PENN STATE UNIVERSITY
Department of Economics

Econ 597D Sec 001 Computational Economics — Gallant
Homework 4 — Fall 2015
Due Sep 22, 2015

Available at the course website http://www.aronaldg.org/courses/compecon by clicking on `Source Code` and then `ch05` are files prog01.cpp through prog05.cpp and a makefile with which to build them.

prog01 illustrates various means to store an array, methods of indexing, inner product computations, and the sort function. The arrays a and b are allocated on the stack.

prog02 is the same as prog01 except that the arrays a and b are allocated on the heap. Memory allocation failure is handled with traditional error handling methods. As n is increased, prog01 will fail to run sooner than prog02

prog03 through prog06 are the same as prog02 with the sort illustration removed and the time, clock, gettimeofday, and stopwatch functions, respectively, used to time the speed of an inner product. C++ exceptions, i.e, try and catch, are used to deal with memory exhaustion.

1. In prog01, increase n until the program fails. At what value of n did it fail?

2. Use this same value of n in prog02. Did it fail?

3. Run prog06 to see if you can detect a difference in the speed of the inner product computations.

   You will probably have to increase n to detect a difference on newer machines. Be very cautious as you increase n, being careful to stop if you notice that the machine becomes the least little bit sluggish, unless you own the machine. You are liable to hog resources to the point that no one else can use the machine. This can provoke the wrath of IT personnel.

   What are the times for the six methods of computing an inner product using prog06?

4. Another method of timing, e.g. prog06, is to call it with "`time prog06`". Try this. How does it compare to the sum of the times in question 3?

5. In prog06 where a usage similar to `sum += a*b`, appears, replace it with `sum=sum+a*b`. Did this usage degrade performance?

6. Turn in the answers to the five questions above.

You can earn two bonus points as follows. If a machine has `librt` and therefore has `clock_gettime`, you can do a better job of timing; `argux6` has `clock_gettime` as does `cygwin`; a Mac does not. Look at the comments at the end of the source code for `stopwatch` in `libscl` to see how to use `clock_gettime`. Replace the usage of `gettimeofday` in `prog05` with `clock_gettime`, being careful to note that `gettimeofday` uses micro seconds (10^{-6} seconds) whereas `clock_gettime` uses nano seconds (10^{-9} seconds). To get the bonus points, in addition to the answers to the five questions above, turn in the modified source code and the timings from a run.