Timing Tests, Reading

Proportionality formulae (that is, **f(n)** as a function on **n**) tell us a lot about *how* solutions can be expected to scale, but they don't really offer anything we can use to **predict runtime**. For example, if we ran our DVC schedule history program with 700 records instead of 70,000 and timed it with a stop watch, knowing the n^2 relationship, could we predicte the runtime for 70,000? Probably so. But we need a way to do so without using an actual "stop watch". Here's how we can include a **virtual stop watch** in our programs:

```
#include <ctime>
...
int main()
{
  ...
  clock_t startTime = clock( ); // start the virtual stop watch...
  //do something here...
  clock_t endTime = clock( ); // ...stop the virtual stop watch
  double elapsedSeconds = (double)(clock() - startTime) / CLOCKS_PER_SEC;
  ...
```

This computes the number of seconds to run. The function **clock( )** and the constant **CLOCKS_PER_SEC** come from the **ctime** library. (Sorry, but this may not work for some versions of Linux!)

But don't use a **cin.get( )** in what you're timing, because that pauses the timer on some systems!

Applying Timing To The DVC Schedule History

So if we go back to our original solution in Module 5 and stop at 700 records and get it to run in 0.01 seconds, let's see if we can predict the runtime for 70,000. n would be 100 times larger. n^2 would then be 10,000. Multiply that by 0.01 and we get about a minute and a half! So you can see the effect of scaling and the importance of knowing it!