Convex Hull Problem - Honors Project

Thomas Bergendahl

December 2022

1 Invariant Documentation

- Upon receiving the input, my algorithm breaks the points down into smaller sets, until the sets have no greater than k (some arbitrary constant, for benchmarking I ran the algorithm using $k = 5$) points in each set.
- Then, my algorithm runs the base-case, naive algorithm on the smallest sets. This algorithm runs in $O(n^3)$ time. As such, it's illogical to run this on large inputs. But these sets are of k size, so we run the algorithm here, and we have some number of sets with valid convex hulls. Then, we begin merging these correct convex hulls.
- While merging, my algorithm groups two sets at a time, which are each accurate convex hulls for those sets of points, and uses minimum and maximum tangents to construct the new convex hull covering both sets in $O(n)$ time. This property is invariant - at each iteration, each individual set is represented in the correct clockwise order of its convex hull. When merging, assuming that both sets have the correct convex hull, my algorithm will always produce the correct convex hull from those two sets.
- This means that the last step will take the last two sets, and correctly produce the full convex hull. We are done once we are left with one set of points (which is represented in the correct clockwise order of the convex hull). Because the sets were split into halves over and over again, then merging took linear time (though often was much quicker than linear), the algorithm's asymptotic runtime was $O(nlogn)$.

2 Benchmarking Analysis

- When comparing the results of benchmarking the naive base algorithm to the divide-and-conquer algorithm, it's evident that the naive algorithm quickly becomes outclassed. I was unable to realistically run the naive algorithm systematically for inputs of greater than 750 points. At the 500 point mark, building the convex hull took around 3 seconds. On the other

hand, the divide-and-conquer algorithm was still computing the convex hull almost instantly at this size of input. This was to be expected, as the naive algorithm was exponentially slower than the divide-and-conquer algorithm.

- Examining the trajectory of the divide-and-conquer algorithm at large inputs, the asymptotic runtime appears to be nearly linear. It can certainly be bounded above by $O(nlogn)$. In this case, the algorithm is running faster than its worst-case expectation. My explanation for this is that the worst-case runtime for merging two hulls is $O(n)$, in relation to the size of the hulls, but it is often faster than this. It only requires a few iterations of moving down and up both convex hulls in order to find the resulting tangent lines. Additionally, merging often removes points from the sets, as they are not included in the convex hulls. So our problems are shrinking as we merge up.