STAT/BIOSTAT 534 Statistical Computing
Spring Quarter 2017
Final Project

Adrian Dobra
adobra@uw.edu

The final project is due on Sunday, June 4 at 11:00pm. You should use the dropbox to submit your code. Please note that you will be graded not only on how your code works, but also on how readable your code is.

Problem 1 (100 points)

You need to implement in C the following sorting algorithm for a vector `double v[n]`. Construct a binary tree corresponding with the vector `v` as we did in class. For each element `v[i]`, the left subtree should contain elements of `v` smaller than `v[i]`, while the right subtree should contain elements of `v` greater or equal to `v[i]`. Please use the code I gave you and implement only the parts of the code you do not already have.

Repeat the following steps until the tree becomes empty:

1. Find the node in the tree containing the smallest key (number).
2. Record this number.
3. Remove this node from the tree.

Please note that, at each iteration, your code must produce the updated tree that has one node less than the tree from the previous iteration. A fully recursive solution will not be accepted.

The resulting sequence of numbers will be the elements of `v` in increasing order. This sequence should be the output of your code. Your program should function properly for the file"somenumbers.txt".

Problem 2 (200 points)

Consider a discrete random variable X that takes values $\{0, 1, \ldots, n\}$ with probabilities

$$p_i = P(X = i) \propto \binom{n}{i} = \frac{n!}{i!(n-i)!}.$$

Remember that $\Gamma(n+1) = n!$. Your code should work for any $n = 1, 2, \ldots$.

Please write a parallel program that samples from the distribution of X as follows. Use the Master/Slave scheme we studied in class. The Master process should proceed as follows:

1. The Master should ask each Slave process to generate 25000 samples from the distribution of X, and return an estimate for $p_i = P(X = i)$. It is assumed that all the processes know the distribution we need to sample from, hence the work request of the Master should include two integer numbers: the number of samples to generate and $i \in \{0, 1, \ldots, n\}$.

2. The Master should wait until each Slave finishes its work and record the corresponding results. Each Slave process returns 2 real (double) numbers: i, and an estimate for p_i.

3. The Master should output the estimates of $p_0, p_1, \ldots, p_n$ *in this order* (not in the order in which the Master has received the results).

Each Slave process should proceed as follows:

1. The Slave should listen for an work order from the Master.

2. Once an order has been received, the Slave should sample the number of samples requested by the Master process using the following independent Metropolis-Hastings algorithm. At iteration 0, start with some arbitrary value $i^{(0)} \in \{0, 1, \ldots, n\}$. The state of the chain at iteration t is $i^{(t)}$. At iteration t, draw another i^* uniformly from $\{0, 1, \ldots, n\}$. Set $i^{(t+1)} = i^*$ with probability

   $$\min\{p_{i^*}/p_{i^{(t)}}, 1\}.$$

   Otherwise, set $i^{(t+1)} = i^{(t)}$.

3. The Slave determines an estimate $\hat{p}_i$ of $p_i = P(X = i)$ where i was the index contained in the request received from the Master by counting how many samples $i^{(0)}, i^{(1)}, \ldots, i^{(25000)}$ are equal with i, and dividing by the number of samples generated (25,000 in this application).

4. The Slave transmits i and $\hat{p}_i$ back to the Master.