases. If \(x < 0\), then \(y\) is set equal to \(-2|x|/x = (-2)(-x)/x = 2\). If \(x > 0\), then \(y\) is set equal to \(2|x|/x = 2x/x = 2\). If \(x = 0\), then \(y\) is set equal to 2. Hence in all cases \(y = 2\) at the termination of this program.

8. We prove that Algorithm 8 in Section 5.4 is correct. It is clearly correct if \(n = 0\) or \(n = 1\), so we assume that \(n \geq 2\). Then the program terminates when the for loop terminates, so we concentrate our attention on that loop. Before the loop begins, we have \(x = 0\) and \(y = 1\). Let the loop invariant \(p\) be “\((x = f_{i-1} \land y = f_i) \lor (i \text{ is undefined} \land x = f_0 \land y = f_1)\)”. This is true at the beginning of the loop, since \(i\) is undefined and \(f_0 = 0\) and \(f_1 = 1\). What we must show now is \(p \land (1 \leq i < n)\{S\}p\). If \(p \land (1 \leq i < n)\), then \(x = f_{i-1}\) and \(y = f_i\). Hence \(z\) becomes \(f_{i+1}\) by the definition of the Fibonacci sequence. Now \(x\) becomes \(y\), namely \(f_i\), and \(y\) becomes \(z\), namely \(f_{i+1}\), and \(i\) is incremented. Hence for this new (defined) \(i\), \(x = f_{i-1}\) and \(y = f_i\), as desired. We therefore conclude that upon termination \(x = f_{i-1} \land y = f_i \land i = n\); hence \(y = f_n\), as desired.