Avery Einhorn

UID: 405-115-287

CS 32 Homework 4 – 5/26/19

**2.**  The one-argument form of the insert function requires that the ‘>’ operator compare value against a value in the sequence, but *the Coord class has no way of accomplishing the > operator.* The two-argument insert function does not use the ‘>’ operator, and the one-argument insert function with an int type sequence has a defined ‘>’ operator, so they run into no problems.

**4b.** A one-parameter implementation of listAll() would not be able to carry down the label values from higher-order nodes to the leaf nodes, so when trying to display each unique list of labels, we would only be able to display higher order label values once. What I mean is that we could print:  
cs.  
ee.  
math.ucla.  
math.  
cs.caltech.edu.  
i.e. the higher order domain names like edu and ucla could only print once we were totally done printing the subdomains, so we would only get a few *complete* listings.

**5a.** The algorithm is **ON3** complex. It has three nested for loops, the innermost of which is comparing values created in that for loop to values created within each of the other two, creating a cubic relationship. Comments with locations of non-trivial operations are below.

const int N = some value;

bool isFriend[N][N];

...

int numMutualFriends[N][N];

for (int i = 0; i < N; i++) // N comparisons, 2N assignments

{

numMutualFriends[i][i] = -1; // the concept of mutual friend

// makes no sense in this case

for (int j = 0; j < N; j++) // N^2 comparisons, assignments

{

if (i == j) // N^2 comparisons

continue;

numMutualFriends[i][j] = 0; //N^2 comparisons

for (int k = 0; k < N; k++) // N^3 comparisons, assignments

{

if (k == i || k == j) //2N^3 comparisons

continue;

if (isFriend[i][k] && isFriend[k][j])//2N^3 comparisons

numMutualFriends[i][j]++;

}

}

}

**5b.** The improved algorithm is **ON3**. The second nested for loop only goes up to the value of the first loop, but the leading term in the complexity is still N2 when you multiply it out. So it’s the same as the answer for 5a.   
 const int N = some value;

bool isFriend[N][N];

...

int numMutualFriends[N][N];

for (int i = 0; i < N; i++) // N operations and comparisons

{

numMutualFriends[i][i] = -1; // the concept of mutual friend

// makes no sense in this case

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

{ // *Is this NlogN?? no*

numMutualFriends[i][j] = 0;

for (int k = 0; k < N; k++) // Within the last thing, add on an N

{

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

continue;

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

numMutualFriends[i][j]++;

}

**numMutualFriends[j][i] = numMutualFriends[i][j];**

}

}

**6a.** Ignoring the calls to *get()* and *insert()*, the function has **ON** time complexity, but taking into account the fact that this calls get() and insert() (which has **O** complexity itself, if we assume that iterating through nodes and ignoring ItemType values within nodes, as is done by the nodeAtPos function, is inconsequential to time complexity), it has **ON.   
*If*** we do indeed care about the complexity of iterating through nodes, and not just visiting ItemTypes, then the first for loop has complexity **ON2.** (because ON\*ON)

void interleave(const Sequence& seq1, const Sequence& seq2, Sequence& result)

{

Sequence res;

int n1 = seq1.size();

int n2 = seq2.size();

int nmin = (n1 < n2 ? n1 : n2); // assume nmin = to N

int resultPos = 0;

for (int k = 0; k < nmin; k++) //loop N times, accessing 2N or **O**N times

{

ItemType v;

seq1.get(k, v); // N complexity respect to nodes not   
 // ItemType values though

res.insert(resultPos, v); // complexity with looping nodes

resultPos++;

seq2.get(k, v);

res.insert(resultPos, v);

resultPos++;

}

const Sequence& s = (n1 > nmin ? seq1 : seq2);

int n = (n1 > nmin ? n1 : n2);

for (int k = nmin ; k < n; k++)

{

ItemType v;

s.get(k, v);

res.insert(resultPos, v);

resultPos++;

}

result.swap(res);

}

**6b.** The new and improved function has complexity **ON** in terms of ItemTypes (with the exclusion of visited nodes). And it also has complexity **ON** with respect to nodes if we care. Because it calls out ItemTypes within for loops that run N times, and the insertBefore function has time complexity **O** (a constant) because it just runs its lines of code once (no loops).