Name: ____________________________

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

1. (10) There is a algorithm for sorting known as *even/odd transposition sort*. At each iteration, each array element is swapped with either its left or right neighbor. (More commonly, contiguous chunks in the array are swapped.) If this algorithm were implemented in MPI, what MPI function would be especially appropriate?

2. (15) Consider the Thrust matrix transpose example in Section 6.7.1. State the contents of **dmap** just before the execution of the scatter operation in line 42.

3. (15) Consider the MPI mutual-outlinks example in Section 8.6.8. Suppose Line 46 were to be changed to

```
i = me;
```

(and Line 51 would become blank). Write a single line of code—and state between which two lines it should be inserted—that does a check for whether the program will still work, and prints out a warning if not.

4. (60) The code below computes the transpose of an n x n matrix (presumably very large), to be done in streaming mode in Hadoop. It is assumed that the input matrix is as in Section 9.10, stored one matrix row per line of the file, with a row numbers column prepended on the left and with spaces used as the delimiter. The output matrix will have the same form, except of course without the prepended column. So, for instance, an input matrix

```
1 1 2 6
2 3 4 8
3 0 5 12
```

will produce output

```
1    3    0
2    4    5
6    8    12
```

Fill in the blanks. **Assume that there will be only 1 reducer and 1 output file.**

mapper:

```
#!/usr/bin/env Rscript

con <- file("stdin", open = "r")
repeat {
   line <- readLines(con,n=1)  # read 1 line
   if (length(line) == 0) break
   tks <- strsplit(line,split=" ")
   tks <- tks[[1]]
   i <- as.integer(tks[1])
   elts <- tks[-1]
   n <- length(elts)
   for (j in 1:n) {
      newpos <-  blank (a)
      blank (b)
   }
}
```

reducer:

```
#!/usr/bin/env Rscript

# get n from command line
args <- commandArgs(T)
n <- as.integer(args[1])

con <- file("stdin", open = "r")
# make vector of n integers
arow <- integer(n)
for (blank (c)) {
   for (blank (d)) {
      line <- readLines(con,n=1)
      line <- strsplit(line,split="\t")
      line <- line[[1]]
      blank (e)
   }
   blank (f)
}
```

Solutions:

1. MPI_Sendrecv()

2. [0, 2, 4, 1, 3, 5]

3. Insert, say just before Line 62:

```
if (nnodes < n) printf("not enough MPI processes\n");
```

4.

mapper:

```
#!/usr/bin/env Rscript

# map/reduce pair inputs rows of a square matrix, and emits one record
# for each element of the matrix

con <- file("stdin", open = "r")
repeat {
   line <- readLines(con,n=1)  # read 1 line
   if (length(line) == 0) break
   tks <- strsplit(line,split=" ")
   tks <- tks[[1]]
   i <- as.integer(tks[1])
   elts <- tks[-1]
  # print(elts)
   n <- length(elts)
   for (j in 1:n) {
      newpos <- (j-1) * n + i
      # print(newpos)
      cat(newpos,"\t",elts[j],"\n")
   }
}
```

reducer:

```
#!/usr/bin/env Rscript

# get n from command line
args <- commandArgs(T)
n <- as.integer(args[1])

con <- file("stdin", open = "r")
arow <- integer(n)
for (lineout in 1:n) {
   for (j in 1:n) {
      line <- readLines(con,n=1)
      line <- strsplit(line,split="\t")
      line <- line[[1]]
      arow[j] <- line[2]
   }
   cat(arow,"\n")
}
```

4.

```
# arguments:
#    a:  adjacency matrix
#    lnks:  edges matrix; shared, nrow(a)^2 rows and 2 columns
#    counts:  numbers of edges found by each thread; shared

# in this version, the matrix lnks must be created ahead of time; since
# the number of rows is uknown a priori, one must allow for the worst
# case, nrow(a)^2 rows; after the run, the number of actual rows will be
# in counts[1,length(cls)]

getlinksthread <- function(a,lnks,counts) {
   require(parallel)
   nr <- nrow(a)
   # get my assigned portion of a
   myidxs <- getidxs(nr)
   myout <- apply(a[myidxs,],1,function(rw) which(rw==1))
   # myout[[i]] now lists the edges from node myidxs[1] + i - 1
   nmyedges <- Reduce(sum,lapply(myout,length))  # my total edges
   me <- myinfo$id
   counts[1,me] <- nmyedges
   barr()
   if (me == 1) {
      # use cumsum() to determine where each node will store its results
      # in lnks
      counts[1,] <- cumsum(counts[1,])
   }
   barr()
   # lnksidx will be the next row to write within lnks
   lnksidx <- if (me == 1) 1 else counts[1,me-1] + 1
   for (idx in myidxs)  {
      # corresponding index to idx within myout
      jdx <- idx - myidxs[1] + 1
      myoj <- myout[[jdx]]
      endwrite <- lnksidx + length(myoj) - 1
      if (!is.null(myoj)) {
         lnks[lnksidx:endwrite,] <- cbind(idx,myoj)
      }
      lnksidx <- endwrite + 1
   }
   0  # don't do expensive return of result
}

test <- function(cls) {
   mgrinit(cls)
   mgrmakevar(cls,"x",6,6)
   mgrmakevar(cls,"lnks",36,2)
   mgrmakevar(cls,"counts",1,length(cls))
   x[,] <- matrix(sample(0:1,36,replace=T),ncol=6)
   clusterExport(cls,"getlinksthread")
   clusterEvalQ(cls,getlinksthread(x,lnks,counts))
   print(lnks[1,counts[1,length(cls)],])
}
```