COMP161 — Lab 9

Spring 2018

For this lab you'll take a careful look at *insert*, the core procedure used by insertion sort. First you'll be stepping through the ecursive and iterative versions of insert as presented in lecture notes 14. When that's done, you'll take a stab at inverting the logic of these procedures.

Lab 9

Get the code for lecture notes 14 from */home/comp161/sp18*. Within that library is two implementations of insert: one iterative and one recursive. You'll be asked trace some very specific calls to both functions and then come up with some examples of your own to trace. Instructions for doing traces are given in the subsections below. We'll be using a tabular style of tracing that we started using in class this week.

1. Trace the following iterative insert:

```
1 std::vector<int> ex({2,4,6,8,10,12,5});
2 ln14::iter::insert(ex,0,ex.size()-1);
```

2. Come up with an example that will have a trace table that has exactly 9 rows. Trace it.

3. Answer the following: For a vector of size n what is the minimum and maximum number of rows you can have in its trace table for this iterative insert?

4. Trace the following recursive insert:

```
1 std::vector<int> ex({7,2,4,6,8,10,12});
2 ln14::recur::insert(ex,1,ex.size());
```

5. Come up with an example that will have a trace table that has exactly 7 rows but with a vector that has a size of 10. Trace it.

6. Answer the following: For a vector of size n what is the minimum and maximum number of rows you can have in its trace table for this iterative insert?

7. Both sort procedures work by repeatedly calling insert in such a way that the vector gets sorted. Test your understanding of insert and running program traces by doing a tabular trace of both the iterative sort procedures for the following vector.

```
std::vector<int> ex({3,1,7,0,2})
```

In these traces treat insert like a black box. Don't trace it within the trace of the sort. This means your iterative sort trace should have one row for each repetition of the sort loop and not one for each sort loop repetition and one for each insert repetition within each sort repetition. For the recursive sort you must handle the fact that it is not tail recursive. To do this add a row to the table twice for each call to sort: once when it's called and once when it gets returned to, i.e. after it's finished recursively sorting "the rest" of the vector. The order of the rows should reflect the order in which events occurs in the program.

Tracing ln14::iter::insert

When tracing iteration we need to track the state that controls the loop and whatever state is accumulating the result. In the case of our insert procedure this means the value of *i* and the current contents of the vector *data*. A natural time to record this information is at the moment prior to each check of the loop's continuation condition as this provides a snap shot of key program state before iteration, during each step iteration, and when the iteration is complete.

We can record this in tabular format as follows. Label the table with the procedure call. In the first column list the value of *i* and in the second column list the value of *data*. The array value should be written just like it would in C++. For example, if we were tracing something like:

```
std::vector<int> ex({3,5,7,9,4});
ln14::iter::insert(ex,0,ex.size()-1);
```

Then we might start out like:

Trace of *ln14::iter::insert(std::vector<int>({3,5,7,9,4}),0,4)*

i	data
3	std::vector<int>({3,5,7,9,4})

Tracing ln14::recur::insert

When tracking iteration we could simply track key state at the the moment of testing the loop continuation. Tracing a recursive procedure can take a bit more care because we must maintain a clear

picture of where we are within the stack of recursive procedure calls. This can mean tracking state whenever the function context changes, i.e. when a function is called and when it is returned to.

Thankfully, the insert procedure is what's known as TAIL RECURSIVE, meaning the last thing it does is make a recursive call[1]. We'll only need to worry about taking snapshots whenever a recursive procedure call is made.

[1] This is in contrast to the max procedure we saw last week which completed some computation after the recursive call returned.

Once again, we can use a tabular format to trace our computation. This time each row represents a recursive procedure call. This is tracked by the values of it's arguments: *fst*, *lst*, and *data*. We will not start a table with each new call. For example, if we made the following call to insert:

```
1 std::vector<int> ex({3,5,7,9,4});
2 ln14::recur::insert(ex,1,ex.size());
```

Then we might begin our trace like this:

Trace of *ln14::recur::insert(std::vector<int>({10,3,5,7,9}),1,5)*

fst	lst	data
1	5	std::vector<int>({10,3,5,7,9})
2	5	std::vector<int>({3,10,5,7,9})