2. The call to Map<Coord, int>:: insert calls the doInsertOrUpdate function, which then calls the findFirstAtLeast function, and then attempts to compare a Coord object to the data member m\_key using the Boolean operator ‘!=’. This causes at least one compilation error because the Map template function doesn’t explicitly define any comparisons of the Coord class, meaning the program doesn’t know how to compare two Coords using ‘!=’, as required in the findFirstAtLeast function.

3e. Considering test case 3 has to do with a vector of integers, it fails because it attempts to utilize an iterator after items have been added to the vector, leaving the old iterator pointing to a random spot in memory. This happens because when you add an item to a vector, the program may shuffle the address of the vector without the user knowing, rendering the old iterator useless as it doesn’t point to the right place any more.

4b. I could not solve this problem given the constraints in part a if we only had a one-parameter listAll because we were not allowed to use any global variables. In other words, there would be no way for the program to obtain the “path” string that is in the parameter of the two-parameter overloaded listAll. Because the one-parameter listAll would need the parameter to include the constant pointer to a class, following the instructions to implement a recursive function, we would have to find a different way to keep track of the path string other than inserting a string into the parameter. In order to do so, we would have to keep track of a global “path” string that can keep adding on to it as the program pleases, but the instructions forbid us of using any kind of global variable, which is why we could not solve this problem if we had to implement a one-parameter listAll as a recursive function.

5a. The time complexity of this algorithm is O(N^3) because the algorithm goes through three different for loops, each from 0 to N. For each for loop, the time complexity increases by a factor of N. Therefore, utilizing three for loops would be the same as multiplying N by N by N, which would result in O(N^3).

5b. The time complexity of this algorithm is still O(N^3) because we would still be considering the worst-case scenario which is still going through 3 different for loops. Even though the loop limit for the second for loop is i, it doesn’t make a difference in terms of big O because i still iterates a maximum amount of N times. Thus, the overall time complexity of this program would be no different from part a because the highest order term for both algorithms is still N^3.

6a. Its time complexity is O(N^2) because the first for loop iterates through m, which runs through all N elements. The second for loop is present within the get function, which in the worst case, also has to go through all N elementsagain as it insert’s i – 1’s key into each pair in i. There is no other process within the loop that outsizes the current time complexity and thus, the time complexity for this algorithm remains O(N^2).

6b. Its time complexity is O(N) because it is iterating through a linked list, which is one straight path as it runs through all N elements within the list. It is better than the implementation in part a because O(N) means that the algorithm only requires roughly N operations to process N items whereas, O(N^2), which is what we had in part a, means that the algorithm requires roughly N^2 operations to process N items. Ultimately, O(N^2) requires more operations than O(N) to process the same number of itiems, which is why part b’s time complexity is better than that of a.