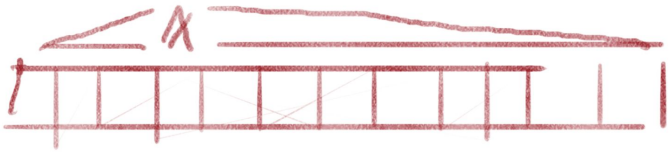
Lisa Xu
108059610
Timothy Chan's Convex Hull Algorithm report

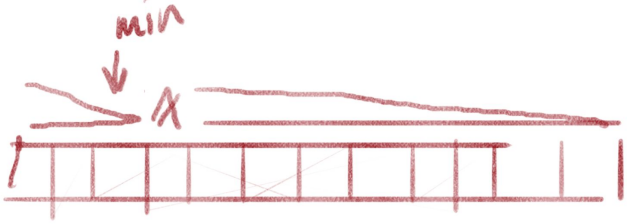
For this project I implemented Chan's O (nlogh) convex hull algorithm. The premise is fairly simple. Take a set of points, divide them up into maxsize n/m subsets, where m is the size of the full convex hull. Since we don't know what m is, we guestimate it with 2^{2^t} where t is each iteration (so it goes 4, 16, 256, etc). Once we divide up the points, we then run graham scan on each subset. Once each subset is graham scanned into subhulls, we then find the tangent point of each hull to the most recently added point to the convex hull, and run Jarvis march over the points, keeping the one that give the largest angle between P_{k-1}, P_k, and the new point.

The first few steps of the algorithm are fairly straightforward. I used a few simple calculations (Left and Area2) as documented from the textbook. For purposes of Javascript and Canvas screen coordinates (more y is down), I had to flip a couple of the variables around to make it work properly. What is nice about JS, is that just clicking points on a screen the most I have to deal with is .5 in terms of decimal points, which is a blessing because Javascript isn't the best when it comes to floating point precision. I implemented Graham Scan fairly close to the pseudocode described in the slides/textbook, as well. No surprises there.

The Jarvis marc bits are definitely the most difficult to implement. First I tried to just, straight binary search for the highest angle produced, but it did not work, because the order the angles in term of size compared to index are not in this order (which would allow classic binary search to work directly with some finagling)



But rather, like this:

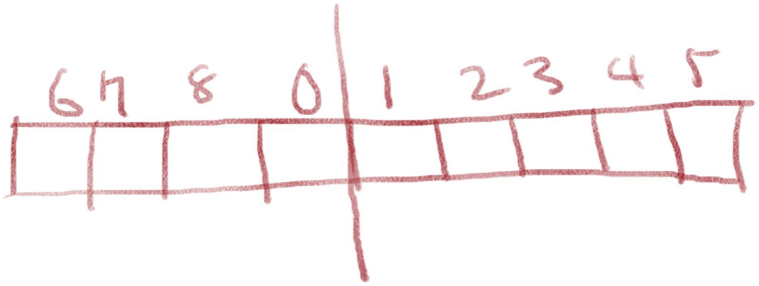


Lisa Xu
108059610
Timothy Chan's Convex Hull Algorithm report

So after that failure I decided to look up how to get tangents to a polygon from a point, but that was also very tricky and I couldn't quite get a few edge cases right. I didn't want to have a convex hull algorithm that only worked 90% of the time, so I tried something else.

I thought, what if instead of searching single value in a binary search, I search the start and end of each block? It's 2 queries instead of 1, but 2logn is still O(log(n)) time. I drew a couple of test arrays on an whiteboard and it looks like it works, so I go and implemnt it in code. Instead of a single index, I instead check the Max(start, end) of each range of indices against the other half, and do binary search that way.

It works. The counter-clockwise rotation allowed this to work--a true unsorted array would not have worked like that.

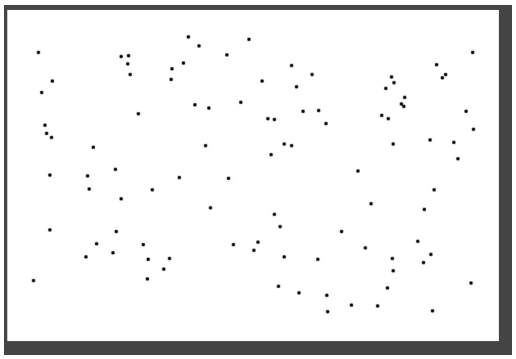


Here's the diagram of how some of them are ordered-in traditional binary search, which is what I did the first time, the split would cause me to go to the right side, and I'd never be able to reach 8, the true maximum. However, because of the wrap-around nature, even though it's not fully ascending/descendingly sorted, since 5 is cut off at the end, but not the largest, then the the index of 5 +1 mod the length (aka index 0) would have a greater value than 5 if 5 is not the biggest number in the array. So max(6,0) > max(1,5), and the side with the larger values win out.

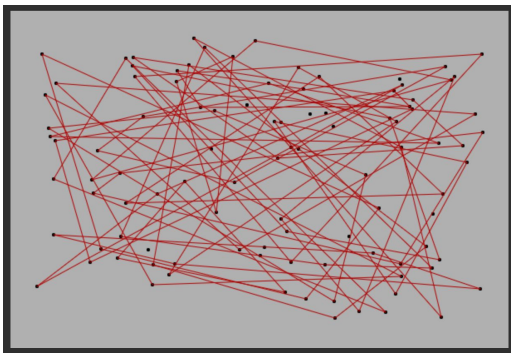
Anyway this works out, and I get my convex hull. I ran a bunch of (rather manual) tests to see if there's no odd points out. This should work when you generate a bunch of values and then manually add points, too. You can even generate a bunch of points, generate the convex hull, add some points manually and it'll generate again.

Anyway, some screenshots of the algorithm in motion, thanks to chrome's built in breakpoints (uncomment and breakpoint at line 272 to see jarvis march in motion)

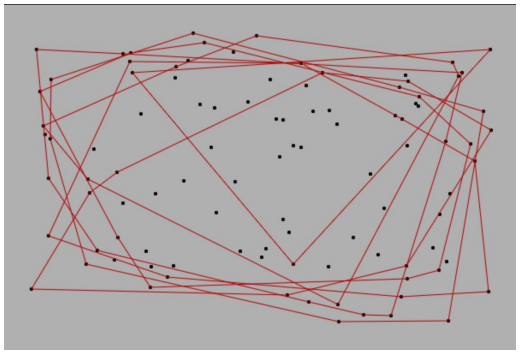
Lisa Xu
108059610
Timothy Chan's Convex Hull Algorithm report



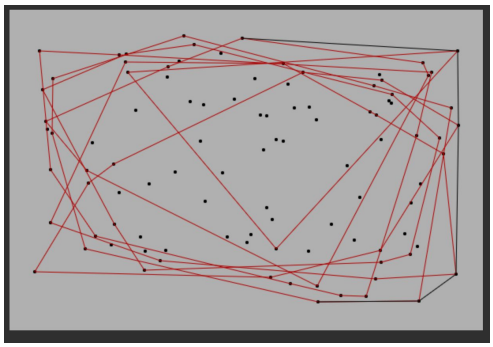
100 points.



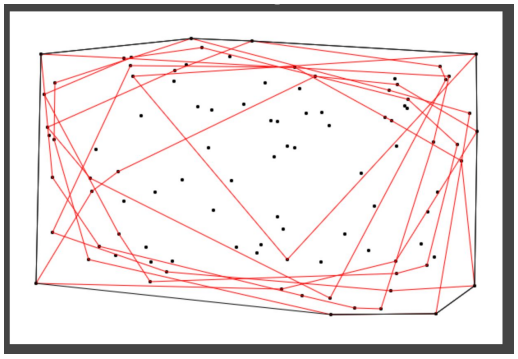
m=4. Looks like a mess. There *is* 100 points.



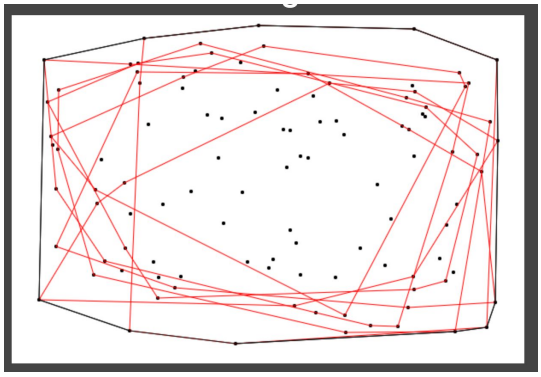
m = 16. Much cleaner.
This looks likesomething we can use.



Black shows the outer hull in Jarvis march.



Complete Convex hull.



Manually add more points? Still works.