Selection Sort Subroutine

In this exercise you will use the "Selection Sort" algorithm to sort a list of numbers read from a file, and gain experience at writing a Fortran subroutine.

Subroutines are an important part of Fortran. By solving a general problem (like sorting a list of numbers) and turning the solution into a subroutine the programmer builds modular tools which can easily be used in future programs. Subroutines are also a good way to write programs in a structured form, leading to more efficient code and fewer errors.

1. The Algorithm

The "selection sort" algorithm (I sometimes call this the "sweep" sort) is similar to the bubble sort, but more efficient. The basic element of the selection sort is still to compare two numbers in the list and exchange them if they are out of order. The only difference is the order in which the pairs are examined.

Instead of comparing nearest neighbor pairs of numbers, as is done in the bubble sort, in the selection sort you begin with the top number in the list and compare it with every other number in the list. If you find a number which is smaller then you exchange it with the number at the top of the list. After you have gone through the whole list you will have the smallest number at the top of the list. Next, you move down to the second number in the list, and compare it with all numbers below it. Once you have done that you will have the next smallest number in the second position. Essentially, once you have completed one of these "sweeps" through the list, you have the next number in order, and a list of unsorted numbers with one less element. You then apply the same procedure to sort this smaller list.

2. Operation Counts

It's useful to compare the two different sorting algorithms in terms of the amount of work done by the computer to sort a list of N numbers. Since the basic "compare and swap" step is the same for both algorithms we just need to figure out how many of these steps are needed for each algorithm. In the bubble sort there are $N-1$ comparisons of pairs of numbers for each pass through the list, and there are $N-1$ passes through the list, so the total number of steps is $(N-1)^2$. Since the leading behavior of this for very large N is N^2 we say that this is an "order N-squared" algorithm, or "$O(N^2)$.

You should determine the number of steps required to sort a list of N numbers using the selection sort rather than the bubble sort. Is it $O(N^2)$ or is it better? The selection sort is actually better than the bubble sort – why is this so?

3. Problem

Write a program which reads some number (up to 2000) of REAL values from a given file and then calls a subroutine called SORTEM to sort these numbers. The first argument to SORTEM will be the (integer) number of numbers to sort. The second argument to SORTEM will be the name of a (real) array containing the numbers to be sorted. When the subroutine returns the numbers in the array are to be sorted in ascending order.

The SORTEM subroutine should use the "Selection Sort" algorithm I described above. If you have any questions about the algorithm please ask me.

4. Input

Your program should read from the standard input (the terminal) the name of the file containing the numbers to be sorted. It should then open this file and read all the numbers to be sorted into an array. There will be one number per line. You should *not* echo the data you read from the file, but do remember to echo the file name.

5. Output

Write the sorted list of numbers to a file which has the same base name as the assignment but has the "extension" of "`.rpt`" (Example: `mavassar12.rpt`). Use an `F13.6` format to write the numbers to the file, and put as many as you can on a single line, leaving appropriate space between them so that the list is readable. You should print a note to the user telling them that the sorted data have been written to this file. Do not write the sorted list to the terminal!

In your sample run you should use the Unix "time" command to show how long your program takes to sort the list of numbers. Also run your bubble sort program, and compare the times. Do they match what you would expect from your calculation of the algorithmic complexity?

6. Testing

To test your program you will need a file containing a list of numbers. You can copy such a file from one of my directories. The file `~myers/fortran/R2000.d` contains 2000 random floating point numbers, one per line. Also, you can create your own file containing a list of random numbers by copying the program `~myers/fortran/realist.f` to your own directory and compiling it and running it.

7. Revision Control System (RCS)

To gain experience with using RCS (the Revision Control System) you should also include at the top of your program (after your name and date and the like) an RCS comment line like the following:

```
C @(#)   $Revision: 2.3 $ :  $Date: 2003/03/15 12:36 $ :  $Author: mavassar
$
```

You do not have to actually use RCS in this exercise, but you will use it in the next exercise, so it's good to start getting ready for it.

8. Reading

You may find a discussion of sorting algorithms in your book. If you want further information about sorting, see *Sorting and Searching* by Donald Knuth (this is Volume 3 of his series on *The Art of Computer Programming*).

You will need to read about how to write subroutines in Fortran, and how to invoke a subroutine from a program. In particular, you will need to read about the `SUBROUTINE` and `RETURN` statements and the `CALL` command.

You might find it useful to use a `PARAMETER` statement to define the length of the array used to hold the numbers to be sorted. Then if you ever need to sort more than 2000 numbers you can easily change the array bounds by changing just one parameter.