

```

1  package edu.neu.coe.info6205.sort.linearithmic;
2
3  import edu.neu.coe.info6205.sort.Helper;
4  import edu.neu.coe.info6205.sort.InstrumentedHelper
5  ;
6  import edu.neu.coe.info6205.util.Benchmark;
7  import edu.neu.coe.info6205.util.Benchmark_Timer;
8  import edu.neu.coe.info6205.util.Config;
9
10 import java.util.ArrayList;
11 import java.util.List;
12
13 public class QuickSort_DualPivot<X extends
14     Comparable<X>> extends QuickSort<X> {
15
16     public static final String DESCRIPTION = "
17         QuickSort dual pivot";
18
19     public QuickSort_DualPivot(String description,
20         int N, Config config) {
21         super(description, N, config);
22         setPartitioner(createPartitioner());
23     }
24
25     /**
26      * Constructor for QuickSort_3way
27      *
28      * @param helper an explicit instance of Helper
29      * to be used.
30      */
31     public QuickSort_DualPivot(Helper<X> helper) {
32         super(helper);
33         setPartitioner(createPartitioner());
34     }
35
36     /**
37      * Constructor for QuickSort_3way
38      *
39      * @param N the number elements we expect
40      * to sort.
41      * @param config the configuration.

```

```

36     */
37     public QuickSort_DualPivot(int N, Config config
38 ) {
39         this(DESCRIPTION, N, config);
40     }
41     @Override
42     public Partitioner<X> createPartitioner() {
43         return new Partitioner_DualPivot(getHelper
44 ());
45     }
46     public class Partitioner_DualPivot implements
47 Partitioner<X> {
48         public Partitioner_DualPivot(Helper<X>
49 helper) {
50             this.helper = helper;
51         }
52         /**
53          * Method to partition the given partition
54          into smaller partitions.
55          *
56          * @param partition the partition to divide
57          up.
58          * @return an array of partitions, whose
59          length depends on the sorting method being used.
60          */
61         public List<Partition<X>> partition(
62 Partition<X> partition) {
63             final X[] xs = partition.xs;
64             final int lo = partition.from;
65             final int hi = partition.to - 1;
66             helper.swapConditional(xs, lo, hi);
67             int lt = lo + 1;
68             int gt = hi - 1;
69             int i = lt;
70             X v1 = xs[lo];
71             X v2 = xs[hi];
72             // NOTE: we are trying to avoid

```

```

68 checking on instrumented for every time in the
   inner loop for performance reasons (probably a
   silly idea).
69 // NOTE: if we were using Scala, it
   would be easy to set up a comparer function and a
   swapper function. With java, it's possible but
   much messier.
70         if (helper.instrumented()) {
71             helper.incrementHits(2); // XXX
   these account for v1 and v2.
72             while (i <= gt) {
73                 if (helper.compare(xs, i, v1
74 ) < 0) helper.swap(xs, lt++, i++);
75                 else if (helper.compare(xs, i
76 , v2) > 0) helper.swap(xs, i, gt--);
77                 else i++;
78             }
79             helper.swap(xs, lo, --lt);
80             helper.swap(xs, hi, ++gt);
81         } else {
82             while (i <= gt) {
83                 X x = xs[i];
84                 if (x.compareTo(v1) < 0) swap(
85 xs, lt++, i++);
86                 else if (x.compareTo(v2) > 0)
87 swap(xs, i, gt--);
88                 else i++;
89             }
90             swap(xs, lo, --lt);
91             swap(xs, hi, ++gt);
92         }
93
94         List<Partition<X>> partitions = new
   ArrayList<>();
95         partitions.add(new Partition<>(xs, lo
96 , lt));
97         partitions.add(new Partition<>(xs, lt
98 + 1, gt));
99         partitions.add(new Partition<>(xs, gt
100 + 1, hi + 1));
101         return partitions;

```

```

95         }
96
97         // CONSIDER invoke swap in BaseHelper.
98         private void swap(X[] ys, int i, int j) {
99             X temp = ys[i];
100             ys[i] = ys[j];
101             ys[j] = temp;
102         }
103
104         private final Helper<X> helper;
105     }
106
107     public static void main (String[] args) {
108         int N = 10000;
109
110         for(int i=2;i<12;i++) {
111
112             InstrumentedHelper<Integer>
instrumentedHelper = new InstrumentedHelper<>("
QuickSort_DualPivot", Config.setupConfig("true", "
0", "0", "", ""));
113             QuickSort_DualPivot<Integer> s = new
QuickSort_DualPivot<>(instrumentedHelper);
114             int j = N * i / 2;
115             s.init(j);
116
117             Integer[] temp = instrumentedHelper.
random(Integer.class, r -> r.nextInt(j));
118
119             Partitioner<Integer> partitioner = s.
createPartitioner();
120             List<Partition<Integer>> partitionList
= partitioner.partition(new Partition<>(temp, 0,
temp.length));
121             Partition<Integer> p1 = partitionList.
get(0);
122             Partition<Integer> p2 = partitionList.
get(1);
123             Partition<Integer> p3 = partitionList.
get(2);
124

```

```
125         Benchmark<Boolean> benchmark1 = new
        Benchmark_Timer<>("Sorting with", b -> s.sort(temp
        , 0, p1.to, 0));
126         double b1 = benchmark1.run(true, 20);
127         Benchmark<Boolean> benchmark2 = new
        Benchmark_Timer<>("Sorting with", b -> s.sort(temp
        , p2.from, p2.to, 0));
128         double b2 = benchmark2.run(true, 20);
129         Benchmark<Boolean> benchmark3 = new
        Benchmark_Timer<>("Sorting with", b -> s.sort(temp
        , p3.from, j, 0));
130         double b3 = benchmark3.run(true, 20);
131
132         long nCompares = instrumentedHelper.
        getCompares();
133         int nSwaps = instrumentedHelper.
        getSwaps();
134         int nHits = instrumentedHelper.getHits
        ();
135
136         double nTime = (b1 + b2 + b3);
137
138         System.out.println("When array size is
        : " + j);
139         System.out.println("Compares: " +
        nCompares);
140         System.out.println("Swaps: " + nSwaps
        );
141         System.out.println("hits: " + nHits);
142         System.out.println("Time: " + nTime);
143
144         System.out.println("\n\n");
145     }
146 }
147
148 }
149
150
```