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: If you believe that nothing needs to be placed into a blank, simply give **Nothing** as your answer in your file. If you do not answer at all, put 00 in your file.

1. (70) The following Hadoop code multiplies a vector **x** by a (presumably very large) matrix **a**. The input matrix has prepended to it a column of row numbers. The vector **x** is input by executing code in a file **x.py**.

So, if the input matrix is

```
0 1 2 0
1 5 8 -4
2 0 0 3
```

and the contents of **x.py** are

```
x = [5,12,13]
```

then the final output will be

```
29
69
39
```

Note: No row/element numbers in the final output. Don't worry about leading blanks in the output.

Fill in the blanks. You may find the Python **len()** function useful; it returns the length of a Python list (array), so that for instance **len(x)** is 3 in the above example. Also, the **int()** function is like **atoi()** in C.

axmap.py:

```
#!/usr/bin/env python

from x import x  # input x from file x.py
import sys

for line in sys.stdin:
    tks = line.split()
    rownum = tks[0]
    row = tks[1:]
    sum = 0
    for i in range(BLANKa):
        sum += BLANKb
    print BLANKc
```

axred.py:

```
#!/usr/bin/env python

import sys

for line in sys.stdin:
    line = line.strip()
    tks = line.split('\t')
    print BLANKd
```

2. (30) Fill in the blanks in the Snow code below, which finds the unique elements of an array in parallel. The built-in R function **unique()** works like this:

```
> x <- sample(1:8,10,replace=T)
> x
  [1] 4 7 3 1 1 2 3 3 2 8
> unique(x)
 [1] 4 7 3 1 2 8
```

Code:

```
# not claimed efficient, and
# no guarantee of ordering in result

parunique <- function(cls,x) {
    parts <- clusterSplit(cls,1:length(x))
    xparts <- lapply(parts,function(part) x[part])
    tmp <- clusterApply(cls,xparts,BLANKa)
    tmp <- Reduce(BLANKb)
    BLANKc
}
```

Solutions:

1.

axmap.py:

```python
#!/usr/bin/env python

from x import x  # input x from file x.py
import sys

for line in sys.stdin:
    tks = line.split()
    rownum = tks[0]
    row = tks[1:]
    sum = 0
    for i in range(len(row)):
        sum += int(row[i]) * x[i]
    print '%s\t%s' % (rownum, sum)
```

axred.py:

```python
#!/usr/bin/env python

import sys

for line in sys.stdin:
    line = line.strip()
    tks = line.split('\t')
    print tks[1]
```

2.

```r
# not claimed efficient, and no guarantee of ordering in result

parunique <- function(cls,x) {
    parts <- clusterSplit(cls,1:length(x))
    xparts <- lapply(parts,function(part) x[part])
    tmp <- clusterApply(cls,xparts,unique)
    tmp <- Reduce(c,tmp)
    unique(tmp)
}
```