Keeping track of CPU time in C++

There are several libraries that allow for the tracking of time from within a C++ program. For today, we will rely on the following:

```
#include <time.h>
```

This library defines a new data type `clock_t` that is an integer-like value that represents a quantity of clock ticks (similar to how `size_t` is a datatype for measuring the size of a container).

It then provides a function `clock()` that returns a `clock_t` instance describing the number of clock ticks devoted to the execution of the current program. The number of clock ticks per second depends upon the machine architecture; that value can be found using the defined `CLOCKS_PER_SEC`.

To measure the time used for a specific portion of your program, the typical approach used is to record the starting time, then do the work, then record the stopping time, and then calculate the difference between the start and stop times. For example

```
clock_t begin, end;
begin = clock();    // Go!
...                 // Do something
end = clock();      // Stop!
double elapsed = (end - begin) / ((double) CLOCKS_PER_SEC); // measured in seconds
```

For today's lab, our goal is to gather statistics about the efficiency of C++ vectors and lists, hoping to see evidence of the amortized nature of vectors and the relative efficiency of lists. The experiment setup should be to run a loop to insert N numbers into a container, recording the time it takes for each individual insertion and then computing the overall maximum insertion time as well as the overall average insertion time. Furthermore, we wish to vary the experiment along two axes.

- Trying this with a `vector` and with a `list`.
- Performing the insertion at the *front* of the container or the *back* of the container.

Note that both the `vector` and `list` classes support the `push_back` method for adding a new element at the end. While the `list` supports `push_front` as well, this is not supported by a `vector` (for good reason, as we will see). A syntax that can be used for both the vector and the list for inserting at the front is `data.insert(data.begin(), item)`, assuming that `data` is the name of the container.

The reverse of this page gives charts for you to fill out. We doubt you will get to complete all of the charts, but do your best to gather enough evidence to draw a conclusion about the efficiency of those behaviors.

Inserting at the back of a vector

N	cumulative	average	worst
10,000			
20,000			
40,000			
80,000			
160,000			
320,000			
640,000			
1,280,000			
2,560,000			
5,120,000			
10,240,000			
20,480,000			
40,960,000			
81,920,000			
163,840,000			
327,680,000			

Inserting at the front of a vector

N	cumulative	average	worst
10,000			
20,000			
40,000			
80,000			
160,000			
320,000			
640,000			
1,280,000			
2,560,000			
5,120,000			
10,240,000			
20,480,000			
40,960,000			
81,920,000			
163,840,000			
327,680,000			

inserting at the back of a list

N	cumulative	average	worst
10,000			
20,000			
40,000			
80,000			
160,000			
320,000			
640,000			
1,280,000			
2,560,000			
5,120,000			
10,240,000			
20,480,000			
40,960,000			
81,920,000			
163,840,000			
327,680,000			

inserting at the front of a list

N	cumulative	average	worst
10,000			
20,000			
40,000			
80,000			
160,000			
320,000			
640,000			
1,280,000			
2,560,000			
5,120,000			
10,240,000			
20,480,000			
40,960,000			
81,920,000			
163,840,000			
327,680,000			