

1.e.

* The reason why the test case 3 fails is because its iterator is invalidated. Unlike a list, reallocation or **push\_back** applied on a vector changes the address of the dynamic container ( supports the reason why vectors are good for randomly accessing elements of the container, but lists are useful in insertion/erasing at random points in a container).

3.

* The way which the single parameter insert works is comparing the elements of a sequence with the > operator. While this would work for a typename of int, the coord class which we have designed ourselves does not define this operator.

4.b

* If we were required to recursively implement the listAll function with only the one parameter function, this would not be possible because we need a way to hold the previous path of labels. With the two parameter listALl, path is updated and held as a parameter for each new subdomain which isn’t empty.

5.a

* The time complexity is O(N^3). We first can look at the inner-most for-loop, which is O(N). This for-loop is a nested loop within another for-loop comparing j N amount of times, which in total would then be O(N^2). Finally, there

is one more for-loop, comparing i N times, so then total time complexity results in O(N^3).

5.b

* The time complexity is still O(N^3)! The reasoning can be justified because we still have 3 for-loops. The inner-most for-loop doesn’t change, and we still compare k N amount of times ( making the inner for-loop order N). If we look at the for loop which this is directly nested in, we see that j is being compared i amount of times. Looking then at the final for-loop, we can analyze that in it’s worst scenario, the maximum of i is n, therefore the second for-loop totals to O(N^2) [almost like we compare j N amount of times]. The outer-most for loop is the same as previous [ compare i N amount of times] therefore the total time complexity results to O(N^3)

*\*NOTE\* if we wanted to actually analyze the second for-loop, we see that it is measured as N\*(N-1) / 2, but since we only care about high-order, the ½ can be neglected, resulting in order N^2*

6.a

* The overall time-complexity would be O(N^2). First, it is important to note that only the first for-loop will ever be called, because it is specified that n2 and n1 are equal. Thus, when we look at the time complexity of the code within the first for-loop, we see the functions insert() and get() being used. While insert() has a time complexity of O(1)( *because our sequence is a linked list* ), the get() function has a time complexity of O(N)*( because we would have to iterate through each value to reach the index and time complexity is formed based on the worst-possible scenario, which would be iterating through each element!*).

Although the for-loop calls get() twice, we would only add time complexities together if within the same loop, resulting in O(2N), where 2 is a constant, so it simplifies to O(N) within the loop. Then, the actual for-loop is repeated N times, so the overall time complexity is O(N^2).

6.b

* The time complexity for this implementation of a member-function interleave is of time complexity O(N), which is better than the previous implementation. The reason for this is because our for-loop is iterating with a pointer ( increasing that ), rather than an index k. A sequence cannot be accessed like an array, so the original impementation would require the use of get(), which is O(N). Our new implementation of interleave uses insertBefore() , which is of O(1), and since the for-loop is only of O(N), the entire interleave function is of O(N), assuming seq1 and seq2 are the same size as specified.