Data Structures and Algorithms

Timing a function

Most systems seem to have implementations of the ANSI C routine `clock()`. You can find its specifications with the

```
man 3 clock
```

command.

Each call of `clock()` returns the time in ticks since the last call. So, to time a function, simply call the function to be timed in between two `clock()` calls:

```c
long t;
t = clock(); /* Call function to be timed */
x = f( ..... ); /* Calculate time since previous clock call */
t = clock() - t; /* Convert to seconds */
printf("CPU time %g\n", (double)t/CLOCKS_PER_SEC );
```

A good tool-building approach would have you build another (trivial) function, say,

```c
double TimeUsed();
```

which returned the time difference in seconds and prevented your program from needing to worry about the conversion from ticks to seconds. For an example of a simple function which can readily be included in your program, see timer.h and timer.c.

However, be careful to note that the minimum resolution of the `clock` function will almost invariably be more than 1 tick. On the SGI machines, it’s actually 10ms. This means that you have to ensure that your test function runs for much longer than 10ms to get accurate times. Thus you will usually have to call the function under test quite a few times in order to find an accurate time:

```c
long t;
double tN;
t = clock(); /* Call function to be timed N times */
for(i=0;i<N;i++) {
    x = f( ..... );
}
/* Calculate time since previous clock call */
tN = clock() - t;
/* Convert to seconds */
tN = tN/CLOCKS_PER_SEC;
/* Calculate the average */
printf("CPU time %g\n", tN/N );
```
You will need to determine a suitable value of N by experimentation. It will obviously vary with the complexity of the function being tested!