```java
/**
 * COMP215-Programming Project 3: 01 Knapsack Problem Analysis.
 * QUICKSORT is the Java implementation of the quick sort algorithm.
 * This implementation is based on psuedocode from "CLRS Algorithms."
 * This class is timed using the STOPWATCH class.
 * @author Andrew Parsons
 * @version 27 February 2017
 */
class QuickSort implements Sorter {

    /* --- INSTANCE VARIABLES --- */
    private Stopwatch stopwatch;
    private long elapsedTime;
    private static boolean debug = KnapsackMain.debug;

    @Override
    public Comparable[] sort(Comparable[] dataset, int indexBegin, int indexEnd) {

        stopwatch = new Stopwatch();

        while (indexBegin < indexEnd) {

            int pivot = partition(dataset, indexBegin, indexEnd);

            if (pivot-indexBegin < indexEnd-pivot) {
                sort(dataset, indexBegin,pivot-1);
                indexBegin = pivot+1;
            } else {
                sort(dataset, pivot+1, indexEnd);
                indexEnd = pivot-1;
            }
        }
        elapsedTime = stopwatch.elapsedTime();
        return dataset;
    }

    private int partition(Comparable[] dataset, int indexBegin, int indexEnd) {

        Comparable temp;
        Comparable pivot = dataset[indexEnd];
        int i = indexBegin - 1;
        int j = indexBegin;
        if (debug) {
            assert assertionInvariant1(dataset, indexBegin, i, pivot);
            assert assertionInvariant2(dataset, i, j, pivot);
        }

        /*INIT: subarrays p...i and i+1...j are empty, indexEnd is pivot. */

        for (j = indexBegin; j <= indexEnd-1; j++) {

            if (debug) {
                assert assertionInvariant1(dataset, indexBegin, i, pivot);
                assert assertionInvariant2(dataset, i, j, pivot);
            }
            /*MAIN: if dataset[j] <= pivot, then dataset[j] and dataset[i+1] are swapped and
            then i and j are incremented.
            If dataset[j] > pivotm then only j is incremented. */

            if (dataset[j].compareTo(pivot) <= 0) {
                i++;
                temp = dataset[i];
                dataset[i] = dataset[j];
                dataset[j] = temp;
```

```
            }
        } if (debug) {
            assert assertionInvariant3(dataset, j, pivot);
            /* j = indexEnd, so all data is partitioned. */
        }

        temp = dataset[i+1];
        dataset[i+1] = dataset[indexEnd];
        dataset[indexEnd] = temp;
        return i+1;
    }

    @Override
    public long getElapsedTime() {return elapsedTime;}

    /** check if subarray is sorted in ascending order (called by assert) */
    private static boolean assertionInvariant1(Comparable[] dataset, int start, int i,
    Comparable end) {
        for (int j = start; j <= i; j++) {
            if (dataset[j].compareTo(end) < 1) {
                return true;
            }
        } return false;
    }

    private static boolean assertionInvariant2(Comparable[] dataset, int i, int j, Comparable
    end) {
        for (int k = i+1; k < j; k++) {
            if (dataset[k].compareTo(end) > 0) {
                return true;
            }
        } return false;
    }

    private static boolean assertionInvariant3(Comparable[] dataset, int j, Comparable end) {
        if (dataset[j].compareTo(end) < 1) {
            return true;
        } return false;
    }

    @Override // DO NOT USE
    public Comparable[] sort(Comparable[] dataset) {
        return new Comparable[0];
    }
}
```