Figure 9.1  A parallel compare-exchange operation. Processes $P_i$ and $P_j$ send their elements to each other. Process $P_i$ keeps $\min\{a_i, a_j\}$, and $P_j$ keeps $\max\{a_i, a_j\}$. 
Step 1

Step 2

Step 3

Step 4

Figure 9.2  A compare-split operation. Each process sends its block of size \( n/p \) to the other process. Each process merges the received block with its own block and retains only the appropriate half of the merged block. In this example, process \( P_i \) retains the smaller elements and process \( P_j \) retains the larger elements.
Figure 9.3 A schematic representation of comparators: (a) an increasing comparator, and (b) a decreasing comparator.
Figure 9.4 A typical sorting network. Every sorting network is made up of a series of columns, and each column contains a number of comparators connected in parallel.
<table>
<thead>
<tr>
<th>Original sequence</th>
<th>3 5 8 9 10 12 14 20 95 90 60 40 35 23 18 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Split</td>
<td>3 5 8 9 10 12 14 0</td>
</tr>
<tr>
<td>2nd Split</td>
<td>3 5 8 0</td>
</tr>
<tr>
<td>3rd Split</td>
<td>3 0 8 5</td>
</tr>
<tr>
<td>4th Split</td>
<td>0 3 5 8 9</td>
</tr>
</tbody>
</table>

**Figure 9.5** Merging a 16-element bitonic sequence through a series of log 16 bitonic splits.
Figure 9.6 A bitonic merging network for \( n = 16 \). The input wires are numbered 0, 1, \ldots, \( n - 1 \), and the binary representation of these numbers is shown. Each column of comparators is drawn separately; the entire figure represents a \( \oplus \text{BM}[16] \) bitonic merging network. The network takes a bitonic sequence and outputs it in sorted order.
Figure 9.7 A schematic representation of a network that converts an input sequence into a bitonic sequence. In this example, $\oplus \text{BM}[k]$ and $\ominus \text{BM}[k]$ denote bitonic merging networks of input size $k$ that use $\oplus$ and $\ominus$ comparators, respectively. The last merging network ($\oplus \text{BM}[16]$) sorts the input. In this example, $n = 16$. 
Figure 9.8  The comparator network that transforms an input sequence of 16 unordered numbers into a bitonic sequence. In contrast to Figure 9.6, the columns of comparators in each bitonic merging network are drawn in a single box, separated by a dashed line.
Figure 9.9  Communication during the last stage of bitonic sort. Each wire is mapped to a hyper-cube process; each connection represents a compare-exchange between processes.
Figure 9.10 Communication characteristics of bitonic sort on a hypercube. During each stage of the algorithm, processes communicate along the dimensions shown.
Figure 9.11  Different ways of mapping the input wires of the bitonic sorting network to a mesh of processes: (a) row-major mapping, (b) row-major snakelike mapping, and (c) row-major shuffled mapping.
Figure 9.12 The last stage of the bitonic sort algorithm for $n = 16$ on a mesh, using the row-major shuffled mapping. During each step, process pairs compare-exchange their elements. Arrows indicate the pairs of processes that perform compare-exchange operations.
Figure 9.13  Sorting $n = 8$ elements, using the odd-even transposition sort algorithm. During each phase, $n = 8$ elements are compared.
Figure 9.14  An example of the first phase of parallel shellsort on an eight-process array.
Figure 9.15  Example of the quicksort algorithm sorting a sequence of size $n = 8$. 
Figure 9.16  A binary tree generated by the execution of the quicksort algorithm. Each level of the tree represents a different array-partitioning iteration. If pivot selection is optimal, then the height of the tree is $\Theta(\log n)$, which is also the number of iterations.
Figure 9.17  The execution of the PRAM algorithm on the array shown in (a). The arrays leftchild and rightchild are shown in (c), (d), and (e) as the algorithm progresses. Figure (f) shows the binary tree constructed by the algorithm. Each node is labeled by the process (in square brackets), and the element is stored at that process (in curly brackets). The element is the pivot. In each node, processes with smaller elements than the pivot are grouped on the left side of the node, and those with larger elements are grouped on the right side. These two groups form the two partitions of the original array. For each partition, a pivot element is selected at random from the two groups that form the children of the node.
Figure 9.18 An example of the execution of an efficient shared-address-space quicksort algorithm.
Figure 9.19  Efficient global rearrangement of the array.
Figure 9.20 An example of the execution of sample sort on an array with 24 elements on three processes.
Figure 9.21 The execution of the hypercube formulation of quicksort for $d = 3$. The three splits—one along each communication link—are shown in (a), (b), and (c). The second column represents the partitioning of the $n$-element sequence into subcubes. The arrows between subcubes indicate the movement of larger elements. Each box is marked by the binary representation of the process labels in that subcube. A * denotes that all the binary combinations are included.
Figure 9.22  (a) An arbitrary portion of a mesh that holds part of the sequence to be sorted at some point during the execution of quicksort, and (b) a binary tree embedded into the same portion of the mesh.