5DV022 Software Exercise 2 Fall 2008

Solutions to this assignment are due on October 6, 2008 at 5pm (1700¿ This is a *laboratory exercise*, and the amount of collaboration allowed is as defined in the documents *Riktlinjer vid labgenomförande* (*Rules for the Preparation of Laboratory Exercises*) and *Hederskodex* (the *Honour Code*), which may be found on the course home page.

Late submissions are subject to penalty as described in the course syllabus.

This exercise may be carried out in groups of at most three individuals.

The report **must** be written in English. Reports written in any other language will be returned, ungraded, unless prior written arrangement has been made with the course instructor.

Place your report in the instructor's mailbox or give them to the instructor during class. Do not place it in the mailbox for laboratory reports.

1. General Description

The purpose of this assignment is to examine the efficiency of a super sorting algorithm which combines quicksort and insertion sort. As has been shown, quicksort is, on the average, a very fast sorting algorithm. However, it has the disadvantage that it is not very efficient on small arrays. Therefore, one potential improvement is to switch to a different sorting algorithm when the size of the array to be sorted becomes small. One particular algorithm which is quite good on small arrays is straight insertion sort. While it has $\Theta(n^2)$ worst-case time complexity (where n is the size of the list to be sorted), the constant is small enough that it works very well for small arrays.

2. Specific Required Tasks

The assignment includes writing and profiling a set of procedures. For consistency and simplicity of presentation, the syntax of Pascal is used in this description, but you should develop your solutions in C, since they must be run using the profiling software which you developed for Software Assignment 1.

Write a procedure

```
Sisort(var a: artype;
           low, high: dimtype);
```

with

```
type dimtype = 1..100000;
     artype = array[dimtype] of integer
```

which sorts the subarray of `a` between `low` and `high` inclusive, using a straight insertion sort. Straight insertion sort is described as Algorithm 3.9 on on page 151 of the text.

Write a procedure

```
Qsort(var a: artype;
             low, high: dimtype;
             switchpt: dimtype)
```

which sorts the subarray of *a* between `low` and `high` inclusive, using quicksort for the case of $\texttt{high} - \texttt{low} > \texttt{switchpt}$, but which switches over to `Sisort` when $\texttt{high} - \texttt{low} \leq \texttt{switchpt}$. For quicksort, use a randomly selected key from the subarray to be sorted for the partition element. Make sure that you select an *actual* key from the array; do not simply generate a *possible* key from a random number.

Profile runs of `Qsort` on random arrays of 100000 integers for varying values of `switchpt` to determine the best value. Specifically, start with profiles for `switchpt` = 0, 8, 16, . . . , 104. For each of these values of `switchpt`, run `Qsort` on three different random arrays, but use the same three random arrays in each case. Using these preliminary data, try to zero in on an exact value of `switchpt` which gives the best performance. To do this you will need to run profiles for additional values of `switchpt` not contained in the preliminary range.

Also, for the optimal switchpoint, run a profile using the utility `gprof`. Use this profile as a sort of check on your software. Large discrepancies should be investigated thoroughly.

Turn in:

Your submission should be in the form of a report on an experiment. It should contain the following parts.

1. An introduction which clearly describes the overall goal and ideas of the experiment.
2. A detailed description of the experiment which was conducted. Include a discussion of any problems you encountered, together with an explanation of how you dealt with them. Also, clearly identify the environment in which the experiment was conducted; *i.e.*, the machine and programming environment on which it was run.
3. An analysis of the results of the experiment, together with the conclusions drawn from the experiment. This should include, but not be limited to, a graph and a table of total execution time as a function of the value of `switchpt`. (The *gnuplot* program is recommended for use in plotting the data.) Any large discrepancy between the time reported by `gprof` and by your utility should be analyzed and explained.

4. An appendix containing the following:
   - (a) A source listing of your program.
   - (b) Sample sorts on small input sets to demonstrate that your sorting algorithms work.
   - (c) All profiles generated, with each clearly labelled with the case considered.

If you just turn in a program and some data printouts, you will not receive much credit, even though your programs work perfectly. You **must** submit a clear description of what you did and what conclusions you reached. Furthermore, it must be organized as outlined above. This is an experimental investigation, not a programming exercise.

Notes:

1. Do not output the arrays in the profile. That will just waste time.
2. Thoroughly debug your program on small data sets before running the tests on the large arrays.
3. For the task at hand, only the main program need be profiled. Profiling the procedures individually will add overhead to the experiment. However, for the optimal value of `switchpt` (and for a few other values) run a profile with all procedures individually measured, to determine (a) the kind of overhead such measurement introduces, and (b) the division of time between `Qsort` and `Sisort`.
4. If necessary to obtain good profiling data, the array size of 100000 may be changed. A discussion of what had to be done along these lines should be part of the description of the experiment.
5. To accumulate the test results in a form which is easily usable, you may wish to modify your augmentation routines so that they write out separate files for each switchpoint size; *e.g.*, `timing_0.dat`, `timing_4.dat`, etc. Thus, in `profile.h` you may wish to replace the simple `start_log` command with one which takes a single argument, the name of the log file. You may also wish to improve your analyze program so that it can produce the desired summary data over the range of switchpoints. This is fine, as long as you document clearly what you are doing. This is one of the advantages of writing your own profiling software, as opposed to using packages developed by someone else.
6. The goal of this exercise is to gain insight into the techniques of profiling, and not to come up with results which conform to some nice, predictable result. It may not be the case that the data show a nice, smooth curve with a single, clean minimum. However, there should be a general minimum; in other words, the performance should not be flat or wildly varying over large changes in the value of `switchpt`.
7. If you have particular difficulty in obtaining clean results, try varying the level of optimization of the compiler. Less optimization may give a cleaner result.

The report must be typeset; handwritten reports will not be accepted.