2. The call to Set<Coord>::insert causes a compilation error because the template function for insert calls upon findFirstAtLeast which compares the values inserted to that already in the set (p->m\_value < value). Since there is no comparison implementation in the coord class(< , > or ==) , it will not be able to determine if you can insert that Coord into the set.

3a.The time complexity of the program is O(N3) because in the worst case, you have to iterate through 3 for-loops comparing all values of i,j, and k to N.

3b. The time complexity is still O(N3) because in the worst case, even though the second loop is limited by i, i could equal N. Thus making the entire complexity O(N3)

4a. O(N2) since it is the worst case time complexity. There are 3 conditions in which would result in different time complexities.

If the first condition is met (&result ==&set1) and (&result == &set2) -> return time complexity = O(1)

Second condition: else if(&result == &set2)-> sp = &set1, this results in copying data from &set1 to sp with the time complexity of:

Time complexity =O(N)

And the worse case where you have to set result = set1 O(N), then check if (&set1 == &set2) which will execute the for-loop that calls the get() function for each k from 0 to N-1

Time complexity = O(N2)

4b. Time complexity = O(N log N)

Copying values, deleting values, and copying unique values all have complexity of O(N) because you visit all values of N

The sorting algorithm requires O(N log N )

O(3N) + O(N log N) is simplified to O(N log N) because that the step that takes the longest and we simplify and throw away the lower-order terms and coefficients

4c. Time complexity = O(N)

This implementation only checks for singular node values O(1) of both set1 and set2. The function will be complete once all the values from set1 and set2 are complete. O(N) \* 2 \* O(1) = O(2N) simplified down = O(N)