Name: ____________________________

Directions: **Work only on this sheet.** Put all your "fill the blank" answers in one place, say the lower-right part of this side, or on the back. Format:

```
#1a.
x+y
#1b
if(u > v) w = 3;
...
```

MAKE SURE TO COPY YOUR ANSWERS TO A SEPARATE SHEET FOR SENDING ME AN ELECTRONIC COPY LATER.

1. Below is an MPI version of the bucket sort in our OpenMP example (except that the bin boundaries are assumed known ahead of time, rather than calculated from a sample). The message-passing strategy is outline in the comments at the beginning of the code. Fill in the blanks.

```
// bucket sort, bin boundaries known in advance

// node 0 is manager, all else worker nodes; node 0 sends full data, bin
// boundaries to all worker nodes; i-th worker node extracts data for
// bin i-1, sorts it, sends sorted chunk back to node 0; node 0 places
// sorted results back in original array

// not claimed efficient; e.g. could be better to have manager place
// items into bins

#include <mpi.h>

#define MAX_N 100000  // max size of original data array
#define MAX_NPROCS 100  // max number of MPI processes
#define DATA_MSG 0  // manager sending original data
#define BDRIES_MSG 0  // manager sending bin boundaries
#define CHUNKS_MSG 2  // workers sending their sorted chunks

int nnodes,  //
    n,  // size of full array
    me,  // my node number
    fulldata[MAX_N],
    tmp[MAX_N],
    nbdries,  // number of bin boundaries
    counts[MAX_NPROCS];
float bdries[MAX_NPROCS-2];  // bin boundaries

int debug,debugme;

init(int argc, char **argv)
{
   int i;
   debug = atoi(argv[3]);
   debugme = atoi(argv[4]);
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&nnodes);
   MPI_Comm_rank(MPI_COMM_WORLD,&me);
   nbdries = nnodes - 2;
   n = atoi(argv[1]);
   int k = atoi(argv[2]);  // for random # gen
   // generate random data for test purposes
   for (i = 0; i < n; i++) fulldata[i] = rand() % k;
   // generate bin boundaries for test purposes
   for (i = 0; i < nbdries; i++) {
      bdries[i] = i * (k+1) / ((float) nnodes);
   }
}

void managernode()
{
   MPI_Status status;
   int i;
   int lenchunk;  // length of a chunk received from a worker
   // send full data, bin boundaries to workers
   for (i = 1; i < nnodes; i++) {
      MPI_Send(BLANKa,BLANKb,MPI_INT,BLANKc,DATA_MSG,MPI_COMM_WORLD);
      MPI_Send(BLANKd,BLANKe,MPI_FLOAT,BLANKf,BDRIES_MSG,MPI_COMM_WORLD);
   }
   // collect sorted chunks from workers, place them in their proper
   // positions within the original array
   int currposition = 0;
   for (i = 1; i < nnodes; i++) {
      MPI_Recv(tmp,MAX_N,MPI_INT,BLANKg,CHUNKS_MSG,MPI_COMM_WORLD,&status);
      MPI_Get_count(&status,MPI_INT,BLANKh);
      // memcpy(d,s,nb) copies nb bytes from s to d
      memcpy(BLANKi);
      BLANKj;
   }
   if (n < 25) {
      for (i = 0; i < n; i++) printf("%d ",fulldata[i]);
      printf("\n");
   }
}

// adds xi to the part array, increments npart, the length of part
void grab(int xi, int *part, int *npart)
{
    part[*npart] = xi;
    *npart += 1;
}

int cmpints(int *u, int *v)
{  if (*u < *v) return -1;
   if (*u > *v) return 1;
   return 0;
}

void getandsortmychunk(int *tmp, int n, int *chunk, int *lenchunk)
{
   int i,count = 0;
   int workernumber = me - 1;
   for (i = 0; i < n; i++) {
       if (workernumber == 0) {
          if (tmp[i] <= bdries[0]) grab(tmp[i],chunk,&count);
       }
       else if (workernumber < nbdries-1) {
          if (tmp[i] > bdries[workernumber-1] &&
              tmp[i] <= bdries[workernumber]) grab(tmp[i],chunk,&count);
       } else
          if (tmp[i] > bdries[nbdries-1]) grab(tmp[i],chunk,&count);
   }
   qsort(chunk,count,sizeof(int),cmpints);
   *lenchunk = count;
}

void workernode()
{
   int n,fulldata[MAX_N],  // size and storage of full data
       chunk[MAX_N],
       lenchunk,
       nbdries;  // number of bin boundaries
   float bdries[MAX_NPROCS-1];  // bin boundaries
   MPI_Status status;
   MPI_Recv(fulldata,MAX_N,MPI_INT,BLANKk,DATA_MSG,MPI_COMM_WORLD,&status);
   MPI_Get_count(&status,MPI_INT,BLANKl);
   MPI_Recv(bdries,MAX_NPROCS-2,MPI_FLOAT,BLANKm,BDRIES_MSG,
      MPI_COMM_WORLD,&status);
   MPI_Get_count(&status,MPI_FLOAT,BLANKn);
   getandsortmychunk(fulldata,n,chunk,&lenchunk);
   MPI_Send(chunk,lenchunk,MPI_INT,BLANKo,CHUNKS_MSG,MPI_COMM_WORLD);
}

int main(int argc,char **argv)
{
   int i;
   init(argc,argv);
   if (me == 0) managernode();
   else workernode();
   MPI_Finalize();
}
```

Solutions:

```
// bucket sort, bin boundaries known in advance

// node 0 is manager, all else worker nodes; node 0 sends full data, bin
// boundaries to all worker nodes; i-th worker node extracts data for
// bin i-1, sorts it, sends sorted chunk back to node 0; node 0 places
// sorted results back in original array

// not claimed efficient; e.g. could be better to have manager place
// items into bins

#include <mpi.h>

#define MAX_N 100000  // max size of original data array
#define MAX_NPROCS 100  // max number of MPI processes
#define DATA_MSG 0  // manager sending original data
#define BDRIES_MSG 0  // manager sending bin boundaries
#define CHUNKS_MSG 2  // workers sending their sorted chunks

int nnodes,  //
    n,  // size of full array
    me,  // my node number
    fulldata[MAX_N],
    tmp[MAX_N],
    nbdries,  // number of bin boundaries
    counts[MAX_NPROCS];
float bdries[MAX_NPROCS-2];  // bin boundaries

int debug,debugme;

init(int argc, char **argv)
{
   int i;
   debug = atoi(argv[3]);
   debugme = atoi(argv[4]);
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&nnodes);
   MPI_Comm_rank(MPI_COMM_WORLD,&me);
   nbdries = nnodes - 2;
   n = atoi(argv[1]);
   int k = atoi(argv[2]);  // for random # gen
   // generate random data for test purposes
   for (i = 0; i < n; i++) fulldata[i] = rand() % k;
   // generate bin boundaries for test purposes
   for (i = 0; i < nbdries; i++) {
      bdries[i] = i * (k+1) / ((float) nnodes);
   }
}

void managernode()
{
   MPI_Status status;
   int i;
   int lenchunk;  // length of a chunk received from a worker
   // send full data, bin boundaries to workers
   for (i = 1; i < nnodes; i++) {
      MPI_Send(fulldata,n,MPI_INT,i,DATA_MSG,MPI_COMM_WORLD);
      MPI_Send(bdries,nbdries,MPI_FLOAT,i,BDRIES_MSG,MPI_COMM_WORLD);
   }
   // collect sorted chunks from workers, place them in their proper
   // positions within the original array
   int currposition = 0;
   for (i = 1; i < nnodes; i++) {
      MPI_Recv(tmp,MAX_N,MPI_INT,i,CHUNKS_MSG,MPI_COMM_WORLD,&status);
      MPI_Get_count(&status,MPI_INT,&lenchunk);
      memcpy(fulldata+currposition,tmp,lenchunk*sizeof(int));
      currposition += lenchunk;
   }
   if (n < 25) {
      for (i = 0; i < n; i++) printf("%d ",fulldata[i]);
      printf("\n");
   }
}

// adds xi to the part array, increments npart, the length of part
void grab(int xi, int *part, int *npart)
{
    part[*npart] = xi;
    *npart += 1;
}

int cmpints(int *u, int *v)
{  if (*u < *v) return -1;
   if (*u > *v) return 1;
   return 0;
}

void getandsortmychunk(int *tmp, int n, int *chunk, int *lenchunk)
{
   int i,count = 0;
   int workernumber = me - 1;
      if (me == debugme) while (debug) ;
   for (i = 0; i < n; i++) {
       if (workernumber == 0) {
          if (tmp[i] <= bdries[0]) grab(tmp[i],chunk,&count);
       }
       else if (workernumber < nbdries-1) {
          if (tmp[i] > bdries[workernumber-1] &&
              tmp[i] <= bdries[workernumber]) grab(tmp[i],chunk,&count);
       } else
          if (tmp[i] > bdries[nbdries-1]) grab(tmp[i],chunk,&count);
   }
   qsort(chunk,count,sizeof(int),cmpints);
   *lenchunk = count;
}

void workernode()
{
   int n,fulldata[MAX_N],  // size and storage of full data
       chunk[MAX_N],
       lenchunk,
       nbdries;  // number of bin boundaries
   float bdries[MAX_NPROCS-1];  // bin boundaries
   MPI_Status status;
   MPI_Recv(fulldata,MAX_N,MPI_INT,0,DATA_MSG,MPI_COMM_WORLD,&status);
   MPI_Get_count(&status,MPI_INT,&n);
   MPI_Recv(bdries,MAX_NPROCS-2,MPI_FLOAT,0,BDRIES_MSG,MPI_COMM_WORLD,&status);
   MPI_Get_count(&status,MPI_FLOAT,&nbdries);
   getandsortmychunk(fulldata,n,chunk,&lenchunk);
   MPI_Send(chunk,lenchunk,MPI_INT,0,CHUNKS_MSG,MPI_COMM_WORLD);
}

int main(int argc,char **argv)
{
   int i;
   init(argc,argv);
   if (me == 0) managernode();
   else workernode();
   MPI_Finalize();
}
```