hw

1e. Make sure you understand why the code below passes the first two tests but fails the third. Draw pictures if necessary. Explain in a sentence or two what happens during the execution of test case 3 that eventually leads to test case 3 failing.

When v2.push\_back(MAGIC) is called 5 times, the vector resizes and allocates new memory to accommodate the additional elements. Doing so invalidates the iterator and leads to the iterator exhibiting undefined behavior.

3. Explain in a sentence or two why the call to the one-argument form of Sequence<Coord>::insert causes at least one compilation error. (Notice that the call to the one-argument form of Sequence<int>::insert is fine, as is the call to the two-argument form of Sequence<Coord>::insert.) Don't just transcribe a compiler error message; your answer must indicate you understand the ultimate root cause of the problem and why that is connected to the call to Sequence<Coord>::insert.

For the insert function to work correctly, the ItemType type should support comparison operators (> in this case) to determine the correct position for insertion. If Coord does not provide the necessary comparison operators, you will encounter a compilation error.

4b. We introduced the two-parameter overload of listAll. Why could you not solve this problem given the constraints in part a if we had only a one-parameter listAll, and you had to implement it as the recursive function?

The one-parameter listAll function lacks the ability to keep track of the path to each menu item in the tree. It only operates on a single node at a time without any knowledge of its position within the tree or its parent nodes. To construct the complete path to each menu item in the tree, we need to maintain and update the path as we traverse the tree recursively.

5a. What is the time complexity of this algorithm, in terms of the number of basic operations (e.g., additions, assignments, comparisons) performed: Is it O(N), O(N log N), or what? Why? (Note: In this homework, whenever we ask for the time complexity, we care only about the high order term, so don't give us answers like O(N3+4N2).)

The time complexity of this algorithm is O(N^3), since it has three nested for loops that iterate N times. Other operations such as assignments, comparisons, and logical operations are considered constant.

5b. What is the time complexity of this algorithm? Why?

The time complexity of this algorithm is also O(N^3) because the maximum j value for the second for loop is N, so similar to the last problem, it has three nested for loops that iterate N times. Other operations such as assignments, comparisons, and logical operations are considered constant.

6a. Assume that seq1, seq2, and the old value of result each have N elements. In terms of the number of linked list nodes visited during the execution of this function, what is its time complexity? Why?

Within the for loop that iterates N times, the function get is called, which calls the helper function nodeAtPos. nodeatPos runs through its for loop N times (since the coefficient doesn’t matter). The second for loop is disregarded because the sequences both have N elements. Thus, the concatReverse function has two nested for loops, which makes the time complexity of this algorithm O(N^2).

6b. Assume that seq1, seq2, and the old value of \*this each have about N elements. In terms of the number of linked list nodes visited during the execution of this function, what is its time complexity? Why? Is it the same, better, or worse, than the implementation in part a?

Both for loops have about N elements, so the second for loop is disregarded yet again, leaving this algorithm with a time complexity of O(N). This means this is better than the implementation in part a since it runs faster.