Written Problems. due Mar 24 (Wed), 6.30 pm.

1. **(Running Times)** Write down the number of times the following Python programs print `hello` (as a function of the input n) using the $O(\cdot)$ notation.

```
def hello3(n):
    if n <= 0: return 0
    else:
        print "hello"
        hello3(n/2)
```

```
def hello4(n):
    if n <= 0: return 0
    else:
        m = n/2
        hello4(m)
        for i in range(m):
            print "hello"
        hello4(m)
```

2. **(Counting Inversions)** The following is a variant of the subroutine we used for counting the number of inversions in Lecture 9:

```
def sortcount(a):
    """returns (count, sorted list)"""
    ans, aux = 0, []
    print "input ", a
    if len(a) <= 1:
        ans, aux = 0, a
    else:
        mid = len(a)/2
        (left, ans1) = sortcount(a[:mid])
        (right, ans2) = sortcount(a[mid:])
        ans3 = crosscount(left,right)
        ans, aux = ans1+ans2+ans3, merge(left, right)
    print "ans ", ans
    return (aux, ans)
```

The following is the output of `sortcount([4,3,2,1])`. Fill in the four missing values.

```
input  [4, 3, 2, 1]
input
input
ans  0
input  [3]
ans  0
ans  1
input
input  [2]
```

```
ans  0
input  [1]
ans  0
ans  1
ans
```

Programming Problems. due Mar 25 (Thu), 11.59 pm.

3. **(Large Contiguous Sums)**

   We revisit a problem from Homework 1:

   Given an array of n integers $a_0, a_1, \ldots, a_{n-1}$ positive and negative, such as $-1, 3, 2, -7, 4, 2, -2, 3, -1$, you want to find the largest sum of contiguous integers; in this case it would be $4 + 2 - 2 + 3 = 7$.

   (The empty set has sum 0.)

   Download `ps6_maxsums.zip`.

   Implement a divide-and-conquer algorithm (the function `maxsums_dc(a)`) that computes the largest sum of contiguous integers in time $O(n \log n)$. You should make sure that the "combine" step runs in $O(n)$ time.

   Run `test_maxsums2.py` to help determine if your solutions work. The entire test suite must complete running in under 0.3 secs on `owl.cs.qc.edu`. Submit `maxsums2.py` by copying the file to your `submit` directory.

   (HINT. You should use `partialsums` in the "combine" step.)

4. **(Fraud Detection)** Textbook Chapter 5, Exercise 3

   Download `ps6_fraudcheck.zip`. Complete the implementation of the fraud detection algorithm in `fraudcheck.py` and submit `fraudcheck.py` by copying the file to your `submit` directory.

   Run `test_fraudcheck.py` to help determine if your solutions work. The entire test suite must complete running in under 0.1 secs on `owl.cs.qc.edu`.

   (HINT. Consider $n = 8$ and the list of account numbers is $[1, 3, 3, 3, 2, 2, 2, 4]$. If we recursively call the algorithm on the first and the second halves of the list, then both calls will return TRUE, whereas the final answer should be FALSE. As with counting the number of inversions, we want the algorithm to return some additional information (apart from TRUE/FALSE) so that the COMBINE step will be easier. Note that you cannot sort the list in general since you are only limited to testing equivalence.)