CS109 PROJECT: MEASURING RUNNING TIME
Due Wednesday, Dec. 5

The purpose of doing this lab is to gain some experience in measuring the running time of programs and to convince ourselves that the big-oh analysis is realistic, at least as input sizes get large. In a sense, it is like the first physics experiment where you measure the acceleration of gravity.

We are operating under a constraint this quarter that I can only hope will not make the project unworkable. We're going to measure the running time of programs written in ML. But Mike Cleron's interface does not support timing. Thus, we have to use unix (which has a `time` command, and that means we have to use *sml* (standard ML), which should be compatable with the ML described in the Cleron notes. However, `sml` is available on the old portia, now renamed "bart," and is not yet available on the new portia. I have requested that it be made available, but there are no guarantees. If not, then you must use bart, on which `sml` is available. But bart tends not to be available, since it crashes at random. One possibility is to write and debug the programs you need on the MAC's and mail them to yourself on bart for the measurement of running time.

The Programs to be Written

To do the experiment you need to write ML programs that implement

1. Selection sort, similar to the Pascal program in Fig. 2.2 of the notes.
2. Merge sort, like the Pascal program of Section 2.8. You have already written the important `split` function in ML, and I gave you `merge` as an example on the board.

Measuring Running Time

Suppose you have written an ML program on a file `foo`. You can execute it by typing

```
sml <foo
```

in response to the unix prompt. You can run the same program and get an analysis of the running time by typing instead

```
time sml <foo
```

After whatever output you get from the `sml` interpreter and your own program, you'll see a line like

```
0.6u 0.7s 0:04 31% 1128+1839k 30+1io 127pf+0w
```

The first element says you took 0.6 seconds of "user time," i.e., your program kept the processor busy for that long. The second element says that in addition, the processor spent 0.7 seconds handling the system calls (for disk operations, e.g.) required by your program. The third says that the total elapsed time was 4 seconds, including time the machine was handling somebody else's program. The fourth says that you got 31% of the processor while you were running. The remaining elements tell you about the space used by the program, and how many disk operations of various types were performed on your behalf. If you are curious, a complete explanation can be had by consulting the on-line unix manual; type

```
man time
```

In a sense, the appropriate measure of the running time of your program is the sum of user and system time, or 1.3 sec. in the above example. However, system time is often a function of the load on the machine, and can vary for different runs. (User time can vary too, but generally not by much.) In this experiment, let us take only the user time as the "running time."

The Experiments

When you run a program like merge sort under sml, there will be some "overhead" that is not really part of the running time of your program. These costs include starting up the interpreter and preprocessing your program. We can get a good estimate of these costs if you run your programs on empty lists first, so true running time is approximately 0. You can then subtract the "overhead" from all measurements you take.