Insertion, Merge, and Quick Sort Implementations in Java

```
1  ////////////////////////////////////////////////////////////////////////
2  // MergeSort - Feynman Liang (2-18-2012)
3  ////////////////////////////////////////////////////////////////////////
4  public static int[] mergeSort(int[] a) {
5      // pre: array of integers
6      // post: sorted in non-desceasing order
7
8      int len = a.length;
9
10     int blocksize = 1;
11     while (blocksize < len) {
12         // invariant: a is in sorted blocks of size
13         // blocksize/2
14         int lo = 0;
15         while (lo < (len - blocksize)) {
16             // invariant: a[0..lo-1] is sorted in blocks of
17             // size blocksize
18             int hi = lo + 2 * blocksize - 1;
19             // check to see if we've run off a[]
20             if ((len - 1) < hi)
21                 hi = len - 1;
22             inPlaceMerge(a,
23                          blocksize,
24                          lo,
25                          hi);
26             lo = lo + 2*blocksize;
27         }
28         blocksize = blocksize * 2;
29     }
30     // returns merged sorted array (or single array)
31     return a;
32 }
33
34 public static void inPlaceMerge(int[] a, int blocksize, int lo, int hi) {
35     // pre: two sorted subarrays leftn and right
36     // post: a combined array sorted in non-decreasing order
37     int[] merged = new int[hi-lo+1];
38     int i = lo, j = lo + blocksize, k = 0;
39
40     while (i <= lo + blocksize - 1 && j <= hi) {
41         // invariant: merged[] contains all keys < a[i..lo +
42         // blocksize-1] and a[j..hi] sorted in non-decreasing
43         if (a[i] < a[j]) {
44             merged[k] = a[i];
45             i++;
46         }
47         else {
48             merged[k] = a[j];
49             j++;
50         }
51         k++;
52     }
```

```
53      // After i or j runs off its half, copy other remaining half
54      while(i <= lo + blocksize - 1) {
55          merged[k] = a[i];
56          i++;
57          k++;
58      }
59      while(j <= hi) {
60          merged[k] = a[j];
61          j++;
62          k++;
63      }
64
65      // Copy merged back in to a[lo..hi]
66      for (k = 0; k < merged.length; k++)
67          a[lo+k] = merged[k];
68  }
69
70  ////////////////////////////////////////////////////////////////////////
71  // quick Sort - Feynman Liang (2-18-2012)
72  ////////////////////////////////////////////////////////////////////////
73  public static int[] quickSort(int[] a) {
74      // pre: a = array of ints
75      // post: returns a sorted in non-decreasing order
76
77      // pass a to recursive qsort method
78      qsort(a, 0, a.length-1);
79      return a;
80  }
81
82  public static void qsort(int[] a, int lo, int hi) {
83      // pivot = median in small array, median of 3 in length > 7
84      int mid = (lo+hi)/2;
85      if ((hi - lo + 1) > 7) {
86          mid = medofthree(a, lo, mid, hi);
87      }
88      int pivot = a[mid];
89
90      int i = lo, j = hi;
91      while (i <= j) {
92          // Invariant: subarray [lo..i-1] contains keys < pivot
93          // and [j+1..hi] contains keys > pivot, [i-1..j+1]
94          // contains unpartitioned elements and pivots themselves
95          // so that i and j converge around the pivot
96
97          // set i to index of leftmost key > pivot
98          while (a[i] < pivot) i++;
99          // set j to rightmost index of key < pivot
100         while (a[j] > pivot) j--;
101         // if not overlapped, swap the two indexes
102         if (i <= j) {
103             swap(a, i++, j--);
104         }
105     }
106
```

```
107     // Recursively sort sub-partitions
108     if (lo < i-1) qsort(a, lo, i-1);
109     if (i < hi) qsort(a, i, hi);
110 }
111
112 public static void swap(int[] a, int i, int j) {
113     // swaps key at index i with key at index j in array a
114     int temp = a[i];
115     a[i] = a[j];
116     a[j] = temp;
117 }
118
119 public static int medofthree(int[] a, int lo, int mid, int hi) {
120     // finds the median of lo, mid, and hi
121     if (a[lo] < a[mid]) {
122         if (a[mid] < a[hi]) return mid;
123         else {
124             if (a[lo] < a[hi]) return hi;
125             else return lo;
126         }
127     }
128     else {
129         if (a[mid] > a[hi]) return mid;
130         else {
131             if (a[lo] > a[hi]) return hi;
132             else return lo;
133         }
134     }
135 }
```