Attiano Purpura-Pontoniere

504-847-318

CS 32

Homework 4

Problem 2:

When the complier creates a Coord version of the template function insert(value) and template class Set, it creates an exact copy of the insert function and Set class code and replaces all the template typename declarations with Coord declarations. In the Set class there is a nested structure node, which contains a template typename object that gets replaced with coord, so when in the insert(value) function, when the pointer to the object in the node, p->m_value, gets called it is calling a Coord object. The insert(value) function compares that Coord object with the value passed in by the call insert(value), and does so with the "==" operator, the Coord class itself provides no overloading of the "==" operator so the complier doesn't know how to compare two Coords and therefore doesn't know how to perform the insert function on a Set of Coords.

Problem 4 b:

Since we have to implement the function list all recursively, we need to be able to pass into the function all past labels in the tree, so that once it reaches a leaf node it can output the entire path leading up to the leaf node. Passing in the old path allowed the recursive calls to remember everything that happened up until that depth, or to remember its depth. The recursive function needed to be able to recall what had lead it to the current (potentially) recursive call.

Problem 5a:

$O(n^3)$, because the program contains a triple nested for loop, each of which does constant time operation N times, resulting in O(n) complexity for each loop or $O(n*n*n)=O(n^3)$

Problem 5b:

$O(n^3)$ but with a better constant of proportionality than problem 5a. This is because the program has a triple-nested for loop, the innermost of which does a constant operation N times, the second-layer-for loop does the innermost-for loop and constant time operation first 0 times, then 1 time, then 2 times, then 3 times…. Till N-1 times; therefore the 2 inner-most loops happen a total of (THE SUM OF ALL THE NUMBERS FROM 0 TO N-1) times. The sum of all the numbers from 0 to N-1 is ~ (N^2 + N)/2. Therefore the two inner most loops happen (N^2 + N)/2 times and do constant time operation N times in the loops, therefore the complexity is: $O(((n^2+n)/2)*n) \Rightarrow O((n^3 + n^2)/2) \Rightarrow O(n^3/2)$

$\Rightarrow O(n^3)$

Problem 6a:

O(n^2), because everything up until the for loop is constant time, and the for loop executes N times, and in the for loop it does a constant time operation, and “insert()” and “get()”

Insert: calls “findfirstatleast()” and “insertbefore()”

Insertbefore() is constant time, but findfirstatleast() visits at worst N nodes in the linked list and so has complexity N, so the complexity of the insert call is N.

Get: is constant time besides the two for loops, and each for loop has constant time operations which would at worst would execute N/2 times, so “get()” has at worst complexity N.

So over all the complexity of the body of the original for loop in unite is complexity N (since both get and insert at complexity N and they only happen once on each iteration of the for loop), and it executes N times, so the overall complexity is O(N*N + N*N + CONSTANT) = O(N^2)

Problem 6b:

O(n*log(n)), because since doErase() and insertBefore() are constant time, all the loops (the first two for loops, the while loop, and the last for loop) happen N times and have constant time complexity each iteration of the loop. However the sort function has complexity Nlog(N) which is larger than N, and so the overall complexity is O(Nlog(N) + N + N + N + N) = O(N*log(N))

Problem 6c:

O(n), because everything up to the while loop is constant time and the body of the while loop is constant time (insertBefore() is constant time) and at worst the body of the while loop happens N times, and the following for loop has a body that is also constant time, and also happens at worst N times, so the overall complexity is O(constant + N + N) = O(N)