POINTS TO PONDER

7.1 In our local computer center, control cards must be in a fixed format: there must be a dollar-sign in column 1, the operation (e.g., "FORTRAN") begins in column 8, and any additional information begins in column 16. Any deviation in any card typically causes the run to be aborted. Fewer than 30,000 control cards are submitted per day.

(a) If processing free-form input were to add 100 microseconds per card of operating system overhead (a generous allowance), what would this flexibility cost per day? (Answer: 3 seconds.)

(b) What does it cost one user to have to re-submit one job because of a mispunched card?

(c) Debate the pros and cons of free-format versus fixed-format input from the users' and the system's viewpoints.

(d) Find some analogous examples of short-sighted economy at your computer center.

(e) [Term project] Try to get them changed.

7.2 Statement-frequency counts (profiles), although useful measurement tools in a simple language like Fortran, break down to some extent in more complex languages like PL/I where a single "statement" can involve substantial computation. (For example, consider the implicit array operations.) What kinds of Fortran statements require non-trivial amounts of computation? How could the compiler advise the user of the probable complexity of constructions in a program? Would it be worth it? What other aids can you suggest?

7.3 The table of comparisons and exchanges for the three sorts shows that the Shell sort has a much higher ratio of comparisons to exchanges than the interchange sorts. What does this imply? Can the information be used to improve the algorithm?

7.4 Our timing tests of sorting methods were made on arrays of random numbers. Experiment to decide what degree of non-randomness is necessary before the "efficient" sort is faster than the simple sort. What does non-randomness do to the Shell sort?

7.5 Recoding a program in assembly language to make it as fast as possible is a last resort usually taken too early and too often. There is a folk-theorem that "10 per cent of the code takes 90 percent of the run time." Develop a methodology for deciding what parts of a program should be converted to assembly language, based on this observation. (You might try to verify it first.)