Group 9

Ben Sixel and Jack Ziegler

The Details

- **Place:** Second in the class, Third overall
- **Sorting Method:** Timsort on Custom Objects
- **Running Time:** Θ(nlogn), because Mergesort
- **Additional Memory:** N (For our custom objects)
- **Correctness:** Correct

Sorting Approach

```
58          private static int[][] sort(int[][] toSort) {
59                  //Puddle.setThreshold(threshold);
60
61                  // Puddles are just a codename, for fun. Each represents a single point on the "board."
62                  Puddle[] points = new Puddle[toSort.length];
63
64                  int[] p; // Storing the values seems to improve speed. Not sure why, but we got on an abstraction kick and it worked.
65                  for (int i = 0; i < toSort.length; i++) { // Populate all the Puddles.
66                          p = toSort[i];
67                          points[i] = new Puddle(p[0], p[1], p[2]);
68                  }
69
70                  Arrays.sort(points);
71
72                  // Copy the newly sorted values back into toSort.
73                  Puddle pud;
74                  for (int i = 0; i < toSort.length; i++) {
75                          pud = points[i];
76                          toSort[i][0] = pud.x;
77                          toSort[i][1] = pud.y;
78                          toSort[i][2] = pud.n;
79                  }
80
81
82                  return toSort;
```

Puddles

```
175         private static class Puddle implements Comparable<Puddle> {
176                 private int x;
177                 private int y;
178                 private int n;
179                 private double dist;
180
192                 public Puddle(int xco, int yco, int pos) {
193                         x = xco;
194                         y = yco;
195                         n = pos;
196                         //dist = distance(point);
197                         dist = distance(x, y);
198                 }
199
200                 public int compareTo(Puddle that) {
201                         double diff = this.dist - that.dist;
202
203                         if (Math.abs(diff) < (threshold * threshold)) {
204                                 return this.n - that.n;
205                         }
206
207                         //return (diff > 0) ? 1 : -1;
208                         if (diff > 0) {
209                                 return 1;
210                         } else {
211                                 return -1;
212                         }
213
214                 }
```

Distance

```
147         private static double distance(int xco, int yco) {
148         // distance to the first point (set to 0 if too small):
149         double deltaX = x1 - xco;
150         double deltaY = y1 - yco;
151         // beware of integer overflow! d1 has to be a double to accommodate large numbers
152         double d1 = deltaX * deltaX + deltaY * deltaY;
153         if (d1 < (threshold * threshold)) {
154                 d1 = 0.0;
155         }
156
157         // distance to the second point (set to 0 if too small):
158         deltaX = x2 - xco;
159         deltaY = y2 - yco;
160         double d2 = deltaX * deltaX + deltaY * deltaY;
161         if (d2 < (threshold * threshold)) {
162                 d2 = 0.0;
163         }
164
165         // the smaller of the two:
166         if (d1 <= d2)
167                 return d1;
168         return d2;
169 }
```