Brian Pinson
Project 1
COT4521.001
09/22/2017

Convex Hull Project

Input: .txt file containing number of coordinates followed by coordinates
Output: coordinates of the convex hull
1. **Algorithm** Convex Hull
2. Build 2-d vector from input file
3. Organize 2-d vector numerically from the x-axis
4. DivideAndConquer(input-vector);
5. printConvexHull(output);
end;

Input: input: 2-d vector containing coordinates
Output: output: 2-d vector coordinates of the convex hull
1. **Algorithm** DivideAndConquer(input)
2. if(input.size() < 4)
3. return input; // convex hull has already been solved.
4. int size = input.size/2;
5. leftHalf = DivideAndConquer(input(begin(), size);
6. rightHalf = DivideAndConquer(input(size, end());
7. Merger(leftHalf, rightHalf);
end;

Input: leftHalf, rightHalf
Output: merged
1. **Algorithm** Merger(leftHalf, rightHalf)
2. vector merged.size(leftHalf.size() + rightHalf.size();
3. merged.insert(leftHalf);
4. merged.insert(rightHalf);
5. int i = 0; // start on the left's most right point
6. int j = 0; // start on the right's most left point
7. while(!lowerTangent && !upperTangent)
8 — bool above = false; bool below = false;
9. — for(k = 0; k < merged.size(); k++)
10. ———— if (left coordinate and right coordinate form line above all other coordinates)
11. ———— above = true
12. ———— else if (left coordinate and right coordinate form line below all other coordinates)
13. ———— below = true
13. Calculate area of quadrilateral formed by the upper and lower tangent lines
14. for(k = 0; k < merged.size(); k++)
15. — Calculate quadrilateral formed by upper and lower tangent lines using point k as center
16. — if(quadrilateral area = second quadrilateral)
17. ———— delete point k from merged.
18. Return merged;

Example Case: square.txt

A simple square with 100 coordinates (0, 0), (0, 1), (0, 2)… (9, 9). It proves it can handle coordinates that are on the upper and lower tangent lines avoiding incorrectly identifying the point as one of the ends of the tangent line. For example the upper tangent is ((0,9),(9,9)) and the point (5,9) will be ignored even though it forms a tangent with all points that share the same y value of 9. It also proves it can handle sets of coordinates that do not form polygons such as (1, 0), (2, 0), (3, 0) which can occur during the divide and conquer process.

Example Case: triangle.txt

A triangle with the coordinates (0, 10), (-5, 0), (5, 0) repeated 10 times. The program will very quickly identify and delete all duplicate coordinates before it wastes time calculating for the upper and lower tangents. It does not matter what order the duplicate coordinates are put into the input file.

Example Case extreme.txt

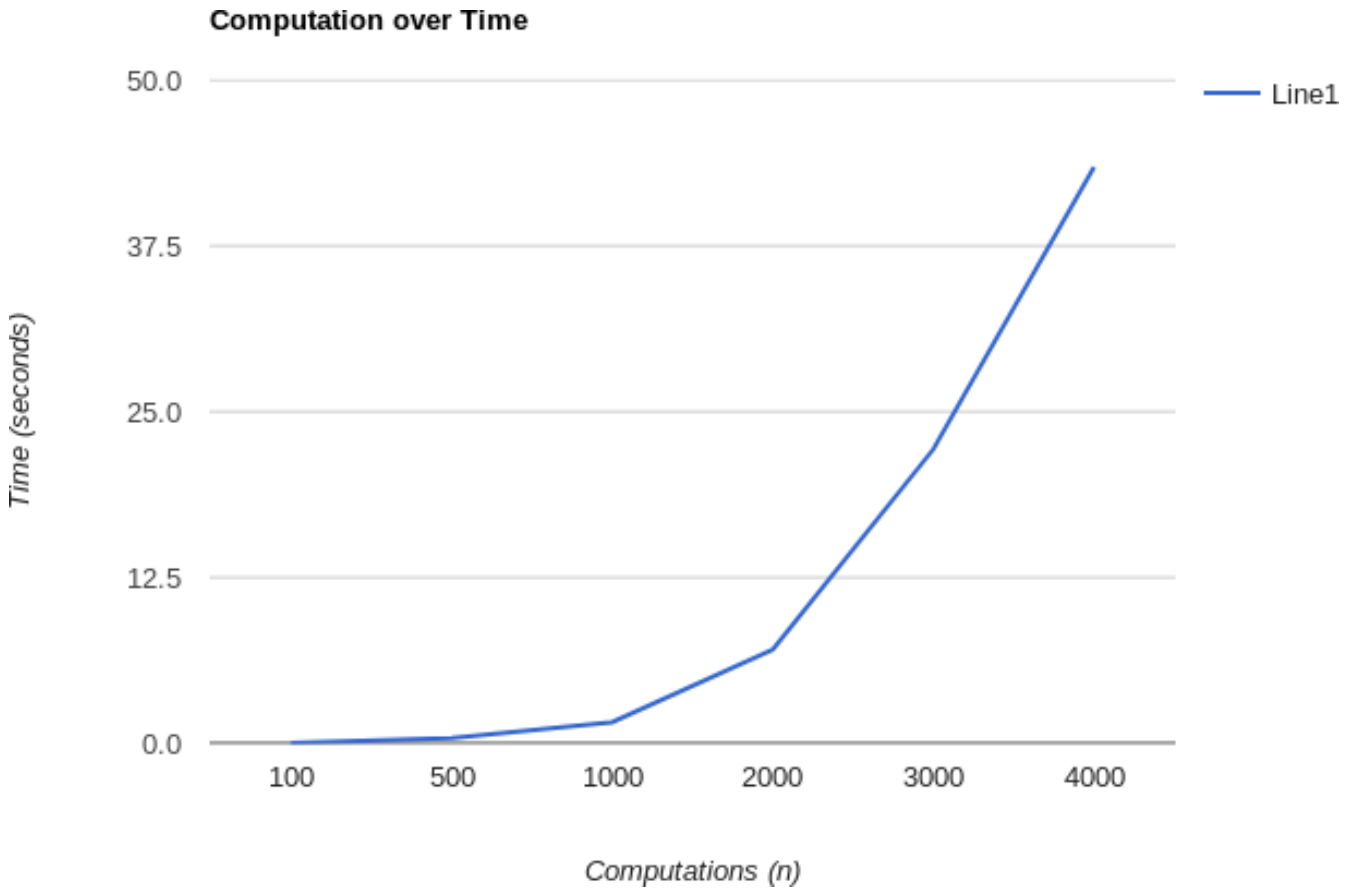
A quadrilateral with 10 initial coordinates (10000, 10000), (0, 1), (3, 3000), (4, 4000), (-10, -10), (-10000, -1000), (444, 999), (22, 34), (56, 77), and (-11, -11). The example is to test how the program handles coordinates separated by extreme lengths. The end result is a convex hull forming a highly skewed quadrilateral with four points: (-10000, -1000), (-11, -11), (23, 34), and (10000, 10000).

Example Case parallel.txt

A parallelogram with 20 initial coordinates: (0, 3), (0, 4), (0, 7), (0, 1), (0, 2), (0, 6), (0, 5), (0, 9), (0, 4), (0, 2), (10, 12), (10, 3), (10, 4), (10, 5), (10, 1), (10, 2), (10, 6), (10, 7), (10, 8), (10, 9). This was designed to have all coordinates along either the upper or lower tangent line. Regardless of how many there are the program is capable of identifying the furthest coordinate on either end of the line and deleting the rest.

Example Case: blob.txt

Taken from a random number generator the coordinates range from 10 to 10000. 10 can be found in “blob.txt” (66, 68), (37, 1), (71, 64), (46, 59), (4, 28), (64, 5) (1, 75), (30, 26), (45, 17) (79, 76). The original 10,000 coordinates can be found in “10000.txt”.



Growth rate is steady increasing doubling in computation time for every increase of a 1,000 new coordinates. Appears to be still below $O(n^2)$ time and significantly below $O(n^3)$ time that would normally come from computing using brute force methods.