Nathan Tjoar

005081232

Homework 4

2. For this, I got the compiler message that the comparison was between a const Coord and Coord, which under normal circumstances would work perfectly fine, but Coord is a class here. This means I am comparing two objects, in which the compiler is unsure on what to do as it sees multiple data points in the object. This, without an overload on the operator confuses the compiler, and the insert function and the findFirstAtLeast function are unable compare the two objects. As an added bonus, the #include header has no “” or <> following it, confusing the compiler as to what to include.

3b. We would not be able to solve this if this was only the one part overload of the listAll function as this would mean, that there would be no way to track and follow our tree data structure. As such, the output would not be able to do much without either outputting a backslash every time, or not outputting one at all.

4a. We know that the most internal loop containing the if then statements has each a time complexity of O(1), which when added gives us O(2). Then, moving to the for loop that encompasses it, we find that this loop has a time complexity of O(N). This gives us an overall total so far to be O(N). Looking at the rest of the functions in the next bounding loop, we get O(N + 2), as each comparison and function call takes about O(1), but this simplifies down to O(N). Then, moving to the for loop, we notice that it will give us another delay of O(N), which yields a temporary total of O(N^2). In the final loop, we have a single assignment which takes constant time, so we ignore it’s delay, as it won’t be statistically effective. In the for loop itself, we can see that the loop will take one more O(N) time, which gives us O(N^3). All other parts take constant time, so we finish here with total delay being O(N^3).

4b. Again here, we start with the function of the innermost loops, which we can see takes constant time, as it is comparison and assignment only. The loop itself takes about O(N) time. It’s next loop has similar functions to that that takes O(1) time, so we can disregard these. Moving to the loop, the function takes O(i) time, which gives us a current overall at O(Ni). Again, the other functions in the outermost loop take O(1) time, which is not very significant, so we move to the loop itself, which takes O(N) time. This yields an overall of O(N^(2)i). However, going back to i, we are able to assume that at worst, this algorithm would run O(N^(3)), as the best case for i is O(N).

5a. We know that the algorithm runs through comparisons of two sets to be O(N). So, going through from top to bottom, we can see that the first comparison will take O(N^2) time, as it is comparing three sets (set1 and result, set2 and result). Moving on, we see that the comparison between set1 and set2 and then the assignment takes again O(N^2) time. However, when adding, this yields O(2\*N^2) time, which boils down again to O(N^2). Proceeding on, we can see that the last comparison takes O(N) time and that its assignment as well. As O(N) is less significant that O(N^2), we can say that O(N) is only in the best case. Seeing the next iteration, we see that the algorithm runs an iteration O(N) times, each with incrementally larger function times inside the loop up until O(N). This gives us a worst case scenario of O(N^2) and a best case of O(N).

5b. In this, when copying all items, we can note that each of these for loops will take O(N) time. As we continue through the function, we note that the sorting algorithm will take O(N log N) timew, which replaces O(N). As we continue, we see a while loop with a time complexity of O(N-1) = O(N). Proceeding on, we find that the next for loop will take about O(N) time yet again, with its contents taking less significant times. As a result, we end with a best and worst case of O(N log N).

5c. We know that the algorithm runs through comparisons of two sets to be O(N). So, going through from top to bottom, we can see that the first comparison will take O(N) time, as it is comparing three sets (set1 and result, set2 and result). Moving on, we see that the comparison between set1 and set2 and then the assignment takes again O(N) time. However, when adding, this yields O(2\*N) time, which boils down again to O(N). We note that the for and while loops have each O(N) delay with each of their comparisons and functions taking up constant delay. This gives us O(N) delay overall.