**Homework 4**

|  |  |  |
| --- | --- | --- |
| NUMBER | QUESTION | ANSWER |
| 2. | Explain in a sentence or two why the call to Map<Coord, int>::insert causes at least one compilation error. | This will cause a compilation error because, as we talked about in class, you need to define your own operators when you use them in maps. However, in this case, since we are mainly using this for strings, ints, and doubles, we just are using the built in, default operator for comparisons (ex. ==, !=). When you’re comparing two objects of type Coord, however, it doesn’t know how to handle this. How do you compare a Coord? It isn’t defined so it won’t work. |
| 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? | You wouldn’t be able to only use the function that has one parameter because you need something to keep track of what nodes you have visited because when you get down to the bottom, you need to be able to print everything from the top to the bottom. You need the second parameter path, to keep track of this (or you would be able to reach the bottom node and not know what to print before that). |
| 4A. | 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).) | The time complexity of this is O(N^3). This is because each for loop has a time complexity of N. Since it is nested, you need to multiply it by a N every time you loop it until N. All of the operations of addition or accessing a specific value (for example like [i],[j] of the array), it’s linear since you’re not asking to loop through until that point, you just go and look specifically at that point in your memory. The work is shown below. |
| 4B. | What is the time complexity of this algorithm? Why? | This also has a time complexity of O(N^3). The main thing that changed is that you’re not running the second loop all the way to N this time. Instead you run it to i, the value of the first loop. By itself, this would have a complexity of ½ N. However, as we discussed in class, this is quite negligible in the bigger scheme when you have large numbers, so it just equals itself back out to N. The work is shown below. |
| 5. | Assume that m1, m2, 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 time complexity? Why? | The time complexity is O(N^2). This is because you only have one loop that will iterate through everything and have a time complexity of N. Then, your functions will have to do intricate things like swapping pointers, and swapping maps. Each has a time complexity of N and you add these together. However because we are using ‘Big-O’, we disregard the addition of each of them and just keep it at N^2. |

WORK - QUESTION 4A

const int N = *some value*;

    bool hasCommunicatedWith[N][N];

    ...

    int numIntermediaries[N][N];

    for (int i = 0; i < N; i++) ---> this has a complexity of O(N)

    {

       numIntermediaries[i][i] = -1;  ---> linear

   for (int j = 0; j < N; j++) ---> this has a complexity of O(N)

       {

           if (i == j)---> linear

            continue;

           numIntermediaries[i][j] = 0;

           for (int k = 0; k < N; k++) ---> this has a complexity of O(N)

           {

            if (k == i  || k == j) ---> linear

                continue;

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

               numIntermediaries[i][j]++; ---> linear

      }

   }

}

WORK - QUESTION 4B

const int N = *some value*;  
 bool hasCommunicatedWith[N][N];  
 ...  
 int numIntermediaries[N][N];  
 for (int i = 0; i < N; i++) ---> this has time complexity of O(N)  
 {  
    numIntermediaries[i][i] = -1;  ---> linear  
    for (int j = 0; j < **i**; j++) ---> this has time complexity of O(½\*N) **// loop limit is now i, not N**  
    {  
        numIntermediaries[i][j] = 0;  
        for (int k = 0; k < N; k++)---> this has time complexity of O(N)  
        {  
            if (k == i ||  k == j) --->linear  
                continue;  
    if (hasCommunicatedWith[i][k]  && hasCommunicatedWith[k][j])  
         numIntermediaries[i][j]++; --->linear  
        }  
        **numIntermediaries[j][i] = numIntermediaries[i][j]; --->linear**  
    }  
 }

WORK - QUESTION 5

bool combine(const Map& m1, const Map& m2, Map& result)  
{  
      // For better performance, the bigger map should be the basis for  
      // the result, and we should iterate over the elements of the  
      // smaller one, adjusting the result as required.  
  
    const Map\* bigger;  
    const Map\* smaller;  
    if (m1.size() >= m2.size()) -->LINEAR  
    {  
        bigger = &m1;  
        smaller = &m2;  
    }  
    else  
    {  
        bigger = &m2;  
        smaller = &m1;  
    }  
  
      // Guard against the case that result is an alias for m1 or m2  
      // (i.e., that result is a reference to the same map that m1 or m2  
      // refers to) by building the answer in a local variable res.  When  
      // done, swap res with result; the old value of result (now in res) will  
      // be destroyed when res is destroyed.  
  
    bool status = true;  
    Map res(\*bigger);               // res starts as a copy of the bigger map  
    for (int n = 0; n < smaller->size(); n++) -->COMPLEXITY OF N   
    {  
        KeyType k;-->LINEAR   
        ValueType vsmall; -->LINEAR   
        smaller->get(n, k, vsmall); -->COMPLEXITY OF N   
        ValueType vbig; -->LINEAR   
        if (!res.get(k, vbig))      // key in smaller doesn't appear in bigger  
            res.insert(k, vsmall);  // so add it to res  
        else if (vbig != vsmall)    // same key, different value  
        {                    // so pair shouldn't be in res  
            res.erase(k);        
            status = false;  
        }  
    }  
    result.swap(res); -->  
    return status; -->LINEAR   
}