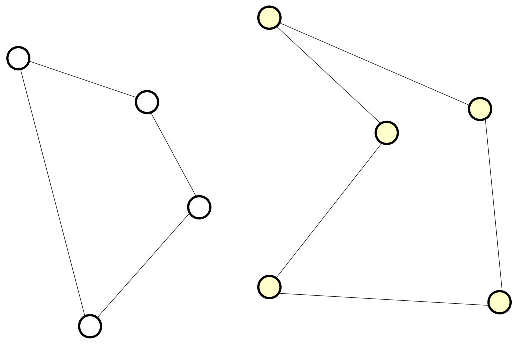
20190919

Convex Hull Algorithms

Given a set of points in the plane, the **Convex Hull** of the points is the smallest convex polygon that includes all of the points. Informally, a polygon is convex if it has no "dents" in it. More formally, a polygon is convex if there are no points in the polygon such that the straight line between goes outside the polygon.



In this diagram the polygon on the left is convex but the one on the right is not

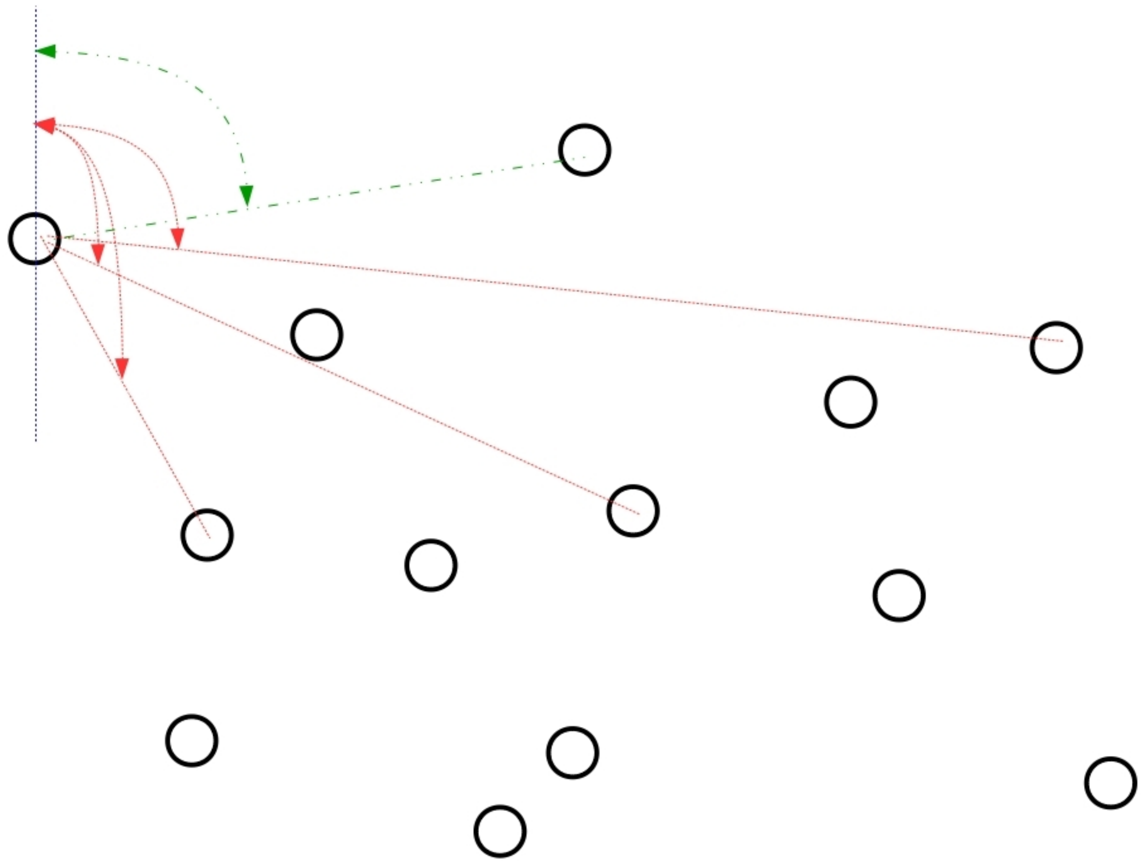
Assume we have n points, each identified by its x and y coordinates. Our goal is to find the convex hull of these points. The first method we will discuss is called The Jarvis March (aka the Gift Wrapping algorithm).

The Jarvis March starts at a point which is known to be on the convex hull (even though we haven't yet found the convex hull). One way to choose such a point is by finding the left-most point of the set. It is easy to show that this point must be on the convex hull.

Now we can visualize a roll of wrapping paper anchored at the left-most point, and extending upwards as far as the eye can see. We pull this paper to the right, keeping it taut – eventually it hits and bends around some point in the set. That's the next point on the convex hull. Now we repeat the process for this point, and so on round the exterior of the set of points until we come back to where we started.

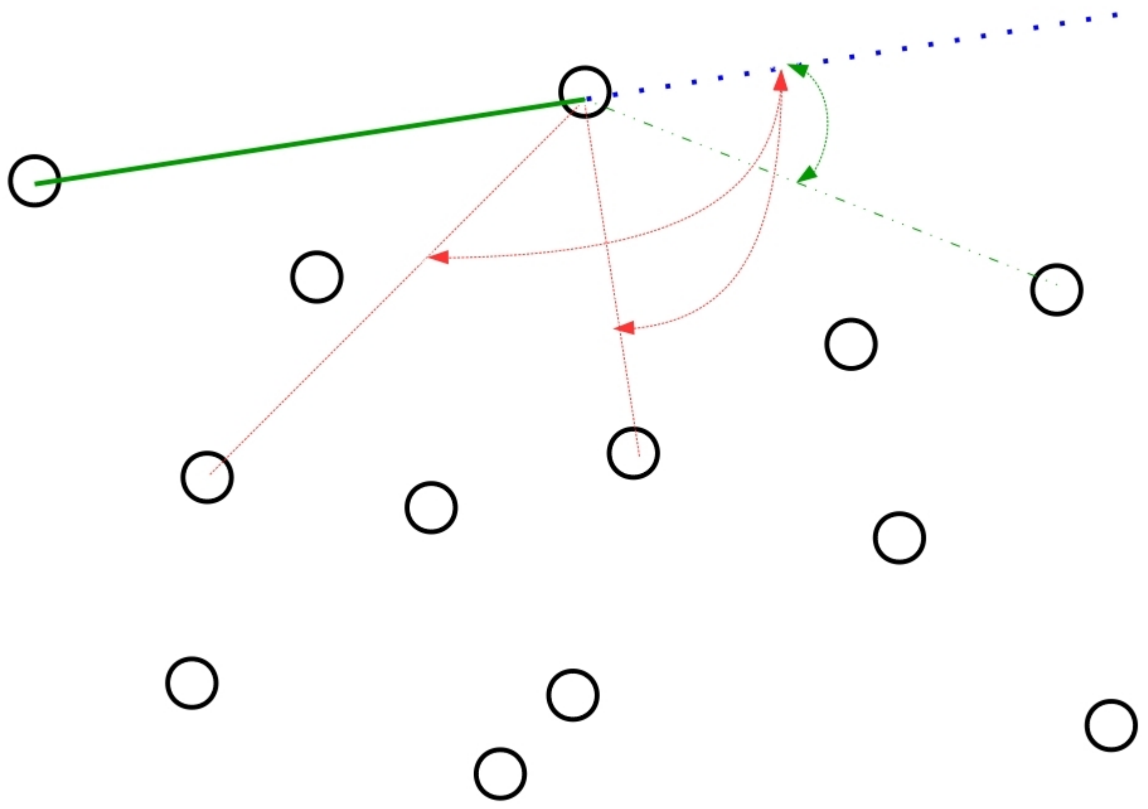
Of course we need to simulate this gift-wrapping process rather than do it for real. The way we do that on the first iteration is we examine all the other points one by one and choose the one for which the straight line from the starting point to the new point is closest to the vertical guide line.

The figure below shows this process. The vertical guide line that we use on the first iteration is shown in blue. A few of the lines and angles are shown in red. The smallest angle and the line to the point that we choose as the next convex hull point are shown in green.



Once we have the first line segment of the hull we continue the “wrapping” process from the point we just found, using the previous line segment as the reference line for comparing angles. Again we compute angles to all the remaining points and choose the one that is closest (in the angular sense) to the reference line. For the second iteration and all future iterations we include the starting point as one of the candidate points for selection. Can you see why, and can you predict what would happen if we did not?

The next figure illustrates the second iteration. Again, only a few of the rejected candidate line segments are shown – they are red – and the selected one is shown in green.



It should be clear that the Jarvis March correctly finds the convex hull – no point can be left on the outside, and any smaller convex polygon would have to omit at least one of the points in the set. So we can turn to the question of efficiency.

Finding the left-most point takes O(n) time. To choose the second point we need to compute n-1 angles, each of which can be computed in constant time. To choose the third point, we also need to compute n-1 angles (as mentioned above, the starting point is also a candidate). After that the number of angles to compute decreases by 1 on each iteration ... and we continue to iterate until we come back around to the starting point.

Thus the total number of angle calculations is bounded above by
$(n-1)+(n-1)+(n-2)+\cdots+2+1$

which is simply $\frac{n(n+1)}{2} - 1$ which is in O(n^2)

However we can look at the complexity of this algorithm in a slightly more careful way. The algorithm stops as soon as it completes the convex hull – so the process of selecting the next point only happens h times, where h is the number of points on the convex hull. Each iteration takes O(n) time so we end up with a complexity of O($h * n$)

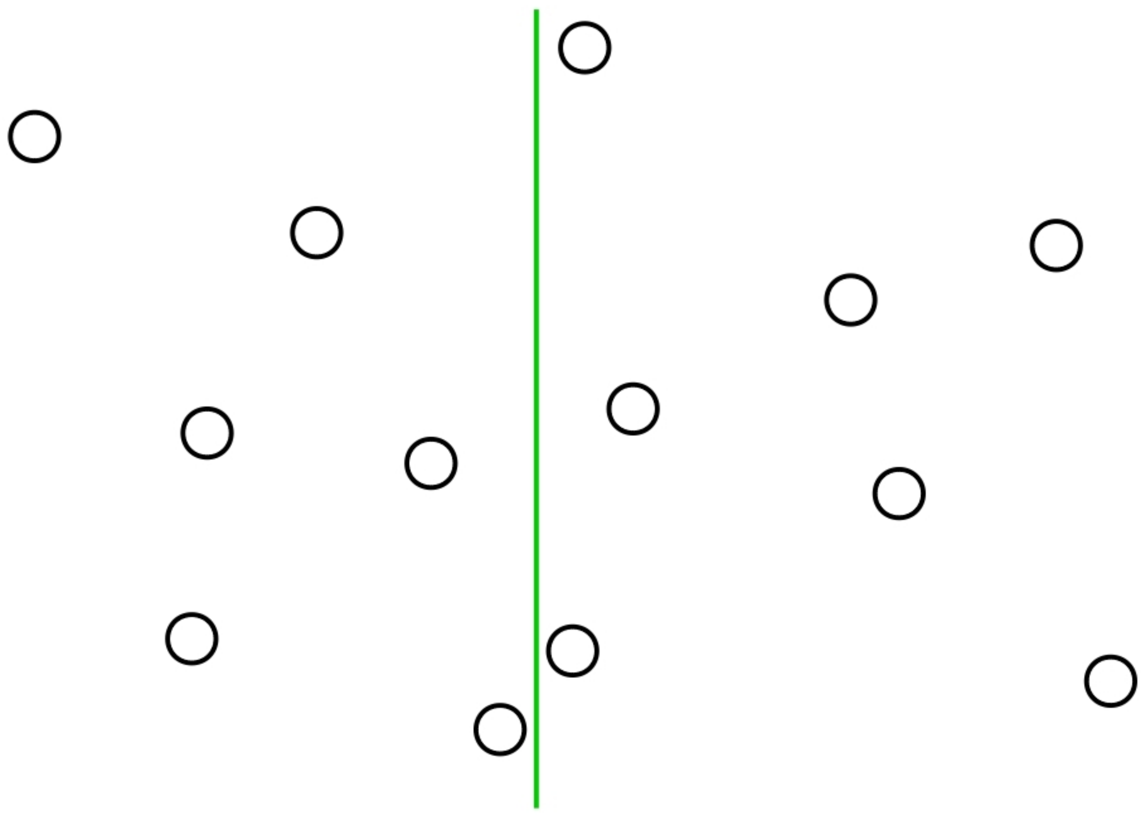
It's possible that every point in the set is on the convex hull – in this case the complexity works out to O(n^2) which is what we had before. But if the convex hull is a simple polygon with relatively few sides – in other words, if most of the points in the set are *inside* the convex hull – then h may be much smaller than n, and $h * n$ much smaller than n^2. In the best situation we may know that the convex hull has no more than some fixed number of corners – let *k* be this number. In this case the algorithm takes O($k * n$) time, and since *k* is a constant this reduces to O(n)

It seems plausible that for a random collection of n points in the plane, most of them will be inside the convex hull. If this is a valid assumption, then Jarvis March is an excellent choice.

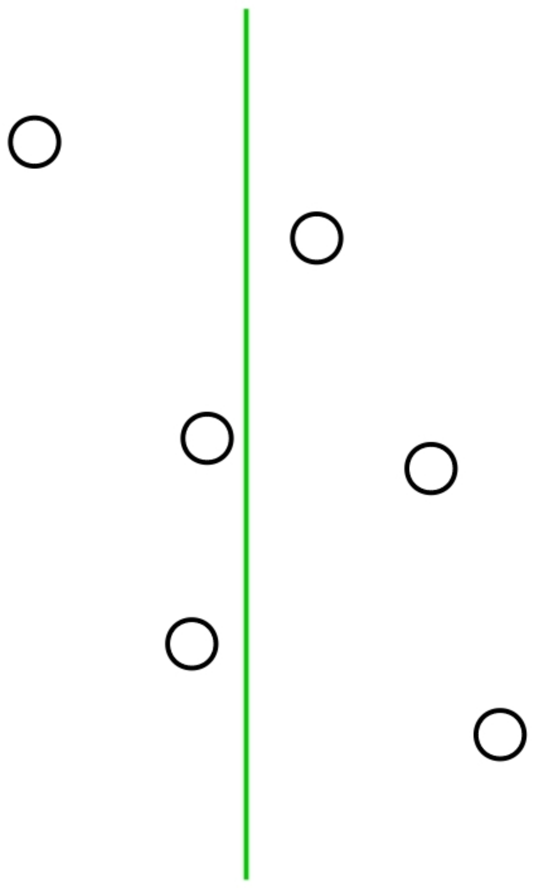
But because it is *possible* that all the points in the set are on the convex hull, the worst case of Jarvis March is still O(n^2). Can we use our friend Divide and Conquer to find an algorithm with O($n * \log n$) complexity, regardless of how many of the points are on the convex hull? Of course we can!

Our D&C approach will be quite predictable: divide the set of points into a left half and a right half, find the convex hulls of the two haves recursively, then combine the convex hulls from the two halves to get the convex hull of the complete set.

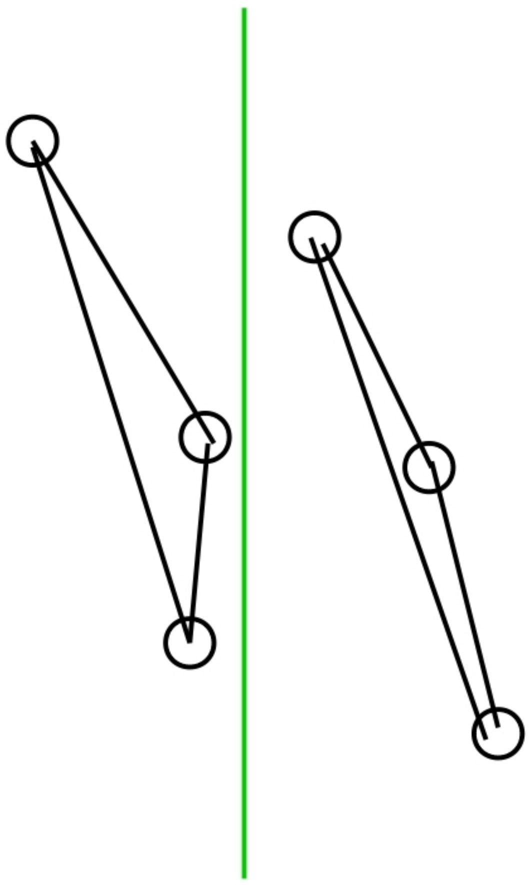
So we start by sorting the points from left to right – this takes O($n * \log n$) time.



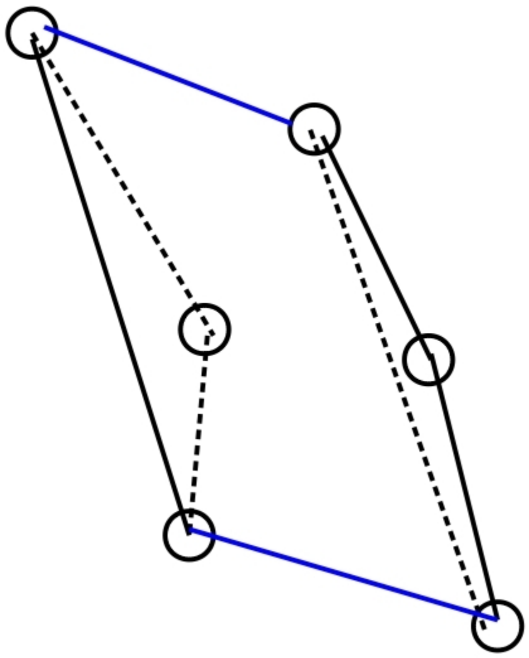
Here we see the points divided into left half and right half. The right half has 1 more point than the left half. That's ok. Let's focus on the left half. We apply the same algorithm recursively so we split it into left and right halves.



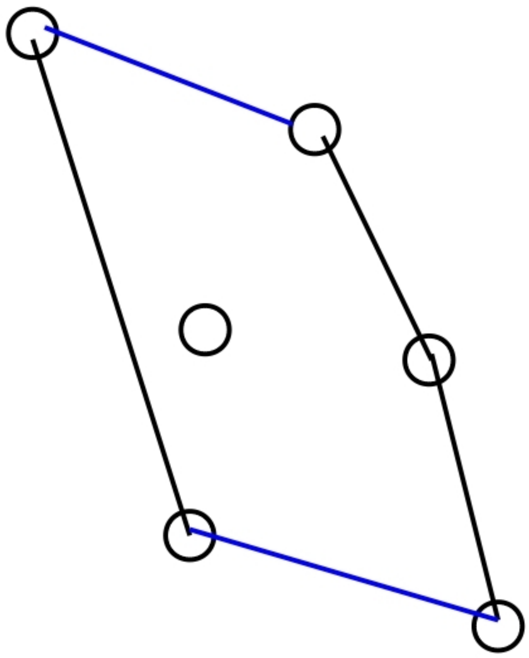
Now there are just three points on each side – and the convex hull of three points is just a triangle (unless all three points are colinear, which we normally prohibit when dealing with this problem). So we can treat these as base cases and solve them directly by building the triangles for these two small sets of points.



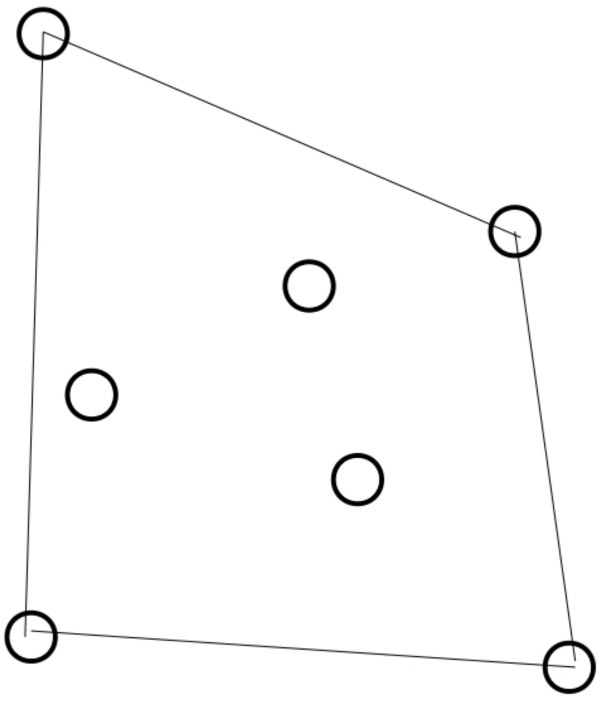
Now we need to combine these two convex hulls into one bigger one. The basic technique is to connect the two top points and the two bottom points. In this case this works perfectly.



Now we eliminate the *right side* of the left convex hull, and the *left side* of the right convex hull, as shown by the dotted lines in the figure above, and we end up with

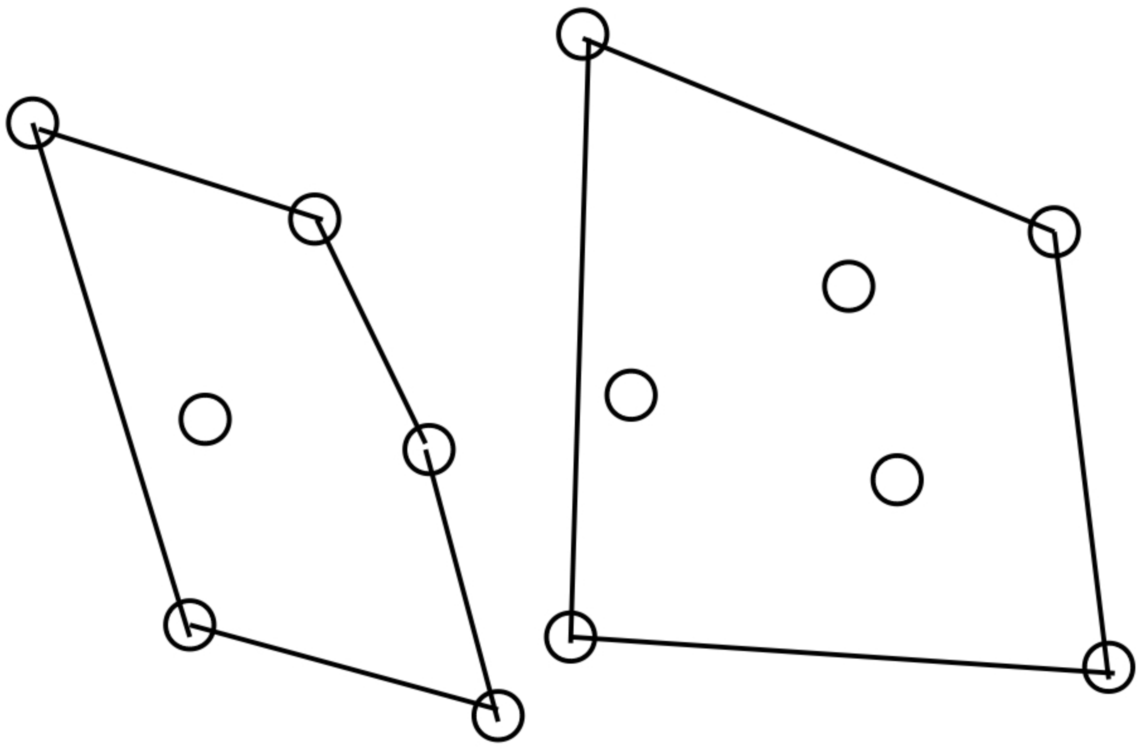


Repeating this for the right side, we end up with

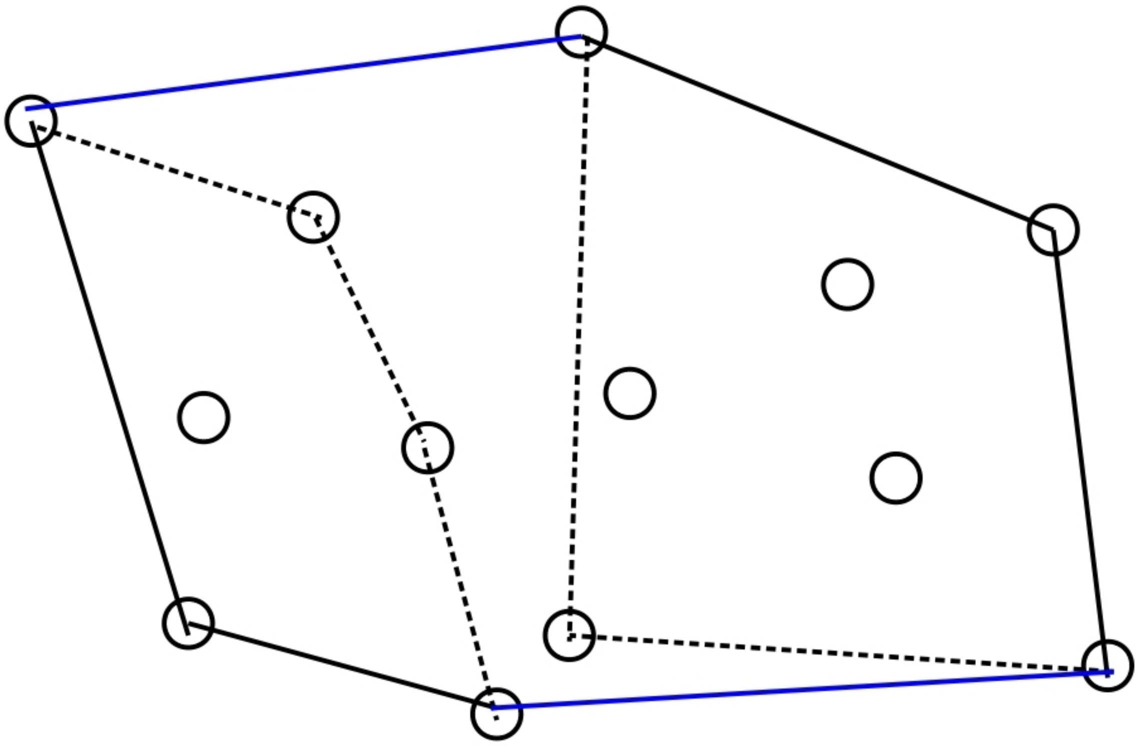


Note that as we recurse down on the right side, we start with 7 points so when we split the points into two sets we get 3 on one side and 4 on the other. We can treat both sides as base cases because computing the convex hull of 4 points is very easy (think about how to do it!). In practice we might want to treat a set of 5 points as a base case too.

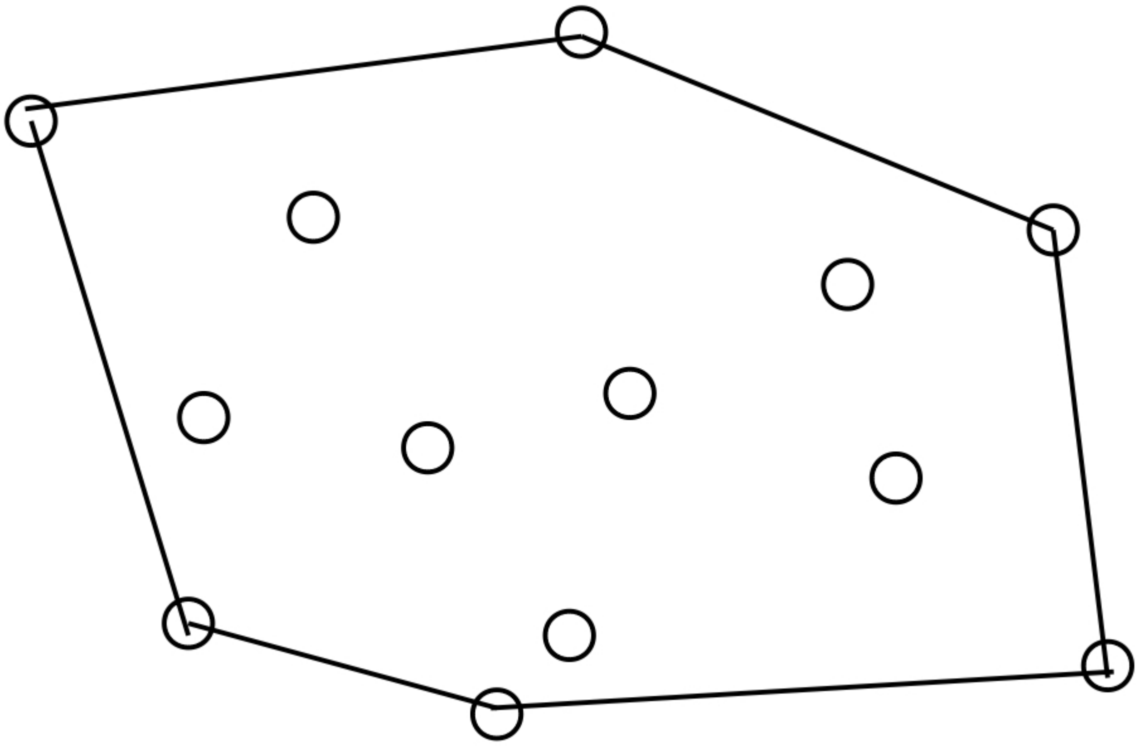
So now we have



Once again we join top to top and bottom to bottom, and discard the sub-paths that are no longer relevant, getting



and finally



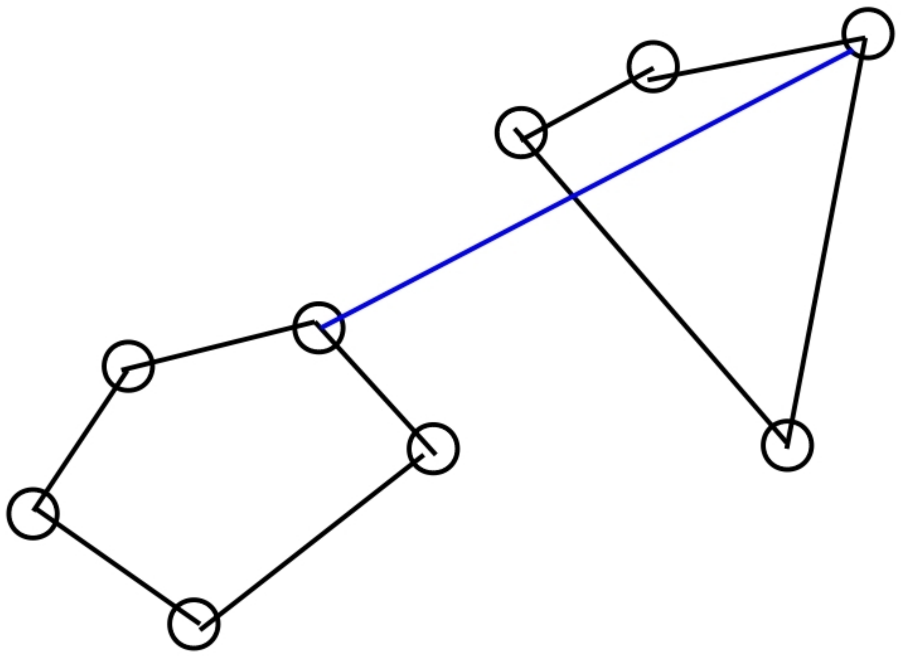
So what is the complexity of this algorithm? Joining the tops together and joining the bottoms together takes constant time, and eliminating the segments we no longer need takes O(n) time because we just have to "walk around" the convex hulls from the two sides, eliminating edges as we go. So the recurrence relation looks like this

$$\begin{aligned} T(n) &= \text{constant for small } n \\ &= 2 * T\left(\frac{n}{2}\right) + c * n \qquad \text{for large } n \end{aligned}$$

which is exactly the same as for the other problems we have looked at ... we know this means T(n) is in O($n * \log n$)

BUT!!!

The joining together doesn't always go quite as smoothly as it did in our nice little example. Here is a situation that can arise when we join the tops (and the same things can happen at the bottoms).

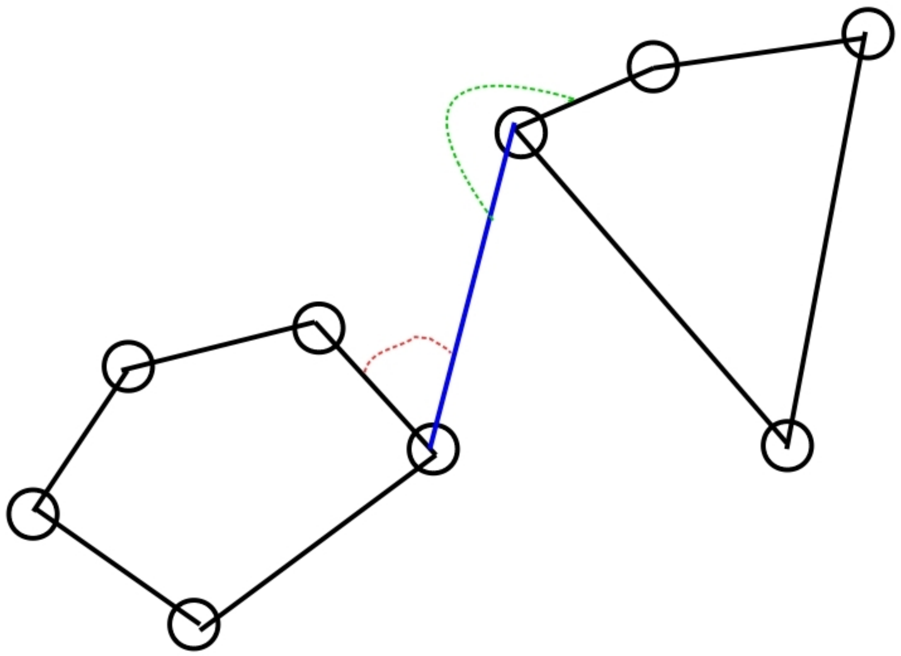


Here we see that the line joining the tops of the left side convex hull and the right side convex hull actually cuts off a couple of points on the right side.

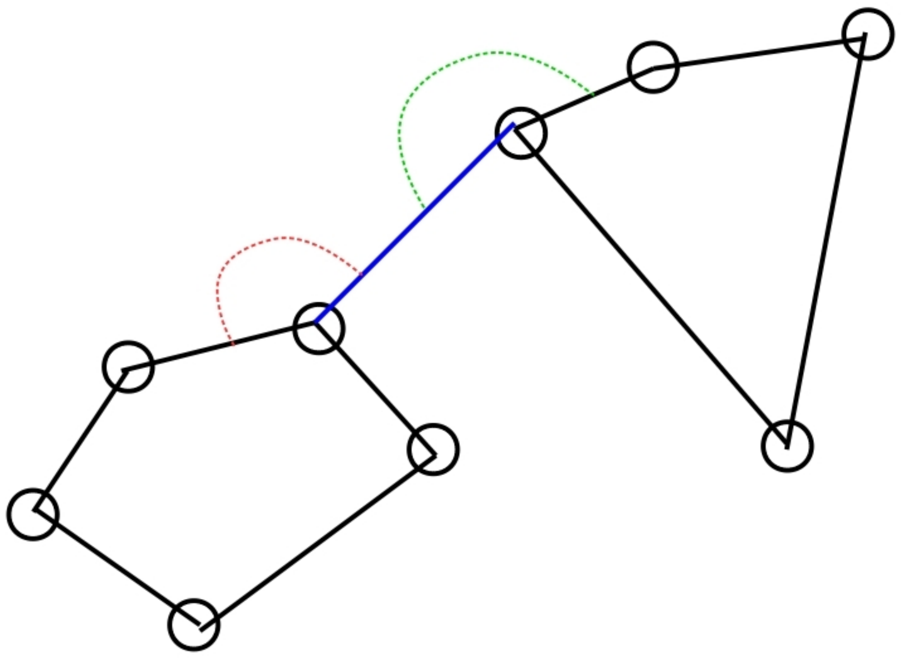
In class I presented a method for getting the top connecting line properly placed that involved starting with the line joining the top point on the left side convex hull to the top point on the right side convex hull, then adjusting the ends to resolve any problems.

Here is another way:

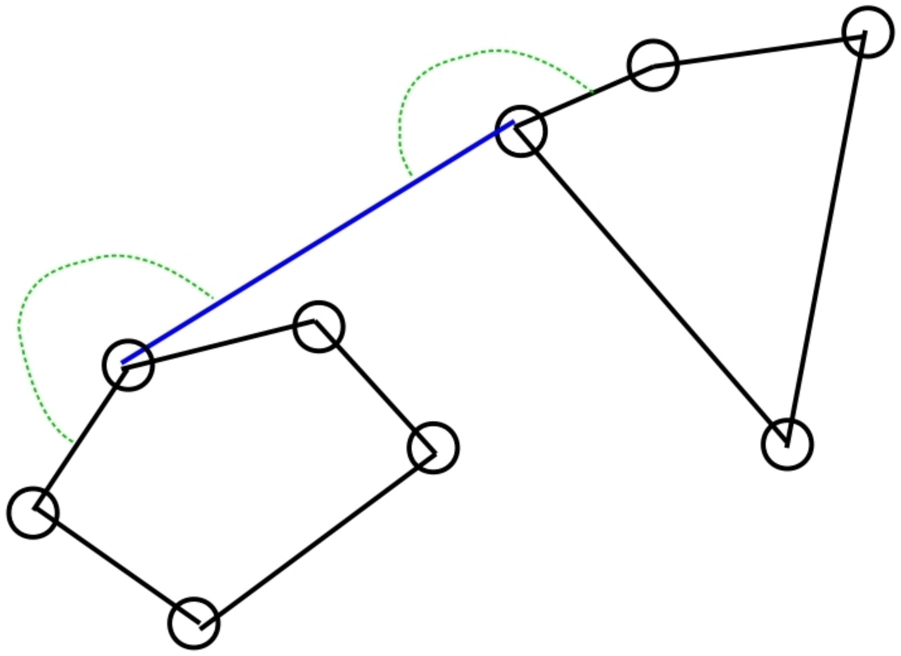
Start by connecting the right-most point on the left side's convex hull to the left-most point on the right side's convex hull. Look at the angles from this line to the line segments immediately "above" the chosen points. If either of these angles is ≤ 180 degrees, shift that end of the line to the next point. Continue this process until both angles are > 180. The following figures show an example of this process. "Bad" angles are shown in red. "Good" angles are shown in green.



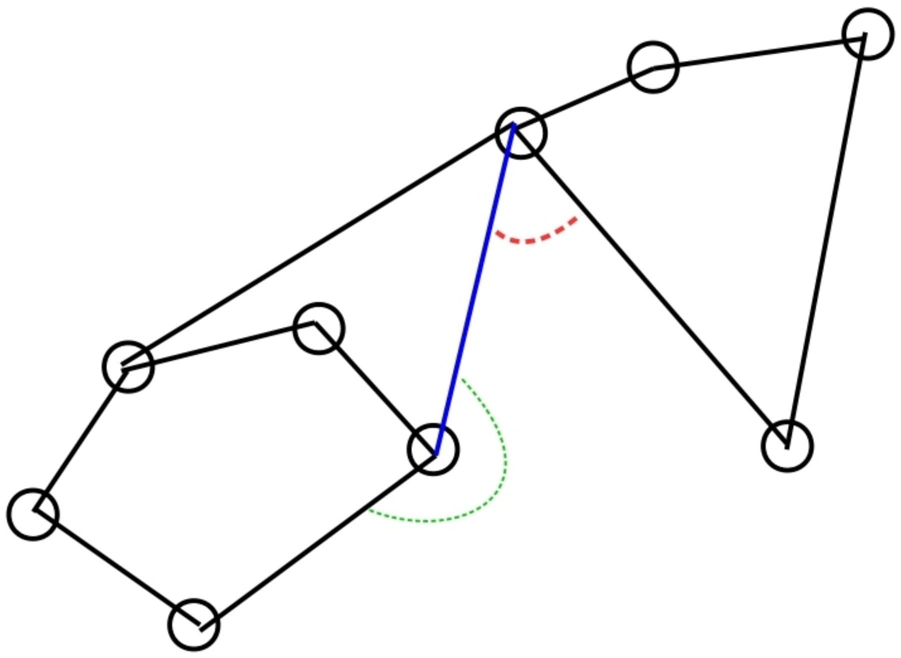
Above we see the initial position of the "top" connecting line. The angle at its left end is bad so we shift this end to the next point.



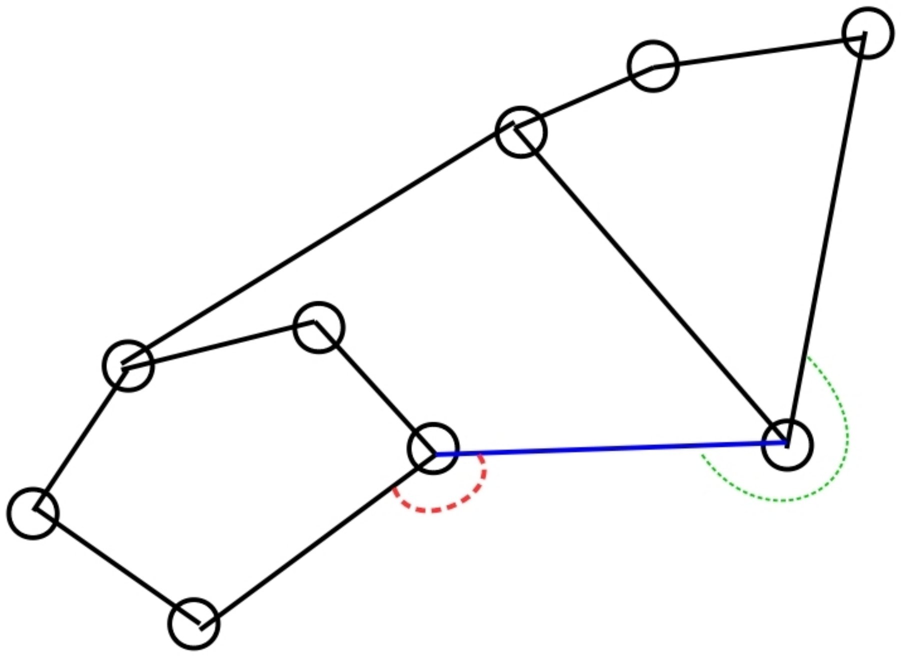
The left end angle is still bad so we shift again:



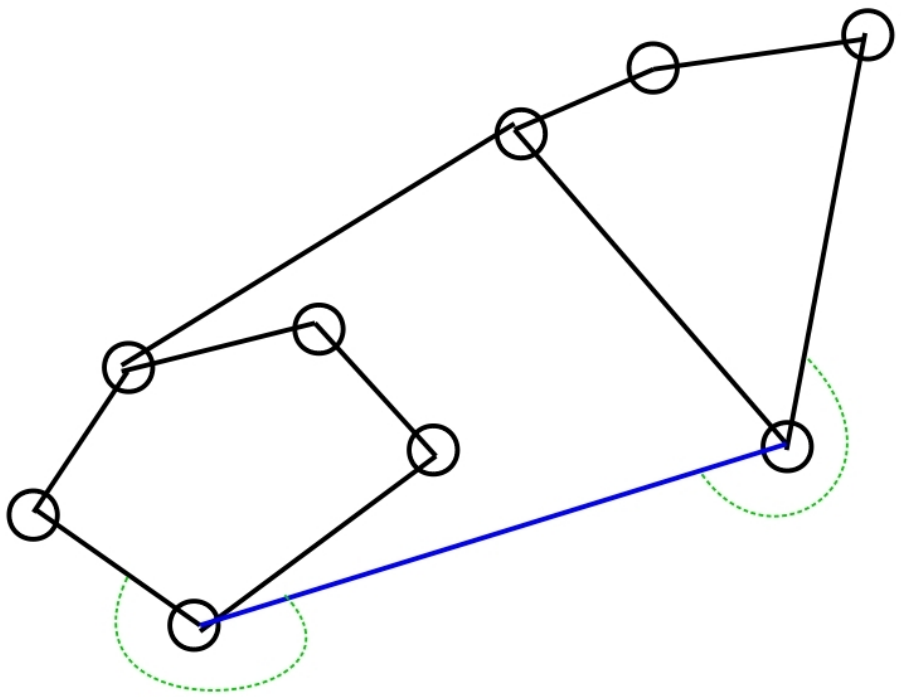
and now the line is in the correct position.   Now we follow exactly the same process for the bottom connecting line, except we shift "downwards" instead of upwards.



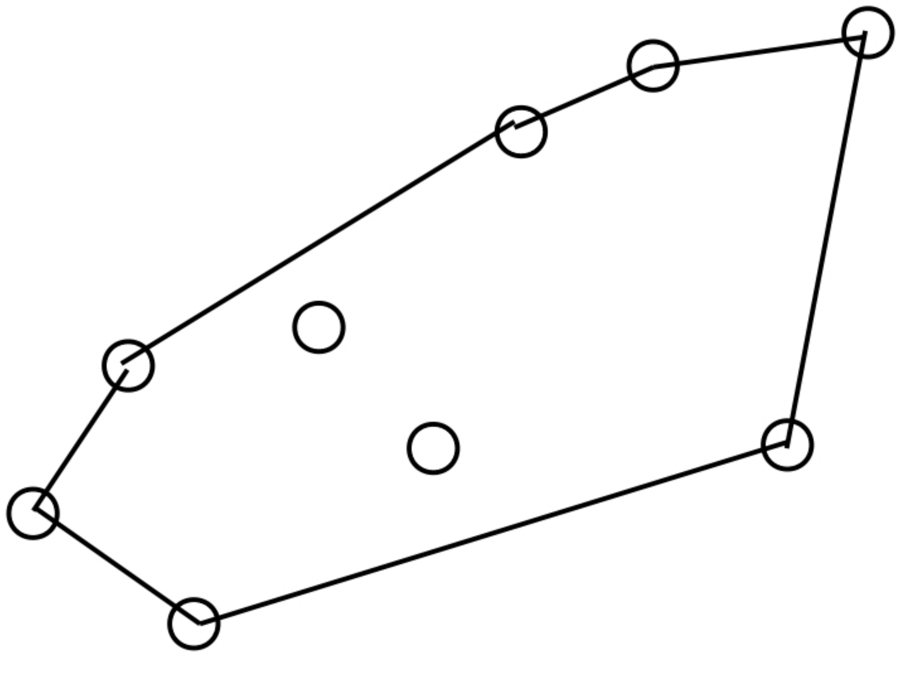
We move the right end down ...



then the left end down ...



and finally we remove the pieces of the two sides' convex hulls that are now internal



It is easy to see that this process is in O(n) because there are only n possible points that the line ends can move to.

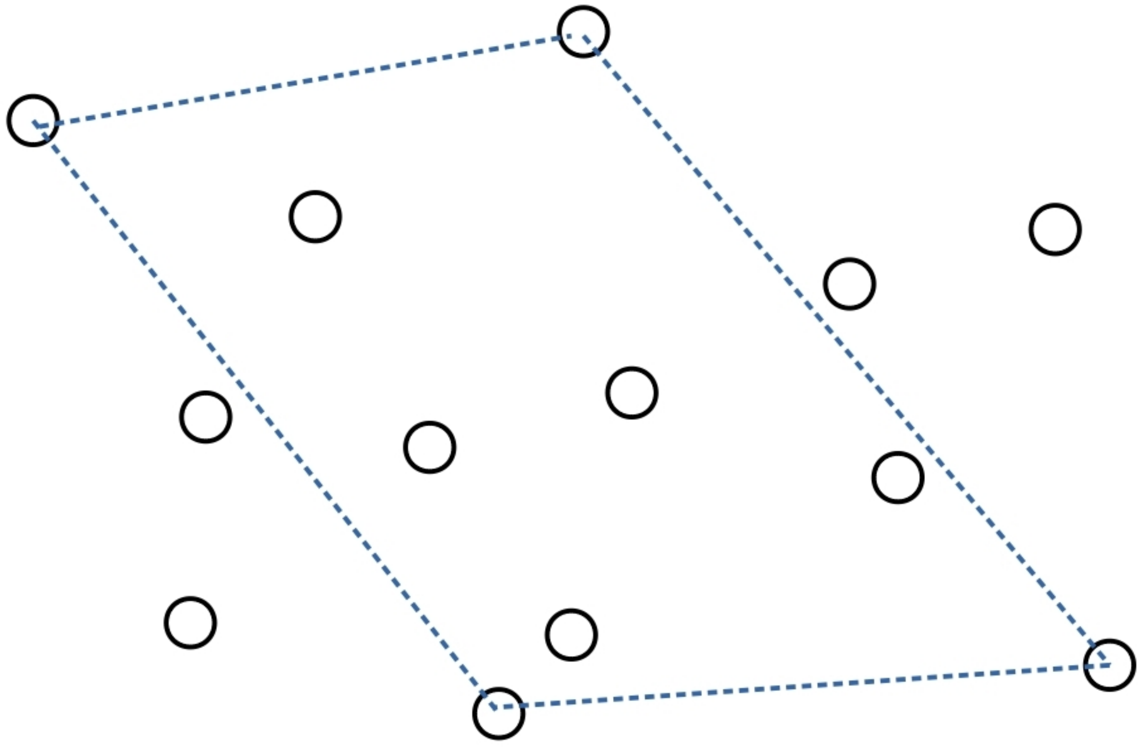
Thus the overall complexity of the algorithm is in O($n * \log n$), just as we said it was before we got all caught up with solving special cases.

Let's return now to the idea that in "real" sets of points (whatever that means) most of the points will not be on the convex hull. It would be nice if we could eliminate some of the points from the set before we start our processing – this should reduce the amount of work we have to do.

Well, we can! Let p be any of the points in the set, and suppose that p is enclosed in a convex polygon formed by some of the other points in the set. It's not hard to see that p *cannot* be on the convex hull: any convex polygon with p as one of its corners would have to exclude at least one of the points that surround p.

So if we consider the left-most, right-most, highest and lowest points in the set, these points form a convex quadrilateral – any points within this polygon can be eliminated from consideration for the convex hull. (Caveat – we can run into problems here if the left-most point (or the right-most) is also the highest (or lowest) point. I'm going to ignore this issue here and just assume that these four points are all distinct.)

Here’s our example again:



The quadrilateral formed by these four points encloses more than half of the remaining points in this set – we can discard these internal points. Whether we proceed with the Jarvis March, the Divide and Conquer, or some other convex hull algorithm (and there are lots of others) our algorithm will run faster because there are now fewer points to consider.

What’s the complexity of deciding which points to eliminate? Testing to see if a point is inside the quadrilateral takes constant time, so we can do it for all the remaining n-4 points in O(n) time.   It’s basically free1

Computational Geometry is a huge field full of interesting problems, many of which are particularly relevant in this age of autonomous vehicles. I encourage you to explore!