**Henry Chang Lab 8**

I commented out the codes for Drawing in ConvexHull.java and kept only the part that actually requires d-heap to organize the randomly generated 2d Points. For each d of 2,3,4,8, I run five trials to see how changes in d would affect the application of d-heaps on Convex Hull calculations. The results are for calculating the ConvexHull for 1000 2D points (unit in seconds):

| d | 2 | 3 | 4 | 8 |
|---|---|---|---|---|
| | 0.355 | 0.342 | 0.33 | 0.324 |
| | 0.402 | 0.345 | 0.323 | 0.388 |
| | 0.419 | 0.343 | 0.427 | 0.381 |
| | 0.378 | 0.384 | 0.372 | 0.473 |
| | 0.371 | 0.369 | 0.337 | 0.42 |
| min | 0.355 | 0.342 | 0.33 | 0.324 |
| max | 0.419 | 0.384 | 0.427 | 0.473 |
| mean | 0.385 | 0.357 | 0.358 | 0.397 |
| standard deviation | 0.025 | 0.018 | 0.043 | 0.054 |

Theoretically, for higher d, the insertion should take less time because for the same size of input the heap will have lesser levels, and swim() would involve less steps. However, higher d would create higher runtime for the delMin() function since the comparing of a parent with its children would take more time during sink().

From the results, we see that the runtime performance of each d is similar. The only subtle trend is that higher d seems more unstable to runtime. It seems that ConvexHull.java, using both delMin() and insert(), balances out the effect of d. However, since the number of trials is small, this result could only give us a vague idea of how d would affect the performance of d-heaps on getting a Convex Hull. More trials, consideration of larger range of d's and closer analysis need to be done to generate more reliable results.

Note: While I tried to only keep the parts that used only d-heap methods such as insert() and delMin(), some other methods tied to them are also included as part of the runtime consideration, so the results may be slightly affected by these other functions, and may not exactly capture the effect of the size of d on getting convex hulls.