Name: _____

Directions: **Work only on this sheet** (on both sides, if needed). MAKE SURE TO COPY YOUR ANSWERS TO A SEPARATE SHEET FOR SENDING ME AN ELECTRONIC COPY LATER.

Important note: Remember that in problems calling for R code, you are allowed to use any built-in R function, e.g. **choose()**, **sum()**, **combn()** etc.

1. (10) In a certain line in some MPI program, we wish to receive a message from node 0, consisting of an array of 10000 integers, placing it in **y**. The message type will be XTYPE. We have declarations

```
int *y;
MPI_Status stat;
```

and have called **malloc()**, assigning the result to **y.** Give the call needed to receive the data.

2. A military commander in old Egypt, Amr ibn al-As, once wrote to a Caliph Umar, "I will send to Medina a camel train so long that the first camel will reach you before the last one has left me." (True story.) Fill in the blanks with terms from our course:

(a) (10) An increase in camel walk speed would reduce the _____.

(b) (10) An increase in the time it takes to load a camel and start it on its journey would reduce the _____.

3. This problem concerns the function **bsort()** in Section 1.3.2.6 of our book.

(a) (10) Briefly—in just one line in your electronic Quiz submission—discuss this algorithm in terms of the load balance issue.

(b) (10) Give a single C statement, to go between lines 92 and 93, that will be executed by the last thread, that is, the thread with the numerically largest thread number. The code with print out the index (i.e. subscript) within **x** at which the chunk written by thread 1 begins. If for example thread 1 writes to **x[12]** through **x[28]**, then 12 will be printed out.

(c) (10) Internally, the pragma on line 43 likely makes multiple calls to which **pthreads** function?

4. (40) This will be a variant of the matrix-multiply Snow code in our text. The two main differences are that (a) we send each worker its assigned rows of the multiplier, rather than sending the full matrix and assigned row numbers, and (b) the multiplier matrix is upper-diagonal.

Concerning (a), say we have two workers and the multiplier has 6 rows. Then in **uv()**, **uchunks[[1]]** will be the submatrix of **u** consisting of the first 3 rows of the latter. Concerning (b), we want to avoid wasteful multiplication by the 0s below the diagonal in **u**, but they are there, i.e. the matrix is not stored in compressed form.

Fill in the blanks (only one line or partial line each):

```
uv <- function(cls,u,v) {
   rownums <- splitIndices(nrow(u),length(cls))
   # tack on row numbers to u
   u1 <- cbind(u,1:nrow(u))
   uchunks <- list()
   for (i in 1:length(cls)) {
      uchunks[[i]] <- blank (a)
   }
   res <- clusterApply(cls,uchunks,chunkmul,v)
   Reduce(c,res)
}

chunkmul <- function(uchunk,v) {
   nr <- nrow(uchunk)
   nc <- ncol(uchunk)
   ncu <- nc - 1
   prod <- vector(length=nr)
   for (i in 1:nr) {
      urownum <- uchunk[i,nc]
      rng <- blank (b)
      prod[i] <- blank(c)
   }
   prod
}
```

Solutions:

1.

```
MPI_Recv(y,10000,MPI_INT,0,XTYPE,MPI_COMM_WORLD,&stat);
```

2.a latency

2.b bandwidth

3.a Load balance could be poor, say if the numbers are skewed toward their lower range.

3.b

```
if (me == nth-1) printf("%d\n",counts[0]);
```

3.c

```
pthread_create()
```

4.

```
uv <- function(cls,u,v)
{ rownums <- splitIndices(nrow(u),length(cls))
   # tack on row numbers to U
   u1 <- cbind(u,1:nrow(u))
   uchunks <- list()
   for (i in 1:length(cls)) {
      uchunks[[i]] <- u1[rownums[[i]],]
   }
   res <- clusterApply(cls,uchunks,chunkmul,v)
   Reduce(c,res)
}

chunkmul <- function(uchunk,v) {
   nr <- nrow(uchunk)
   nc <- ncol(uchunk)
   ncu <- nc - 1
   prod <- vector(length=nr)
   for (i in 1:nr) {
      urownum <- uchunk[i,nc]
      rng <- urownum:ncu
      prod[i] <- uchunk[i,rng] %*% v[rng]
   }
   prod
}
```