Leonardo Azopardo

Report

<u>General Information:</u>
IO:
All the programs read the file with the points in the same way, and store the points the same way: They go through every double and store it in a Point2DWithIndex array.
Point2DWithIndex:
The subclass Point2DWithIndex was created so every point has its original index as a member, when it is stored. This enables us to find the smallest index of each point in O(1), instead of going through the initial Point2DWithIndex array.

<u>Slow Convex Hull:</u>
I implemented two versions of the Slow Convex Hull, SlowConvexHull and SlowConvexHull2.

SlowConvexHull
This program goes through every point in the point set, and after choosing two points, it checks if every other point in the array
is two the left of it. Since it has to go through every point in
the array 3 times, in a triple for loop, it has a worst-case
complexity of \textasciitilde N^3. A hashset structure is
used to store the list of points, since it provides non repeating elements while adding of O(1).

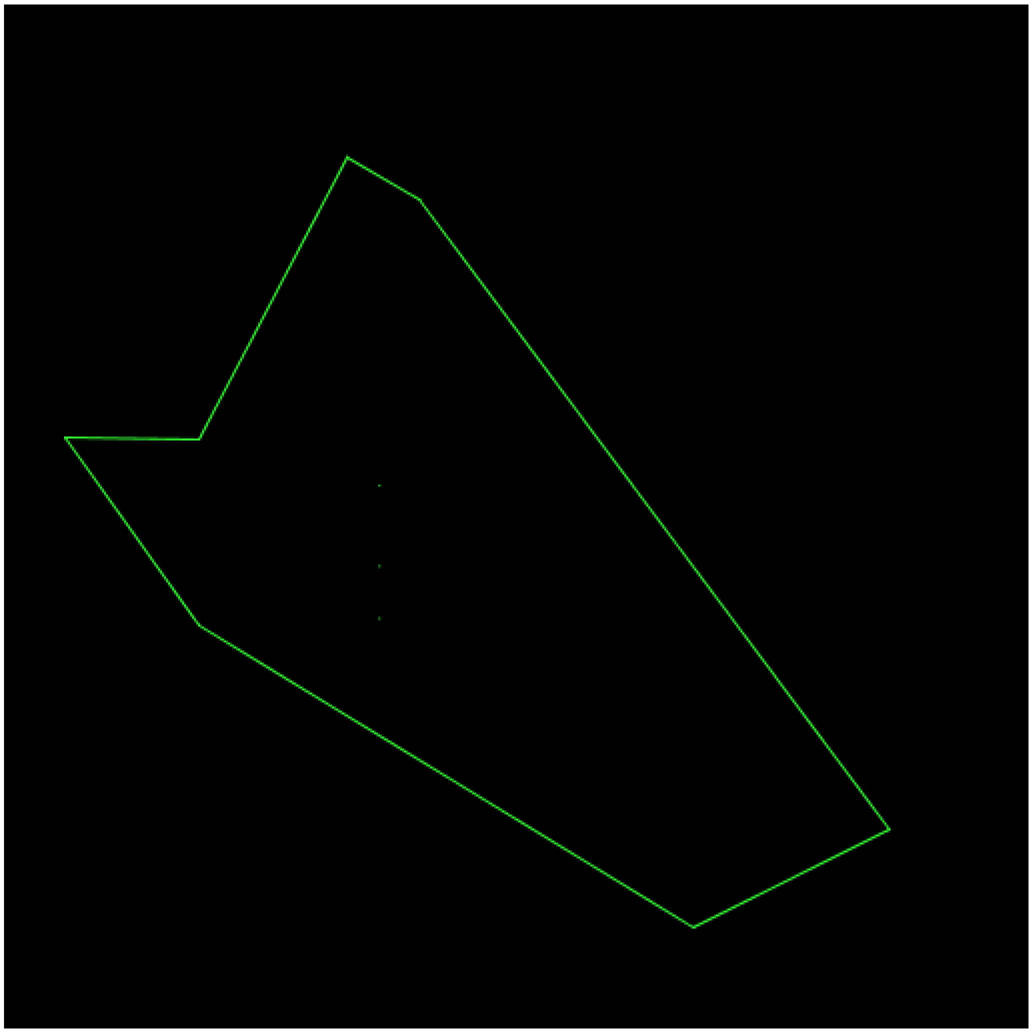
SlowConvexHull2:
This program uses the same loop as the program above but instead
of using individual points, it uses an array of edges to store pair of points that are edges of the convex hull, the worst-case is also \textasciitilde N^3. To add an edge, to the array of edges, it checks every edge to see if it is already in, this has a worst-case complexity of O(n).

<u>Fast Convex Hull:</u>

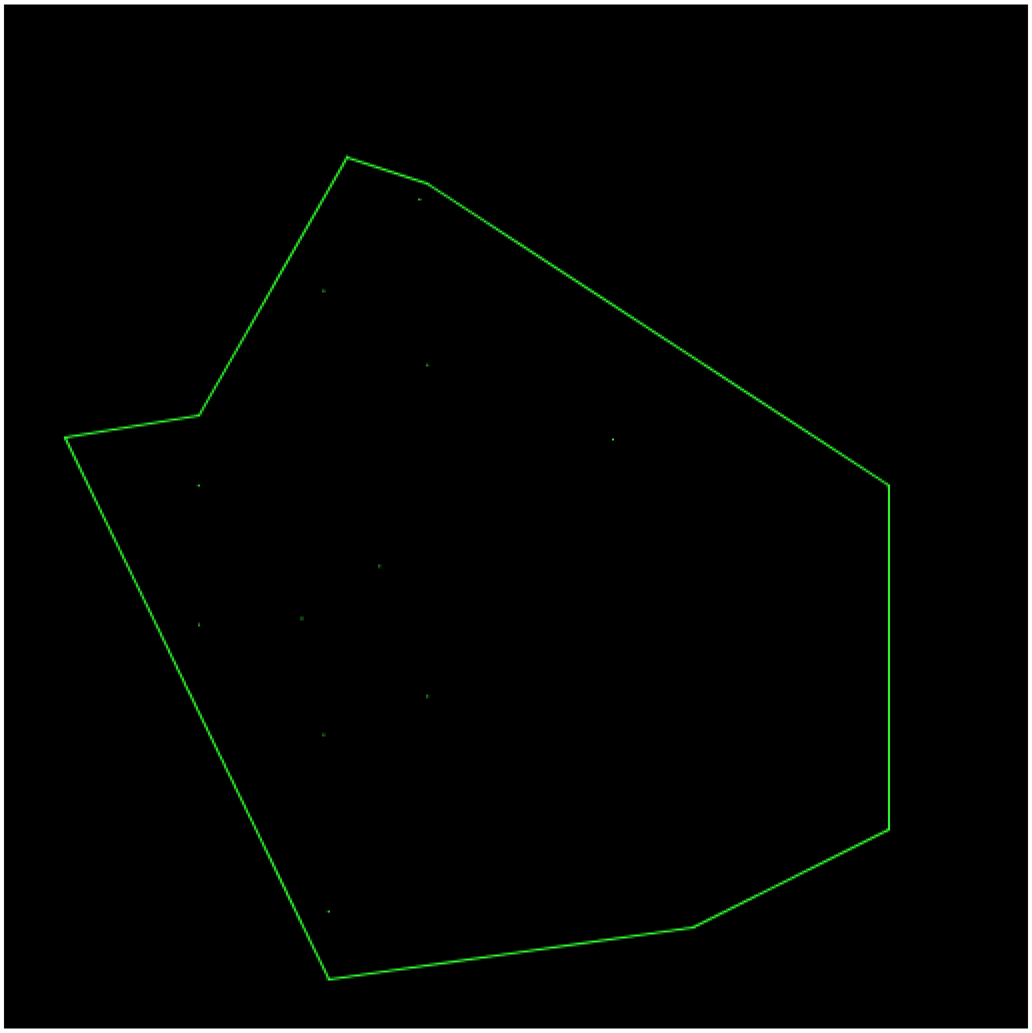
This program uses the algorithm provided in the handout to find the hull,
to make the representation more explicit the program uses an arraylist of size 2, to hold two arraylists: the lower and the upper list. After populating this two arrays, the merging happens in O(1) as it is just adding to the array of arrays. Populating the lists has a worst-case complexity of \textasciitilde nlogn, because of the divide and conquer approach used. To output the hull, it is necessary to find the smallest index(O(n)), and putting the points of the hull in an array so it can be iterated easily(O(n)).

Visualizations:

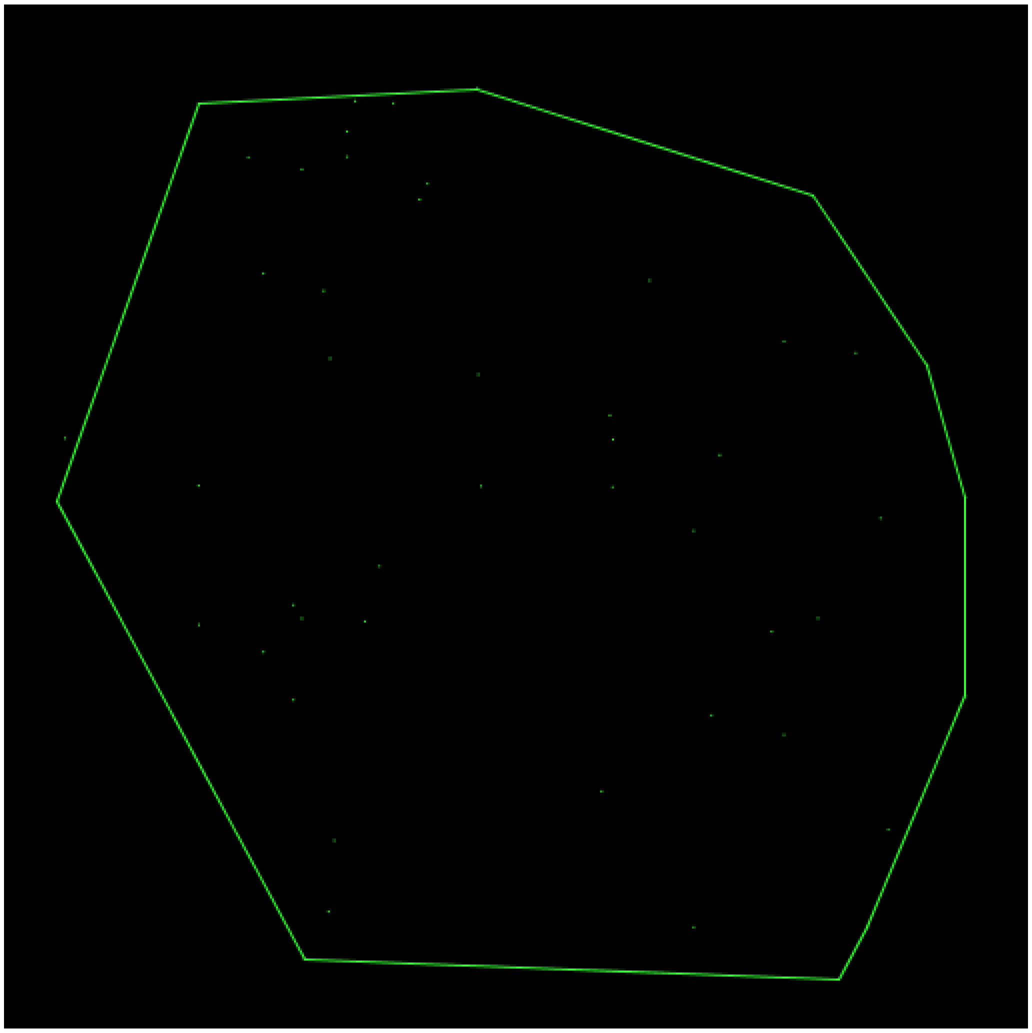
10.txt



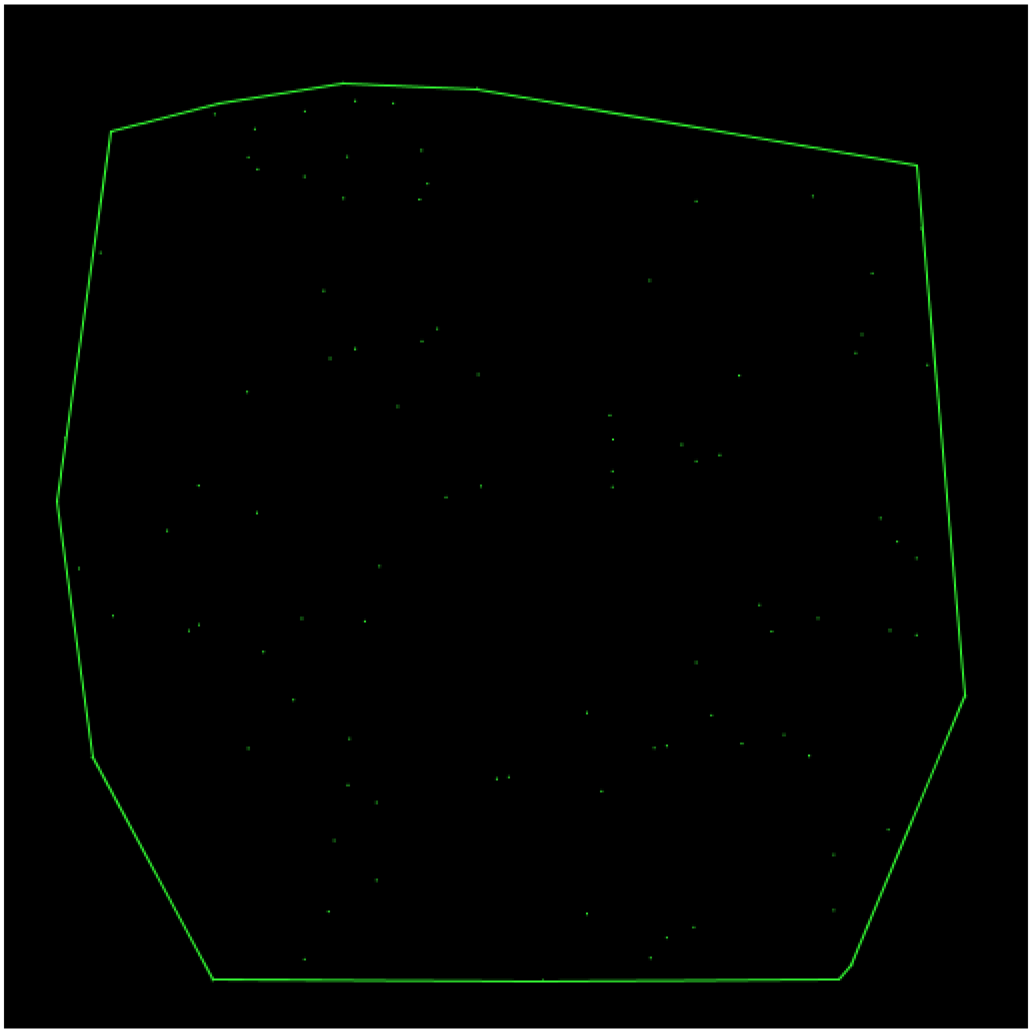
20.txt



50.txt



100.txt



500.txt