CS 198-23 Applications of Algorithms
Homework 2

Due 02/09/2011 5:59pm

Submission Guidelines:
You will need to submit using the Unix Submit function. You will need to submit atleast 8 files -

1. mergesort.cc(as well as supporting files)
2. sort-comparison.txt
3. kElementFind.cc
4. Matrixmul.cc(along with any helper files)
5. Strassen.cc(along with any helper files)
6. A makefile that works with appropriate options
7. Output from a profiler as profileMat.txt and profileStrass.txt.

The submission name will be hw2. If you are unable to complete a question, submit as an empty file.

1. Code Timing/Profiling (3 points)

Assignment:

1. Implement Mergesort. Implementation should be able to run on integers, doubles, floats, as well as custom defined types.
2. Compare this to the Unix sort implementation. Time your code on arrays of type int and each 25 **random** sized arrays of sizes between 2^1 to 2^{20} (you must use sizes 2^1 and 2^{20})

Reference:

1. Section 2.3 - Algorithms(DPV) for Mergesort
2. Find and study the documentation for time.h

2. More Coding Practice (2 points)

Assignment

1. Implement 2.22 from Chapter 2 of Algorithms(DPV). The question for reference is: *You are given two sorted lists of size m and n. Give an O(logm + log n) time algorithm for computing the kth smallest element in the union of the two lists.* Again your implementation must work on any arbitrary type, defined with appropriate templates.

Reference

1. Chapter 2, Algorithms(DPV)

3. Matrix Multiplication (5 points)

Assignment

1. Here is the hard question of this assignment. You must implement Matrix multiplication, both the simple algorithm(that takes n^3 time) and Strassen's Algorithm. We expect an implementation that works only with type int.
2. The second half of this assignment is using gprof and make. You are learn how to compile your files using make. The makefile should accept options for
    a. mat - Simple Matrix Multiply
    b. Strassen - Strassen's Algorithm
3. You also need to submit profiler output. For filenames, refer to submission instructions. Make sure you submit readable output as opposed to gmon.out. Extra points for interpretation of the output.

How to Use GProf in 5 Easy Steps.(from here)

1. **Get your program working!!** gprof is *not* a debugger. Use it once you have a working program to optimize that program.
2. Compile and link with the -pg option. If you use an Owen Astrachan patented makefile this simply means changing the CFLAGS variable.
3. Run your program normally; that is, pretend you didn't do anything to it and do what you would normally do (checking difficult, slow, or fast cases, of course).
4. Type gprof exec > out where exec is replaced by your executable's name and out by some meaningful name for the profiling information. For instance, if your executable were "foo" on the third run compiled with the -O2 option then you might type:
5. `gprof foo > run3.withO2.stats`
6. Look over the output and learn what it means. Hint, go to the *second* table. The first is pretty useless. To get there, search for "granularity" twice from the top of the file (in emacs, use C-s; in less or more use /).

Reference

Section 2.5, Algorithm(DPV)

Debugging Code

Using gdb

Make tutorials

http://mrbook.org/tutorials/make/

http://www.opussoftware.com/tutorial/TutMakefile.htm

Using gprof

Quick Start Guide

Detailed Tutorial

Another Tutorial