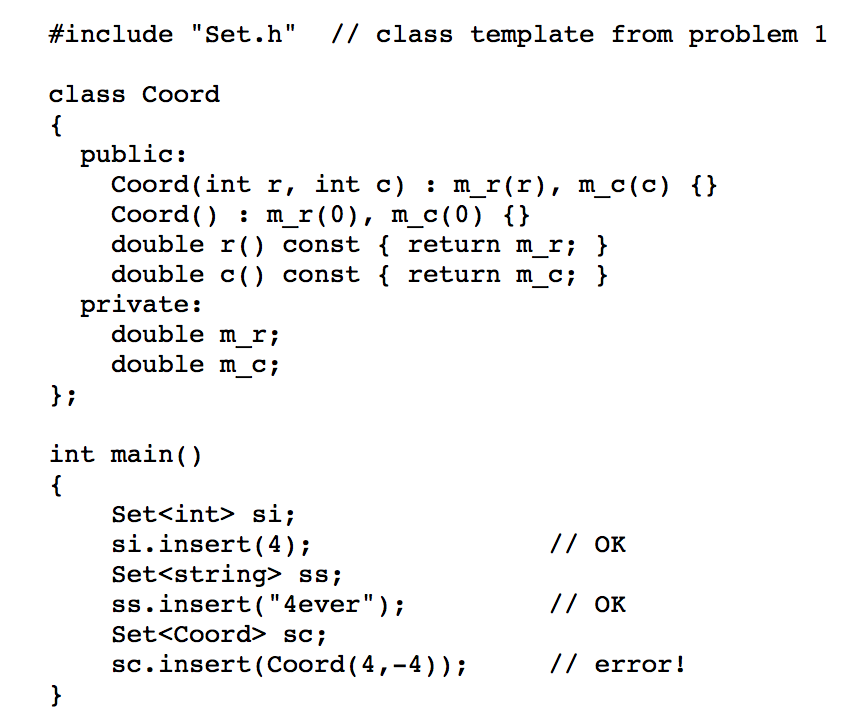
CS32: Homework 4

2. Explain in a sentence or two why the call to Set<Coord>::insert causes at least one compilation error. (Notice that the calls to Set<int>::insert and Set<string>::insert are fine.) 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 Set<Coord>::insert.



The call to Set<Coord>::insert causes a compilation error because Coord is not a built-in type for which a compiler is familiar with how to treat comparison of Coord objects. The insert function itself calls the function findFirstAtLeast which utilizes binary comparison to execute, which is a feature that for class Coord had not been defined. Conversely, this is why the call to insert is fine for built-in types like int and string because binary comparison is built into the compiler for these types, and thereby have well-defined behavior.

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?

Without the string path parameter, we would lose the previous step in the tree. Therefore,

by attempting to implement it recursively, we would lose this pertinent data as it would override what it generated previously. Because this function’s use depends on its data produced iteratively, the use of one parameter that does not hold the previous state would not be effective.

5a. const int N = *some value*;

bool hasCommunicatedWith[N][N];

...

int numIntermediaries[N][N];

for (int i = 0; i < N; i++)

{

numIntermediaries[i][i] = -1; // the concept of intermediary

// makes no sense in this case

for (int j = 0; j < N; j++)

{

if (i == j)

continue;

numIntermediaries[i][j] = 0;

for (int k = 0; k < N; k++)

{

if (k == i || k == j)

continue;

if (hasCommunicatedWith[i][k] && hasCommunicatedWith[k][j])

numIntermediaries[i][j]++;

}

}

}

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(N2+4N).)

This algorithm is O(N^3), as each for loop is executed N times, with 3 loops in total. On

its own, the most outer loop has a O(N) time complexity, but with 2 nested loops, it amounts to O(N^3).

5b. const int N = some value;

bool hasCommunicatedWith[N][N];

...

int numIntermediaries[N][N];

for (int i = 0; i < N; i++)

{

numIntermediaries[i][i] = -1; // the concept of intermediary

// makes no sense in this case

for (int j = 0; j < **i**; j++) **// loop limit is now i, not N**

{

numIntermediaries[i][j] = 0;

for (int k = 0; k < N; k++)

{

if (k == i || k == j)

continue;

if (hasCommunicatedWith[i][k] && hasCommunicatedWith[k][j])

numIntermediaries[i][j]++;

}

**numIntermediaries[j][i] = numIntermediaries[i][j];**

}

}

What is the time complexity of this algorithm? Why?

Although the worst the case in the second loop is N-1, now that the condition for its execution has been changed, it remains O(N^3). This is because the second inner loop still has a complexity of O(N^2) despite the change of condition. And as stated before, the outer loop is O(N) on its own, therefore still amounting to O(N^3).

6a. void unite(const Set& set1, const Set& set2, Set& result)

{

const Set\* sp = &set2;

if (&result == &set1)

{

if (&result == &set2)

return;

}

else if (&result == &set2)

sp = &set1;

else

{

result = set1;

if (&set1 == &set2)

return;

}

for (int k = 0; k < sp->size(); k++)

{

ItemType v;

sp->get(k, v);

result.insert(v);

}

}

Assume that set1, set2, 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 worst case time complexity? Why?

This algorithm is O(N^2), as the overloaded operator in the conditional statements will be executed in the worst-case scenario. Iterating through a linked list is O(N) time, and the loop also has O(N) time, therefore the code within the loop generating a time complexity of O(2N), and combined with the entire loop, this code is O(2N^2). However, for our purposes the coefficient does not matter, and thus the time complexity of this unite function is O(N^2).

6b. Here is an implementation of a related member function. The call

s3.unite(s1,s2);

sets s3 to the set union of s1 and s2. The implementation is

void Set::unite(const Set& set1, const Set& set2)

{

vector<ItemType> v;

// copy all items into v;

for (Node\* p1 = set1.m\_head->m\_next; p1 != set1.m\_head; p1 = p1->m\_next)

v.push\_back(p1->m\_value);

for (Node\* p2 = set2.m\_head->m\_next; p2 != set2.m\_head; p2 = p2->m\_next)

v.push\_back(p2->m\_value);

// sort v using an O(N log N) algorithm

sort(v.begin(), v.end());

// delete result nodes (other than the dummy node)

while (m\_head->next != m\_head)

doErase(m\_head->m\_next);

// copy unique items from v into result

for (size\_t k = 0; k < v.size(); k++)

{

if (k == 0 || v[k] != v[k-1]) // add non-duplicates

insertBefore(m\_head, v[k]);

}

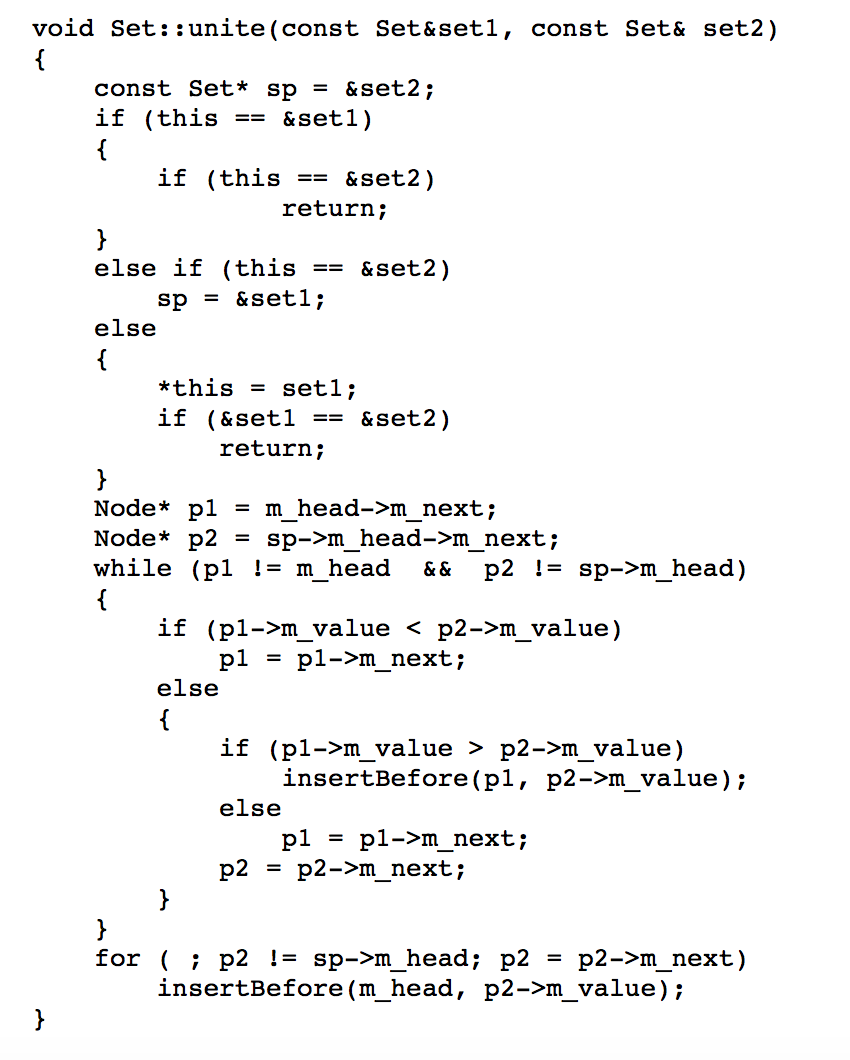
// v is destroyed when function returns

}

Assume that set1, set2, and the old value of \*this each have N elements. In terms of the number of ItemType objects visited (either in linked list nodes or the vector) during the execution of this function, what is its time complexity? Why?

The loops account for O(3N), and the sort function is O(NlogN). But since we just consider the time complexity of the function with that of the highest degree, this unite algorithm has a time complexity of O(NlogN).

6c.



The time complexity for this algorithm is O(N). The loops while iterating through each

node amount to O(N) – and the other code is simply constant time. Therefore, the dominating term in this unite function is O(N), and the time complexity is O(N).