Parallel Quicksort

In this assignment, you'll implement a program to sort integers in parallel by doing a straightforward parallelization of the sequential quicksort algorithm. The algorithm is described here at a high-level and you should fill in the remaining implementation details.

The input to the algorithm is a file containing the number of integers to be sorted followed by the integers themselves. The output of the program is the sorted list of the integers. Input and output of the data is already taken care of for your convenience in the programming framework.

Let n denote the number of integers and p denote the number of processors. The integers are equally distributed to the processors: Each processor has an array A holding either $\lceil \frac{n}{p} \rceil$ or $\lfloor \frac{n}{p} \rfloor$ integers. The algorithm is recursive: At some stage during the algorithm, suppose there is a communicator of size q on which m integers should be sorted and that they are distributed in the usual manner. If $q = 1$, then any serial sorting algorithm can be used to sort the input. You may either write your code, copy from a book or web site if available, or even use the Unix sort program.

If $q > 1$, all processors in the communicator use the same random number generator to generate a random number between 0 and $m-1$, say k. The processor that has the k^{th} integer (called pivot from here on) broadcasts it to all processors in the communicator. Each processor goes through its integer array and partitions it into two subarrays – one containing integers less than or equal to the pivot and another containing integers greater than the pivot. Using an *Allgather* operation, the number of integers in each of the two subarrays on all the processors are gathered on every processor. Compute the total number of integers less than or equal to the pivot (say m') and the total number of integers greater than the pivot (say m''). Partition the q processors to the two subproblems of sorting m' integers and sorting m'' integers respectively, by allocating processors in proportion to the problem sizes (make sure you do not allocate zero processors to a problem). Using the gathered information on the sizes of the subarrays on all processors, each processor can compute where to send its data and from where to receive its data. Use an *Alltoall* communication to perform the data transfer. Create two new communicators corresponding to the two partitions. Recursively sort within each partition.