Shaan Mathur

Smallberg

CS32

May 24, 2016

Homework 4 Questions

2. The reason Set<Bottle>::insert(…) causes an issue is because the class Bottle does not have an operator overload for the != logical operator (unlike for int/string). The compiler does not know what it means for two Bottles to not be equal to one another, and the insert(…) function leads to a series of function calls that winds up trying to invoke the != operator on the Bottle class in the Set<ItemType>::find(…) function.

3. b. The problem would have been impossible to solve without a helper/auxiliary function for listAll(const File\* f). This is because as you are traversing down the tree, the function is supposed to somehow remember the names of the Files/nodes from previous calls so that it can print out the path properly. This would prove impossible since these function calls have their own local variables, so the names are unknown to each other in different calls of the function. The solution was to include another parameter so that each function call can communicate with the other calls and let them know what their path should be.

4. a. The time complexity for this algorithm is **O(N3)**.This is because we have three nested for loops each executing N times, with each loop not doing anything significant to cut down the work of the problem. At most, we skip some iterations where the parameters are equal to one another. But this in comparison to the rest of what has to be searched through is insignificant, thus making it O(N3).

b. This algorithm is still **O(N3)**, just with a better constant of proportionality. Ignoring the third inner loop, the two outer for loops will definitely execute N2 times because of the following mathematical proof:

Thus the growth pattern of the two outer for loops is still N2. Because the inner for loop still runs N times due to the reasoning made in part a, the growth of the entire algorithm is O(N3) still.

5. a. The algorithm, in the worst case, has three different sets of size N. Assigning a set to result is actually linear time because the destructor must destroy all N nodes so it can be replaced with other nodes. The last for loop is really the part of the algorithm that determines the time complexity, since it will have the highest growth relative to everything else. Here we (attempt to) insert every element from the other set into result, where set has N elements. Since the get function has to walk through the entire linked list, that operation also takes O(N). An O(N) operation inside a for loop that runs N times results in **O(N2).** (Note: If we were talking more generally, not just linked list nodes, it is possible this growth could be higher because ItemType’s constructor may not be a simple O(1) algorithm).

b. The time complexity for this algorithm is **O(NlogN)**. This is because out of all the separate operations the algorithm is doing, this one has the largest growth factor (everything else was O(N) or O(1), paling in comparison to the O(NlogN) sort algorithm that was invoked).