Some changes were made to the overall design. Most of these changes were the data types used, such as using a vector in the main function, or changing functions in the classes. Most of it was just for preference and doesn't greatly alter how the algorithm runs. I removed the typing on the classes. This is because ints could result in wacky issues when creating the hull. I decided to avoid such a problem and prevent anyone using this code from having that problem by making everything doubles.

Analysis of Render Algorithm

1) Characterize Input: The input is a vector of points that make up the graph. The algorithm also has a set of points as input, but these are simply copies of certain points in the vector. Specifically, the function grows based on the size of the points in the set. Only the set is considered as it contains all the extremes. It is not determined by the number of total points in either the vector or the set. I will consider n to be the maximum of the x-coords and m to be the maximum of the y-coords.

2) The Basic Op: I would consider the basic op to be <<. It appears throughout the algorithm and is the operator that is used the most.

3) Same Count?: As long as the points have the same max x-coord and max y-coord as a previous instance, then the count will be the same. If I was using a queue method, this could vary.

4) Summation Exp.: I cleaned it up a little. I also forgot that strings are mutable in C++ as they aren't in Python, so that helped a bit. A closed form would be $\sum_{1}^{m} 1 + \sum_{1}^{floor(log_{10}n)+1} + 4$. << makes $m + (floor(log_{10}n) + 1) + 4$ appearances.

5) Closed Form: A closed form will be $m + (floor(log_{10}n) + 1) + 4$. I find the digits by a different method, but this is the best way to represent it in a summation. Clearly, it is linear

6) Growth Function: $m + (floor(log_{10}n) + 1) + 4 \in O(n)$, or linear

Analysis of Quickhull Algorithm

1. Characterize Input: The input is a vector of points. The number of these points, or n decides the running time of the algorithm.

2. The Basic Op: The basic op could be considered to be the comparison in *if* (*Quickhull* :: *findSide*(*leftPoint*, *rightPoint*, *points*[*i*]) == *side* && *temp* > *maxDist*).

Since this function is recursive, this basic op doesn't have too much effect on its growth function.

3. Same Count?: No, depending on the positions of the points in the vector, the runtime will vary greatly. The same number of points may have different runtimes if the points are not organized in the plane in the ideal way.

4. Summation Exp.: Since the function is recursive, we will have to create a recurrence relation instead. As mentioned above, the count will not be the same and will vary. As such, I will assume the ideal situation, or best case. A recurrence relation would be $T(n) = 2T(n/2) + f(n)$. Ideally, the number of points we look at are halved each time. We then need to check each point to see if it is the farthest away.

5. Closed Form: Using the Master Theorem, we get $a = 2,\ b = 2,\ d = 1$. Since $2 = 2^1$ $(a = b^d)$, we get $n \log n$. In the worst case, the divisions would be uneven and result in n^2.

6. Growth Function: $n \log n \in O(n \log n)$