**Question 1e**

Because the original vector size was 5 and the vector size is increased during the iteration to add another element, a larger space in memory is needed to store the additional element. When the push\_back function is called, the elements in the original memory location were copied over to the new memory location and the old memory location was deleted, but the iterator p is still pointing to the old place in memory, which no longer contains anything of use.

**Question 3**

The one argument form of insert will try to insert the ItemType into the linked list in the correct ordered position. However, when Coord is the ItemType, there is no way to compare whether one Coord is greater than, less than, or equal to another Coord so you can't compare the values of Coord to determine where it should be inserted, which is why the one argument form doesn't work.

**Question 4b**

The two parameter overload of listAll allows you to keep track of the current path of the menus. If you only had a one-parameter listAll, there's no effective way to keep track of different branching points of the path.

**Question 5a**

O(N^3) because there is a loop inside of a loop inside of a loop. Basically 3 nested for loops with each loop running N times. So the first for loop runs N times, and for each item you need to run that N times and the items within that need to run N times and you repeat for all N items in the outermost loop. The total number of operations will be (approximately) N\*N\*N or N^3

**Question 5b**

O(N^3). There are still 3 nested for loops with each loop potentially running up to N times. Even though the 2nd nested for loop runs less times, as N becomes very large that doesn't become very significant because the time complexity will still be dominated by the N^3 term since it'll still be running through the 3 nested for loops.

**Question 6a**

O(N^2). There are 2 for loops, where each one goes through one of the sequences, so running the 2 for loops would have O(N) each and O(2N) together, but that's basically O(N) because we don't care about any constant multipliers in this case. Then within each for loop, we have to get the value of the node at a certain position, but you have to traverse through the linked list until you're at the position you want then get the value of that node. So the time complexity would be O(N) for retrieving the value. Then you want to insert the new node into the result sequence at a certain position, so you would traverse through the result sequence until you're at the position you want, then you'll insert a new node, so the time complexity will be O(N) for that. So, you have O(N) and O(N) within each for loop which is O(2N) combined, but that simplifies to O(N) because we ignore constant multipliers. Thus, you have O(N) within O(N) which is O(N^2) for the entire function.

**Question 6b**

O(N). This implementation is better than the implementation in part a because the time complexity grows linearly compared to growing exponentially at a factor of 2. Here, when you insert a new node, you'll have to go through all N elements in the seq1 linked list and copy the nodes into the result sequence(in reverse order), so the time complexity here is O(N). Even though you're traversing through both sequences and copying over the elements and we'll visit 2N nodes, the time complexity overall will still be O(N) since any constant multipliers and constants are disregarded. Also, swapping the results is O(1) but this doesn't impact the overall time complexity, which remains O(N).