COSC262 Assignment: Convex Hulls

**Algorithm Implementation:**

For this assignment I implemented 3 convex hull algorithms, gift-wrap algorithm, graham scan, and quickhull.

*Gift wrap implementation details:*

For the implementation of the gift wrap algorithm I defined auxiliary functions for finding the point of minimum y-value on the convex hull, and a function for swapping a pair of points on a list. I also used the theta function from the slides, but modified it to return 360 under the correct circumstances. Within the main function for the algorithm I maintained two lists, one that was a copy of the list of points to be used in swapping the points around, and a list that contained the indexes, which was also swapped whenever the list of points was swapped in order to preserve the order of the convex hull indexes. The outer loop runs through the list of points on the hull and the inner loops through all the points that aren’t already on the hull to find the next minimum angle.

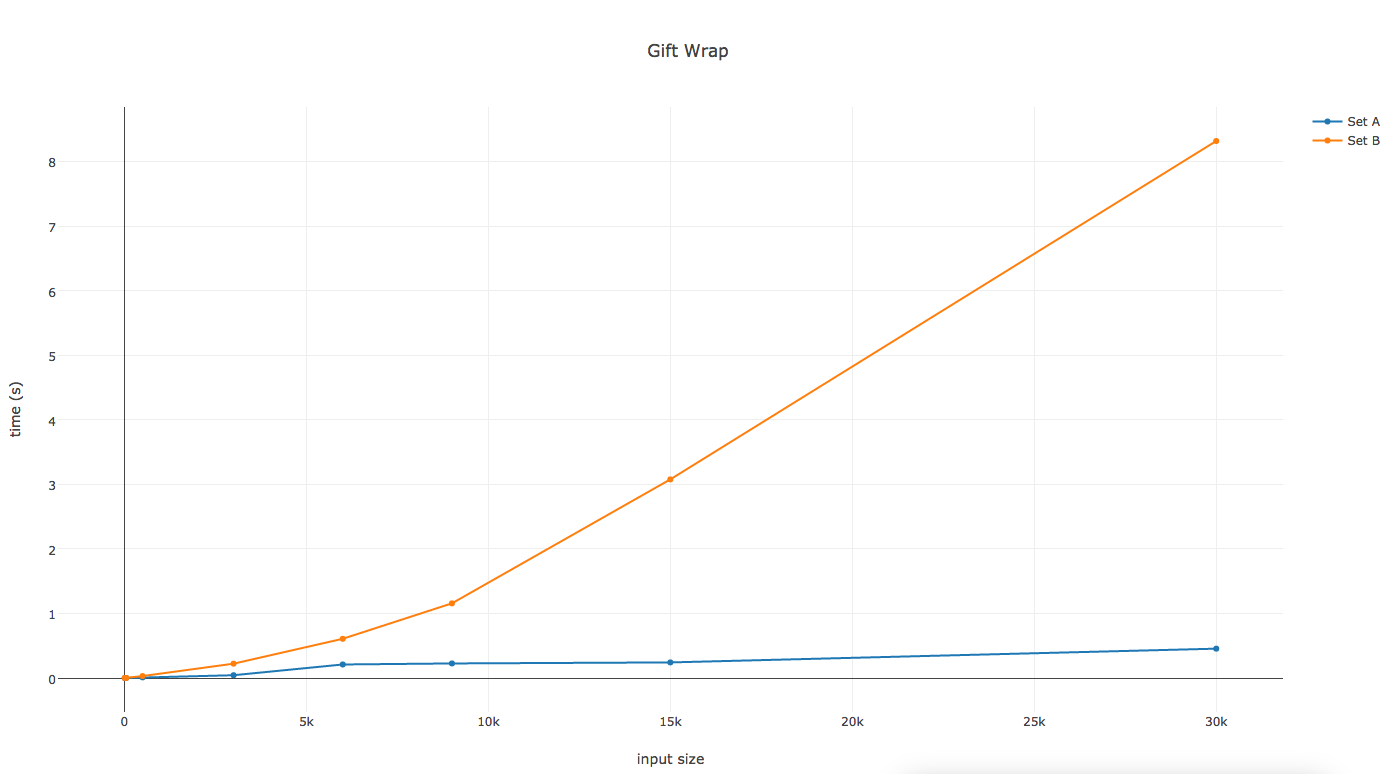
*Graham Scan implementation details:*

The graham scan algorithm also uses the minimum y-coordinate as the first vertex on the hull. The algorithm then requires a sorting of angles from this minimum y-coordinate point. To sort the points I used merge sort, which in turn uses the theta function. In order to preserve the order of indexes I defined a function to add the index to the tuples of points, so that each point was now (*x, y, i*) where *i* is the index of that point. I used a stack, initialized with the first 3 points of the sorted list of points. Points in the sorted list that formed a counter clockwise line, checked using the isCCW function from the slides, were then added to the stack, and those that do not were removed.

*Quickhull implementation details:*

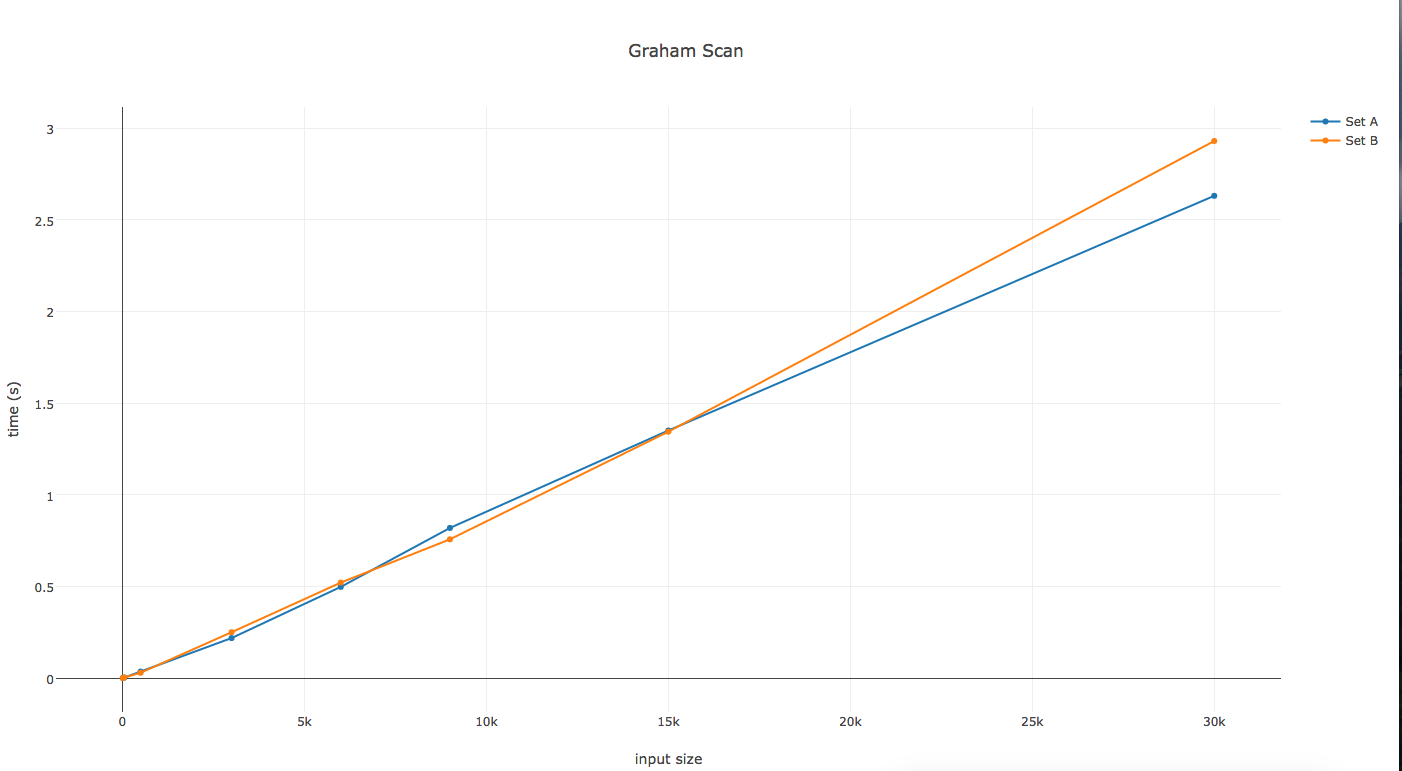
For the third algorithm I implemented quickhull. The quickhull algorithm finds the right-most and left-most point on the hull. These two points form a segment, and the set of points is partitions into points to the right of this line and points to the left of this line. This is done using the line function, linearly checking each point. The segment and the left and right subsets are passed to a recursive function. This function finds the furthest point C from a line segment AB, computed in a separate function. The point C is added to the hull. The points inside the triangle formed by the line segment, AB and the new point C are not a part of the convex hull. The function then finds two new sets, a set of points to the right of the line from C to B and points on the right of the line segment from A to C. This partitioning is again done using the line function. The function is called on these two new subsets and their respective line segments. The function continues until there are no more points to add to the hull.

**Algorithm Analysis:**



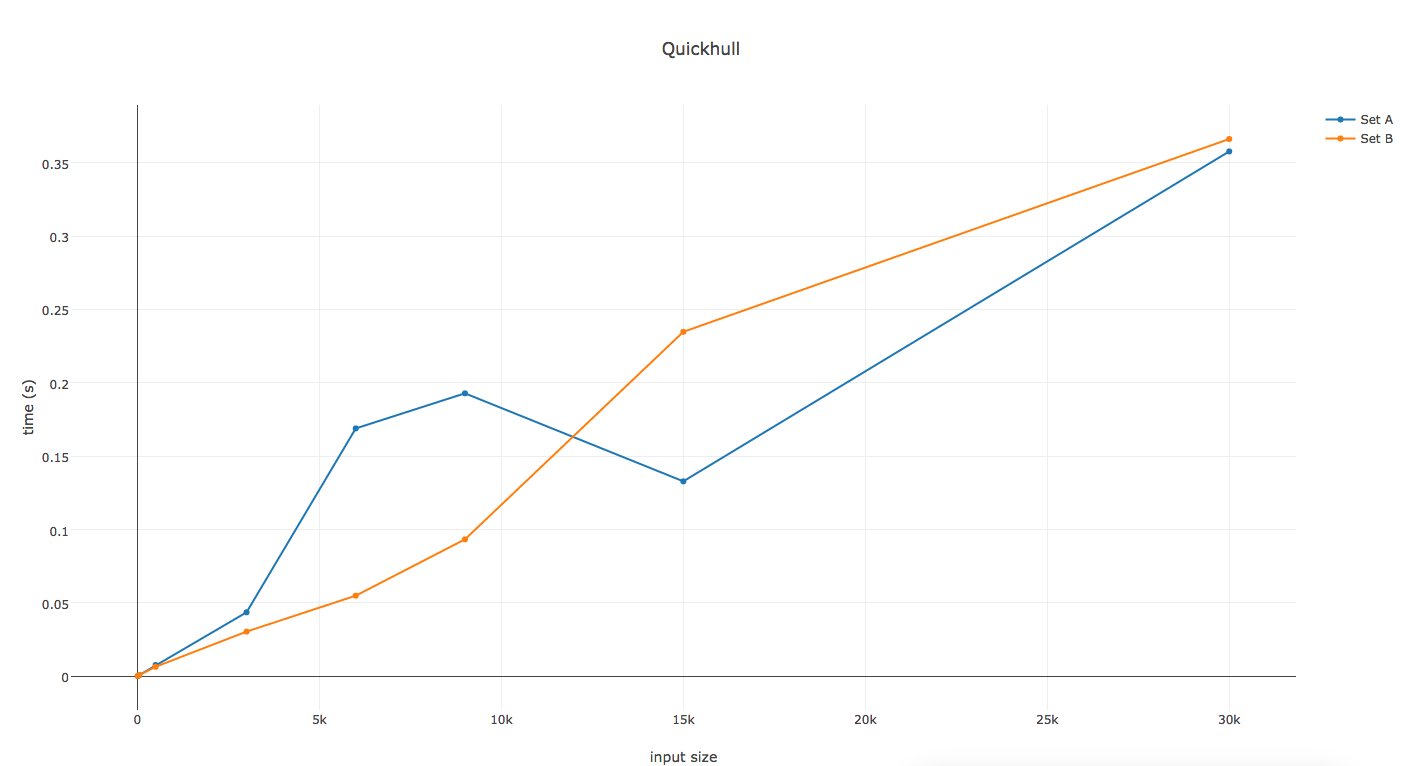
Graph 1. Gift Wrap Algorithm Performance

In graph1, the gift wrap algorithm performs significantly better for set A than set B. The performance for set A takes under 1 second for all input sizes. The theoretical complexity of this algorithm O(mn) where m is the number of points on the hull and n is the number of vertices in the graph. For set B the number of points on the hull is greater, especially as the input size increases so the algorithm would perform for set A than set B, as was the case.



Graph 2. Graham Scan Algorithm Performance

The graph above shows performance of graham scan for sets A and B. Their performance is quite similar for all input sizes, and under 3 seconds. The theoretical complexity of this algorithm is O(nlogn), which is the sorting step. You would expect similar performance for set A and set B since the complexity depends on n, the input size, as was the case in the graph.



Graph 3. Quickhull Algorithm Performance

Graph 3 shows the performance of the quickhull algorithm. Set A and B had similar outputs for the beginning and ending input sizes. There is flunctuations in the performance for the other input sizes. The average complexity of the quickhull algorithm is O(nlogn) while the worst case is O(n2). The quickhull algorithm performs better under conditions when the number of points in each set after the division step is nearly the same. The performance shown the graph matches the theoretical preditcition.

Overall performance:

The quickhull algorithm performed the best out of the 3 algorithms for the largest input sizes, finding the points on the hull in under 0.35 seconds for both sets A and B. This algorithm performed well for these inputs since the division along the left most and right most point led to balanced subsets. The graham scan algorithm while of similar complexity might have performed worse since it also has to finding the rightmost lowest point, which takes O(n) and the nested for loop which also performs in O(n).

References:

<http://www.ahristov.com/tutorial/geometry-games/convex-hull.html>

<https://en.wikipedia.org/wiki/Quickhull>