Name: _____

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. In order to get full credit, **SHOW YOUR WORK**.

1. (5) Fill in the blanks: The members of the **as()** family are _____ functions similar to _____ in C/C++.

2. (10) State three approaches from our course to speed up R code. Be terse, using at most three or four words in each of your three answers.

3. (15) In the discrete-event simulation code, suppose we wish to form a matrix of all pending events whose scheduled times fall into a given time period [a,b]. Fill in the blank:

```
timeab <- function(a,b) return(_____________________________________)
```

4. (20) The class **v10** will consist of vectors whose elements are only 1s and 0s. Fill in the blanks:

```
# constructor; vector has n 1s and 0s, with 1s at the given indices
v10 <- function(indices,n) {
   v <- rep(0,n)
   v[indices] <- _______________________________
   vo <- list(vec = v)
   class(vo) <- "v10"
   return(vo)
}

# returns the indices of the 1s in v, an object of class v10
getindices <- function(v) {
   return(_____________________________________)
}

# prints the given object, in terms of indices of the 1s
print.v10 <- function(v) {
   print(_____________________________________)
}

# returns a v10 object consisting of 1s everywhere va has a 1
# but vb doesn't; the two input vecs assumed equal length
"%-%" <- function(va,vb) {
   a <- getindices(va)
   b <- getindices(vb)
   i <- ___________________________________
   return(_____________________________________)
}
```

Here is an example of use:

```
> x <- v10(c(1,4),4)
> y <- v10(c(3,4),4)
> x
[1] 1 4
> x$vec
[1] 1 0 0 1
> getindices(x)
[1] 1 4
> z <- x %-% y
> z
[1] 1
> z$vec
[1] 1 0 0 0
```

5. (10) Consider the code at the top of p.5 of the Mertz handout. Replace the **assign()** call by a statement achieving the same result.

6. (10) Consider this interactive R session:

```
> u <- c(3,4,5,5,12,13)
> ____________(u,1)
[1] 3
> ____________(u,5)
[1] 12
> ____________(u,6)
[1] 13
```

Fill in the blanks, with the same answer in all three cases.

7. (10) Fill in the blanks in the folllowing R equivalent of Python's **reduce()** function:

```
reduce <- function(f,v) {
   rslt <- ________________________
   for (_____________________) _________________________
   return(rslt)
}
```

8. (20) The following code searches a matrix for the first row whose sum exceeds a given threshhold. It returns the index of the row, or 0 if no such row exists. Fill in the blanks.

```
# in matrix m, find index of first row having total >= rowtot; m
# is processed in blocks of blksz rows, sent one block at a time to
# cluster cls; return 0 if no such row exists; assumes blksz evenly
# divides nrow(m)

parfindfirst <- function(m,rowtot,cls,blksz) {
   for (i in 1:(nrow(m)/blksz)) {
      startrow <- 1 + (i-1) * blksz
      endrow <- i * blksz
      mblk <- m[startrow:endrow,,drop=F]
      rslt <- parApply(_____________________________________)
      rgt <- ________________________________________
      if (any(rgt)) return(_____________________________________)
   }
   return(0)
}
```

Solutions:

1. generic, casts

2. vectorization; writing parts of the code in C; parallel process

3.

```
sim$evnts[sim$evnts[,1] >= a & sim$evnts[,1] <= b]
```

4.

```
1
which(v$vec==1)
getindices(v)
setdiff(a,b)
v10(i,length(va))
```

5.

```
inf_vector <<- v
```

6.

```
"["
```

7.

```
v[1]
for (vi in v[-1]) rslt <- f(rslt,vi)
```

8.

```
parfindfirst <- function(m,rowtot,cls,blksz) {
   for (i in 1:(nrow(m)/blksz)) {
      startrow <- 1 + (i-1) * blksz
      endrow <- i * blksz
      mblk <- m[startrow:endrow,,drop=F]
      rslt <- parApply(cls,mblk,1,sum)
      # rslt <- apply(mblk,1,summ))
      rgt <- rslt >= rowtot
      if (any(rgt)) {
         return(startrow-1+which(rgt)[1])
      }
   }
   return(0)
}
```