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.

Unless otherwise stated, give numerical answers as expressions, e.g. $\frac{2}{3} \times 6 - 1.8$. Do NOT use calculators.

1. (40) Suppose we were to write an OpenMP version of the dice simulation example of the Python **multiprocessing** module. Say we store our grand total in a global variable **tot**, with **count** storing the thread's individual count. Show how to efficiently write the OpenMP version of

```
totlock.acquire()
tot.value += count
totlock.release()
```

2. (30) Consider the Python **multiprocessing** example of Quicksort, using the **Queue** class. Suppose the original array to be sorted was (12,5,13,6,8,10,2,21,20,15). When the work item (i,j,2) is placed into the queue, what will be the values of i and j? Note: The code in **separate()** is not quite right, but assume it works correctly, which is to rearrange the given range within **xc** so that all the elements smaller than **xc**[**low**] are moved to the left of that element, and all the ones larger than that element are moved to its right, with the return value **last** being the final resting place of **xc**[**low**].

3. (30) The function **allgt(x,y,n)** below, to run in an OpenMP context, returns 1 (i.e. True) if all elements of **x** are greater than their counterparts in **y**. Each of the arrays **x** and **y** is of length **n**. (In this implementation, no effort is made to do no further checking after encountering a False case.) The function is to be called from within an OpenMP **parallel** block. Fill in the blanks.

```
int allgt(x,y,n)
{  int all,i;
   #pragma omp ________________________________________
   for (i = 0; i < n; i++) ______________________________
   return all;
}
```

Solutions:

1.

```
#pragma omp atomic
tot += count;
```

2. The first call to **separate()** will rearrange the array to (5,6,8,10,2,12,13,21,20,15) and return 5, and (6,9,1) will be put in the queue. The next thread will work on that, and put (7,9,2) in the queue.

3.

```
int allgt(x,y,n)
{  int all,i;
   #pragma omp for reduction(&&:all)
   for (i = 0; i < n; i++) all &&= (x[i] > y[i]);
   return all;
}
```