Distributed Sorting with MPI
CS 462 Parallel Programming Project #1
Dr. Beck

Problem statement:
Assume that N unsorted data items are initially distributed across n nodes, where \( n^2 \) divides N evenly. The input is initially in an array \( a_i \) in each of the n nodes. The idea is to create arrays \( s_0, s_1, \ldots, s_{n-1} \) on the nodes such that the resulting array spanning across all nodes – when the nodes are ordered by rank – is sorted. More formally,

For all \( 0 \leq i, j, k, m < n \):

If \( k < m \) or \( (k = m \text{ and } i < j) \) then \( s_k[i] < s_m[j] \)

You are to use MPI to implement two solutions to this problem

1. Uniform Distribution (adapted from Bubble Sort)
   a. Sort each array \( a_i \) locally on node i.
   b. Consider each array \( a_i \) to consist of n subarrays \( a_{i,j} \) defined to be the \( M = N/n^2 \) elements \( a_{i,j} = a_i[M*j \ldots M*(j+1)-1] \).
   c. Build a new array \( s_i \) on each node \( i \) consisting of the concatenation of subarrays is \( s_i = a_{0,i} . a_{1,i} . a_{2,i} \ldots a_{n-1,i} \) (where the operator "." denotes array concatenation.)
   d. Sort each array \( s_i \) locally on node i.
   e. For each pair of nodes with consecutive rank \( i \) and \( i+1 \)
      i. Send the largest 16 elements of \( s_i \) to node \( i+1 \) and the smallest 16 elements of \( s_{i+1} \) to node \( i \). Nodes 0 and \( n-1 \) only send in one direction.
      ii. Sort the 32 items at each boundary (e.g. the largest 16 elements of \( s_i \) and the 16 smallest 16 elements of \( s_{i+1} \)) locally on each processor.
      iii. Delete the largest 16 items on every node (except rank \( n-1 \)), and the smallest 16 items from every node (except rank 0).
      iv. Sort the arrays \( s_i \) locally.
   f. Repeat steps d and e until the sort is completed.

2. Approximate Partition (adapted from Quicksort)
   a. Choose \( n-1 \) partition values \( p_1 < p_2 \ldots < p_{n-1} \) (analogous to the pivot in Quicksort) and distribute them to all nodes. The partition should be chosen by choosing one element at random from each array \( a_i \).
   b. Partition each array \( a_i \) into n subarrays \( a_{i,j} \) such that for each item \( x \) in \( a_{i,j} \), \( p_1 < x \leq p_{j+1} \). For the purpose of this condition succinctly, assume that \( p_0 = -\infty, p_n = \infty \).
   c. Build a new array \( s_i \) on each node \( i \) consisting of the concatenation of subarrays \( s_i = a_{0,i} . a_{1,i} . a_{2,i} \ldots a_{n-1,i} \).
Measuring performance of your solutions:

Your MPI programs should generate data randomly on each node, according to a specified input distribution that can be explicitly programmed. You are to test each solution 100 times on two sets of data, one generated with a uniform distribution on the numbers \([0,1023]\) and one generated with a distribution that is divided into quarters, with the probability of data items in the intervals \([0 .. 255]\) having 7 times the probability of data items in the intervals \([256 .. 1023]\).

The performance of each solution should be measured by sorting datasets of 1M, 5M, 25M and 50M items in size for each distribution, on 16, 32 and 64 nodes. If any of your runs exceeds what you consider a manageable time limit you can terminate it and indicate this fact in your results.

Improving your solution:

You should suggest a version of each solution that is modified in some way to improve performance. You should explain the rationale behind your modification, then implement and test it as for your original version (see above). You should explain your results, but obtaining improved performance is not required.