Delaunay Triangulation Project: Divide-and-Conquer Algorithm

Instructions: After extracting the contents of the tar file, compile the code by running `make` in the extraction directory. This should produce an executable named `delaunay`. Command-line useage of the program goes as follows:

```
./delaunay <path to .node file> <-d (for divide and conquer) or -a (for alternating cut)>
```

For example, to triangulate `spiral.node` data file with the *traditional divide-and-conquer* algorithm, run:

```
./delaunay spiral.node -d
```

To triangulate the same file with the *alternating-cut* divide and conquer, run:

```
./delaunay spiral.node -a
```

The output `.ele` file will be saved to the same location as the data file and with the same name.

Timing Comparisons:
Table 1 containing timings for the original Gubias and Stolfi Divide & Conquer triangulation algorithm compared to the alternating-cut modification for data sets of 10,000, 100,000, and 1,000,000 points.

	Divide & Conquer	**Alt-Cut Divide & Conquer**
ttimeu10000 (sec)	0.952599	0.457524
ttimeu100000 (sec)	12.4439	4.75364
ttimeu1000000 (sec)	144.5	45.2758

Table 1: Timing comparisons of divide and conquer and alternating cut on data sets of 10,000, 100,000, and 1,000,000 points.

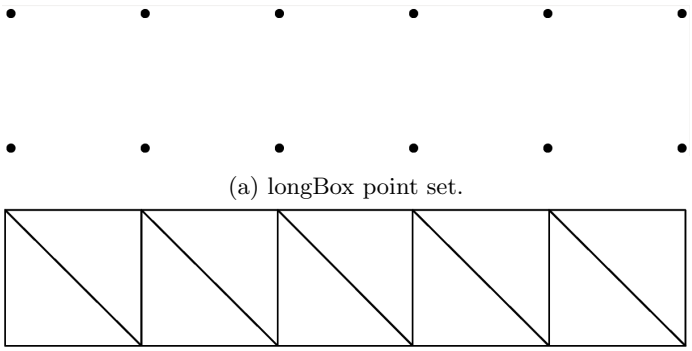
The fastest algorithm is clearly the alternating-cut modification of the divide and conquer algorithm. A reason for this speedup comes from the way in which the algorithm recurses on the data points. For a random point set, partitioning the set always vertically in half based on x-coordinate creates a slightly larger overhead when merging two convex hulls together (due to the number of cross edges that the vertical cut will intersect). When partitioning the set in alternating vertical and horizontal cuts with partial sorting at each step, the number of cross edges that intersect with the vertical or horizontal cut are smaller, therefore giving for a faster run-time due to smaller merges.

Divide-and-Conquer Analysis: To create a point set for which the vertical cuts algorithm is notably faster than the alternating cuts algorithm, we consider the case where only performing vertical cuts on the point set will lead to smaller merges compared to vertical and horizontal cuts. A simple example of such a point set is seen in Figure 1a and it's triangulation is seen in Figure 1b. Empirically, the timings of the triangulation show that the original vertical-cut only divide and conquer algorithm is faster than the alternating cut modification, even for such a small point set (see Table 2). This discrepancy can be accounted for by the number of cross-edges that the cuts bisect.

If we consider the vertical-cut algorithm, the deepest recursion constructs four triangles, and merges pairs of triangles into two rectangles by constructing two connecting triangles. These two rectangles are connected by constructing two final triangles. However, in the alternating-cut algorithm, the deepest recursion (cutting on x-coordinate) considers three collinear points that form a single line. After this recursion completes, the recursion that cut based on y-coordinate now has to merge together two straight lines. This requires the

	Divide & Conquer	**Alt-Cut Divide & Conquer**
longBox (sec)	0.000763	0.000977

Table 2: Timing comparison for longBox dataset shows that the alternating cut performs worse than traditional divide and conquer.



(a) longBox point set.

(b) Triangulation of longBox point set.

Figure 1: Point set in (a) is designed to perform better with traditional divide and conquer compared to alt-cut divide and conquer.

construction of four triangles, double that of what needed to happen in the case of the vertical-only, leading to a slowdown in overall performance for this point set.

References: The divide-and conquer algorithm as well as quad-edge data structure is credited to [1]. Code for robust predicates for floating-point inputs is from [3], and I utilized the ShowMe display program to visualize triangulations from [2]. I also called standard functions from the C++ algorithm library for lexicographic sorting and for the merge/partitioning algorithm.

[1] Leonidas J. Guibas and Jorge Stolfi, Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams, ACM Transactions on Graphics 4(2):74123, April 1985.

[2] Jonathan Richard Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, in "Applied Computational Geometry: Towards Geometric Engineering" (Ming C. Lin and Dinesh Manocha, editors), volume 1148 of Lecture Notes in Computer Science, pages 203-222, Springer-Verlag, Berlin, May 1996.

[3] Jonathan Richard Shewchuk, Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates, Discrete Computational Geometry 18:305-363, 1997. Also Technical Report CMU-CS-96-140, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1996.