**Homework 4**

**1e.** Test case 3 fails due to the use of an iterator to keep track of a position in a vector while pushing new elements onto the vector. Since the addresses can be shuffled during this process, the iterator becomes in invalidated.

**3.** The one-argument form of Sequence<Coord>::insert causes an error because the function uses comparisons of the item types in its algorithm. However, the Coord class didn’t define an overload >= operator, so C++ can’t perform the comparison.

**4b.** Without the extra parameter, there wouldn’t be a way to keep track of the pathway names that came before the current recursion call. We would only have access to the current menus’ name.

**5a.** The complexity of the algorithm is O(N3). The algorithm consists of 3 loops that are nested one within another. The outermost loop runs N times, the middle loop runs N times, and the innermost runs N times. As a result, the inner code runs for a total of N\*N\*N = N3.

**5b.** Although the algorithm has been adjusted so that the middle loop runs ‘i’ times instead of N times, we calculate the complexity considering **worst** case scenario. Meaning that under the worst circumstances, the middle loop will run for a total of N times as before. The outer and inner loops run for N times as they did before. So, under worst case scenario, total: N\*N\*N = 3.

Thus the complexity is O(N3).

**6a.** In order to properly analyze the complexity of concatReverse(), we must determine the complexity of the functions that it calls since they will add complexity to the overall function call. There are two loops which run for a max amount of N times each. Then, each loop calls the following functions which then in turn call other functions. All the functions which are involved in the process are shown below with their respective complexities in terms of how many nodes are accessed:

* .get() 🡪 O(N)
* .insert() 🡪 O(N)
* nodeAtPos() 🡪 O(N)
* insertBefore() 🡪 O(1)
* .swap() 🡪 O(1)

After calculating the complexities, each loop in concatReverse() has a complexity of N2. Since we have two loops, that comes out to 2N2 or just an overall complexity of O(N2).

**6b.** The complexity is O(N), which is much better than before. The time complexity was reduced because the function didn’t rely on calling other functions which added to the complexity. The function did call:

* insertBefore() 🡪 O(1)
* .swap() 🡪 O(1)

Which didn’t impact the complexity significantly.