```java
/** This sorts an array of 30 random integers using merge sort. Note
 * this uses the functions mergeSort, mergeSortHelper, and merge found
 * in the APCS A Course Discription.
 *
 * @author Mr. Dagler
 */

import java.lang.Math;

class MergeSort
{
   public static void main(String[] args)
   {
      final int N = 30;
      int[] vals = new int[N];

      for(int i=0; i<N; i++)
         vals[i] = (int)(1000*Math.random()+1);

      System.out.println("Here are the values before they are sorted:");
      for(int v : vals)
         System.out.print(v + " ");
      System.out.println();

      mergeSort(vals);

      System.out.println("Here are the values after they are sorted:");
      for(int v : vals)
         System.out.print(v + " ");
      System.out.println();
   }

   /** Sort an array of integers into ascending order.
    *
    * @param elements an array containing the items to be sorted.
    *
    * Postcondition: elements contains its original items and items in
    *                elements are sorted in ascending order.
    */
   public static void mergeSort(int[] elements)
   {
      int n = elements.length;
      int[] temp = new int[n];
      mergeSortHelper(elements, 0, n - 1, temp);
   }

   /** Sorts elements[from] ... elements[to] inclusive into ascending
    * order.
    *
    * @param elements an array containing the items to be sorted.
    * @param from the beginning index of the items in elements to be
    *        sorted.
    * @param to the ending index of the items in elements to be sorted.
    * @param temp a temporary array to use during the merge process.
    *
    * Precondition:
    *    (elements.length == 0 or 0 <= from <= to <= elements.length)
```

```
58          *    and elements.length == temp.length
59          * Postcondition: elements contains its original items and the items
60          *    in elements [from] ... <= elements[to] are sorted in ascending
61          *    order.
62          */
63         private static void mergeSortHelper(int[] elements, int from, int to,
64            int[] temp)
65         {
66            if (from < to)
67            {
68               int middle = (from + to) / 2;
69               mergeSortHelper(elements, from, middle, temp);
70               mergeSortHelper(elements, middle + 1, to, temp);
71               merge(elements, from, middle, to, temp);
72            }
73         }
74
75         /** Merges two adjacent array parts, each of which has been sorted
76          * into ascending order, into one array part that is sorted into
77          * ascending order.
78          *
79          * @param elements an array containing the parts to be merged.
80          * @param from the beginning index in elements of the first part.
81          * @param mid the ending index in elements of the first part.
82          *       mid+1 is the beginning index in elements of the second part.
83          * @param to the ending index in elements of the second part.
84          * @param temp a temporary array to use during the merge process.
85          *
86          * Precondition: 0 <= from <= mid <= to <= elements.length and
87          *   elements[from] ...<=  elements[mid] are sorted in ascending
88          *   order and elements[mid + 1] ... <=  elements[to] are sorted in
89          *   ascending order and elements.length == temp.length
90          * Postcondition: elements contains its original items and
91          *   elements[from] ... <= elements[to] are sorted in ascending order
92          *   and elements[0] ... elements[from-1] are in original order and
93          *   elements[to + 1] ... elements[elements.length-1] are in original
94          *   order.
95          */
96         private static void merge(int[] elements, int from, int mid, int to,
97            int[] temp)
98         {
99            int i = from;
100           int j = mid + 1;
101           int k = from;
102
103           while (i <= mid && j <= to)
104           {
105              if (elements[i] < elements[j])
106              {
107                 temp[k] = elements[i];
108                 i++;
109              }
110              else
111              {
112                 temp[k] = elements[j];
113                 j++;
114              }
```

```
115             k++;
116         }
117 
118         while (i <= mid)
119         {
120             temp[k] = elements[i];
121             i++;
122             k++;
123         }
124 
125         while (j <= to)
126         {
127             temp[k] = elements[j];
128             j++;
129             k++;
130         }
131 
132         for (k = from; k <= to; k++)
133         {
134             elements[k] = temp[k];
135         }
136     }
137 }
138 
```