**Quickhull Algorithm Summary**

**Problem to be solved**

The Quickhull algorithm solves the problem of finding a minimum set of points from a data set of points on a cartesian plane that enclose the whole set in a box in the cartesian plane. The box must be enclosed in a way that no point in the box is able to reach another point in the box by going in a straight line and going outside of the box’s boundary lines.

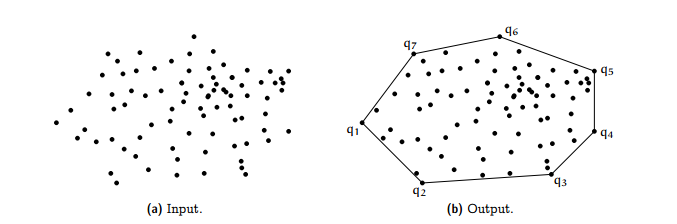


Figure - initial data set and then transformation to convex set

**Process of finding a solution**

The Quickhull algorithm uses the power of geometry and divide and conquer algorithms. The following is a pictorial overview for how quickhull finds the convex set.

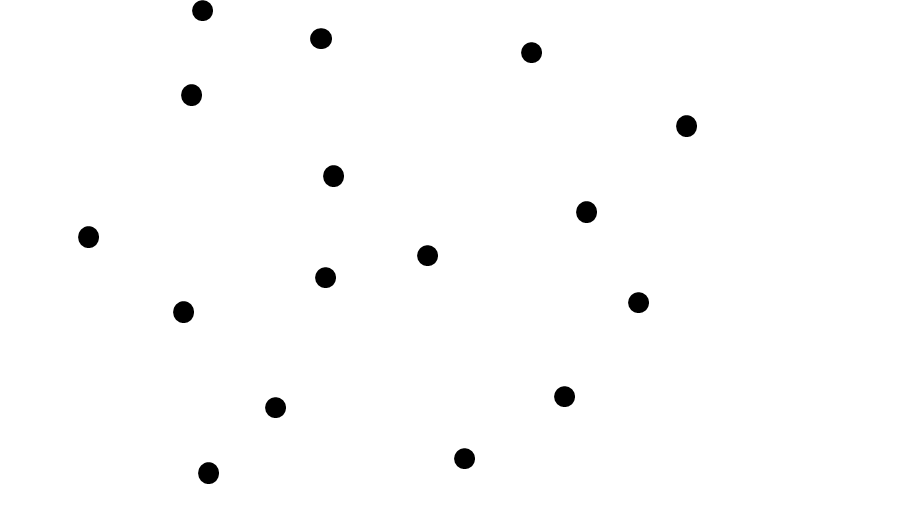


Figure - Initial data set

Step 1) – Find the point with the minimum x value and the point with the max x value and draw a line through them

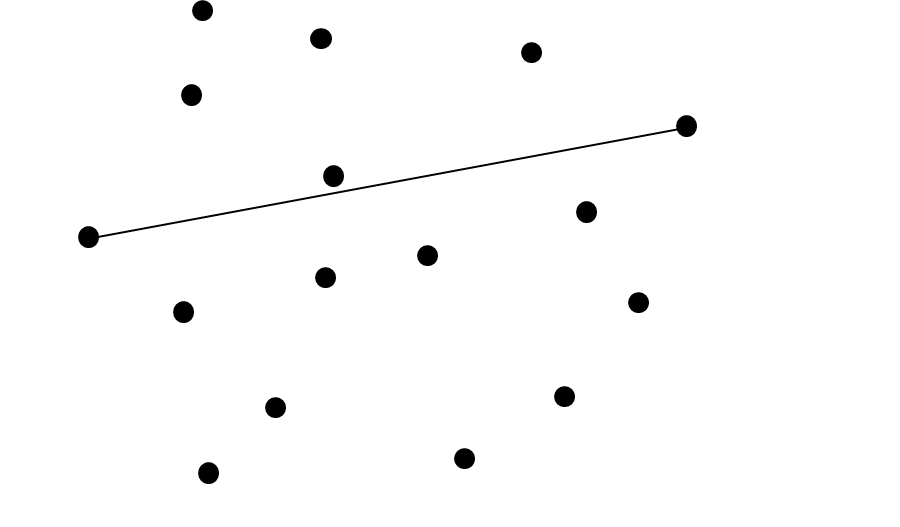


Figure - line separating upper and lower hull sets

Step 2) – separate the points in to two sets, those above the line and those below the line, we will focus on just building the hull with the lower set, but the process is identical for the lower set

Step 3) – find the point furthest away from the initial line by applying:

**X1Y2 + X3Y1 + X2Y3 – X3Y2 – X2Y1 - X1Y3**

And then whichever point has the largest negative value, that will be the point furthest away in the lower hull and part of your convex set



Figure - Point furthest away from lower hull set line

Step 4) – Now draw a line from the added convex setpoint and the already established convex setpoints of X1 and X2 and remove all points that are to the left of those lines and create a left set and a right set and repeat the above sequence of finding a max point away from the new line and removing the inner points and processing the new left and right set until there are no more points to process



Figure - removal of inner points and new left and right set for processing

**Pseudocode Implementation**

-Find the lowest X and highest X points and add them to the convex set

-Divide the points between two sets of those that fall on the left of the line between those points that fall to the right of the line by application of the analytical geometry equation

**If X1Y2 + X3Y1 + X2Y3 – X3Y2 – X2Y1 - X1Y3 > 0, then point is to the left of line**

(Right Set processing)

-Pass in to a recursive function the lowest and X and highest X points from stage 1, and through the analytical geometry equation find the point furthest away from the line

**Min( X1Y2 + X3Y1 + X2Y3 – X3Y2 – X2Y1 - X1Y3) = point farthest away from left of line**

-Add that point(Xnew) to the convex set

-sort the data points from those to the right and left of the line that the Xlowest and Xnew make and throw away all the points to the left and the new set will be the left set

-sort the data points from those to the right and left of the line that the Xnew and Xhighest make and throw away all the points to the left and the new set with be the right set

-Repeat the above steps for right set processing with the Xlowest and Xnew and the left set

-Repeat the above steps for right set processing with the Xnew and Xhighest and the left set

-Continue the above set processing until the set to process has only 2 points

(Left Set Processing)

-This will be done in the same manner as the right set processing except only the points to the left of the lines will be included and those to the right will be thrown out of further processing. The equation for which point is farthest away from the line is:

**Max( X1Y2 + X3Y1 + X2Y3 – X3Y2 – X2Y1 - X1Y3) = point farthest away from left of line**

-Return the convex set

**Technical Problems in algorithm design**

The algorithm is already well established, so the hurdles in programming it were more to do with working in python for the first time and taking a very general description from the textbook and turning that in to actual code. The major bugs and challenges that I had to overcome were this:

* Understanding how python uses, initializes, adds items, and iterates through lists
* Figuring out a common data structure that could be used. We settled on a point class design that could be passed in and returned through lists to hold our data sets of cartesian points
* Understanding what would be a proper base case condition that should stop any further recursive calls and pass back the required points as convex points
* How to get the correct initial points at the beginning with an unsorted data set
* Weeding out a bug, where I ended up making the wrong recursive call in the lower hull algorithm

**Mathematical Analysis of Algorithm**

The Quickhull is a divide and conquer algorithm, so the Master Theorem will be applied to analysis of it

**T(n) = A \* T(n/B) + nD**

Since The algorithm takes the current set and divides it in to two other sets for processing, A = 2. Since the algorithm has to determine the determinant for which point is farthest away from the current line for every point, then D = 1. The choice of what B is acceptable is not as clear, since the data set may decrease by a different percentage for each recursive call, depending on the set placement. I’ve decided that B = 2 is the closest to average case and uniform point distribution so, I’m using it.

This gives a Master Theorem of for Quickhull:

**T(n) = 2 \* T(n/2) + n1**

By application of case 2 of the Master Theorem of A = BD, this leads to a result of the average case upper bounds asymptote of:

**Quickhull Time Efficiency = O( n \* log2n )**