**2.** Consider this program:

#include "Sequence.h" // class template from problem 1

class Complex

{

public:

Complex(double r = 0, double i = 0)

: m\_real(r), m\_imag(i)

{}

double real() const { return m\_real; }

double imag() const { return m\_imag; }

private:

double m\_real;

double m\_imag;

};

int main()

{

Sequence<int> si;

si.insert(50); // OK

Sequence<Complex> sc;

sc.insert(0, Complex(50,20)); // OK

sc.insert(Complex(40,10)); // error!

}

The error with the compiler error in the insert function that only takes one parameter lies with the fact that the function implementation compares the values of the Complex class. However, there is no overloaded “>” that would allow for this comparison and thus it is unable to complete the task in order to insert the value in order.

**3b.** 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?

This would not work because the recursive function would only be able to read data values of its menuItem. The Menus above it would not be able to be passed down to be read.

**4a.** This problem is O(N^3) because we have a two-dimensional array of N steps. We transverse through the array with for loops. There are 3 for loops that all take 3 steps and they are all nested inside one another which means that they can be run for each loop it is in. This means N\*N\*N steps or N^3 steps.

**b.** The first for loop is still to the order of N steps. However, inside of it, there are two nested loops, one of I steps and one of N steps. Though the I steps are less than the N steps, at the very last loop, it will become N steps (loops from 1 to N steps). Thus we can say that the time complexity of this algorithm is O(N^3).

**5a.** There are two for loops in this function but they are not nested inside one another. One runs through the number of items that is the minimum number between the two sequences while the other loop concatenates the remaining values of the sequence on. Since this only runs through the array N (largest n) total times, it is O(N) time complexity.

**b.** For this problem, there are two for loops. However, each loop runs through the entirety of the Sequence, stopping only when both sequences are m\_head. Thus, because each loop separately runs through N, the time complexity is O(N). It is a little less efficient than the previous implementation in part a because instead of N, this one is 2N due to running through the entirety of N twice (not within one another which would give N^2).