StreetMap :: load();

In the most basic sense, this algorithm is taking my input from the file, which we can say consists of N segments. It is running through my ExpandableHashMap, which has an insertion/accessing Big OH runtime of O(1). This is because ExpandableHashMap uses a hashing function to place key-value pairs at certain array indexes. To access that key-value pair, we rehash it, and then access the corresponding array index. If that array index has many other key-value pairs attached to it, then we have to iterate through those pairs. However, because our HashMap is reasonably expandable, hence the name ExpandableHashMap, no array index will have too many key value pairs associated with it. Therefore, we can approximate the runtime of inserting/deleting/accessing an element in our hashmap to be of O(1) time. For this reason, if we are running through N segments of data, and performing an O(1) operation on each of these N segments, our runtime for this function is O(N \* 1), which is O(N). While there is some other complexity within our code, it relates to attaching other segments to the end of the value pair vector of streetsegments for our streetmap. Because each geocoord can only attach to so many segments (really more than 4 is incredibly unlikely), we can pass this complexity off as a constant number, and not a variable depending on any data set. Therefore our time complexity remains O(N).

StreetMap :: getSegmentsThatStartWith();

This function simply accesses the value pair vector of streetsegments that a given geocoord maps to, which it then sets equal to the second parameter of the function: a reference to a vector of streetsegments. Because this code is based off of a fixed number of operations regardless of the data size, our runtime for this function is O(1).

PointToPointRouter :: generatePointToPointRoute();

This function starts by enqueueing a streetsegment out of a data set of N streetsegments, where we link all streetsegments that connect to the current streetsegment continuously, until we reach our desired destination. Each iteration through a loop, and we can say each loop operates through one of the N streetsegments, we call getSegmentsThatStartWith() once, which corresponds to the current streetsegment that we are operating on. We then enqueue all the segments from this call to be operated on in future iterations, and set these segments as visited so that we do not visit them. This means that the maximum number of segments we run over is N, and for each of these segments, we are performing an O(1) operation, which gives a max runtime of O(N).

DeliveryOptimizer :: optimizeDeliveryOrder();

My algorithm is fairly basic, simply rearranging the given deliverycommands in this order:

1. Set your depot as your start point, and then sort the vector of delivery commands by increasing order of distance from start.
2. Choose delivery[0], and push it into a temp vector. Delete delivery[0] from delivery vector, and set start equal to delivery[0]
3. Continue steps one and 2, continually adding delivery[0] into the temp vector, deleting delivery[0], and resetting start.
4. Once delivery.size() ==0, go back from your current location to the depot

This algorithm is operates until deliveries.size() == 0, which means that if deliveries has S items, it operates over each of those S items. Then, in each iteration, it sorts our vector using the std::sort() function. This is an nlog2n algorithm, which signifies our overall algorithm for this function runs in O((N^2)log2N) time.