1. Get copies of the sample program `int_prof.c`, the auxiliary module `timing.c` and header `timing.h`, and the makefile (`Makefile`) in one of your subdirectories. Compile and link the program by typing:

```
make int_prof < cr >
```

This program has 10 built-in functions (numbered 0, 1, 2, ..., 9) and is invoked by typing

```
nice -n 19 int_prof { function number } < cr >
```

**Important: be sure to use “nice -n 19” as above.** Many of these functions will take several minutes of CPU time and the process should have its priority lowered from that of interactive jobs. If you don’t type a function number it uses function zero. The functions are as follows:

0. \( \exp(-(0.5)x^2) \)
1. \( \cos(\sin x) \)
2. \( \sin(x^2) \)
3. \( \sqrt{1-(0.25)\sin^2 x} \) (typical elliptic function)
4. \( 1/(1+x^4) \) (typical rational function)
5. \( 1/(\log x + 2) \) (prime number density function)
6. \( x^4 \sin(1/x) \)
7. \( \sqrt{x} \exp(-1/\sqrt{\sqrt{x}}) \)
8. \( x \log(x) \)
9. \( x \log(1+x) \sin(1/x) \)

The program runs the following four numerical integration algorithms: bounded variation with left and right sums, the trapezoidal rule, the recursive trapezoidal rule, and Simpson’s rule. Fifty definite integrals of the selected function (to 5-place accuracy) are done for each algorithm with the result and number of subdivisions used posted. The total time (real, virtual, profiling) is posted after the run of each algorithm. The important time is the profiling time (PROF) since it shows CPU time used directly by the program and CPU time used by the operating system on behalf of the program (e.g. in I/O). In addition the program computes (estimates actually) the \( C^k \) norms (for \( k = 1, 2, 3, 4 \)) of the function under consideration where

\[
C^k_{\text{norm}} = \frac{\max_{0 < x < 1} |f^{(k)}(x)|}{k!}
\]

**Assignment** Run the program on each of the ten functions and save the results, for example, you could do

```
nice -n 19 int_prof { function number } > tmpfile < cr >
lpr -Pcs4050text tmpfile < cr >
```

Write a one-page explanation of the results. Things you should explain include (but are not limited to): why the recursive trapezoidal rule usually beats the trapezoidal rule despite using more subdivisions, why the bounded variation with left and right sums is usually the slowest, and why the recursive trapezoidal rule and Simpson’s rule are generally faster than the other methods but can be very slow for some functions. You should refer to specific results in order to back up your conclusions. Send me your explanation in-line as ASCII text – do NOT send me a Microsoft Word document.