Written Problems. due Feb 10 (Wed), 6.30 pm.

1. **(Running Times)** Write down the running times of the following Python programs (as a function of the input n); you may use the $O(\cdot)$ notation.

```
def hello1(n):
    i=0
    while i<n:
        i += 1
        print "hello"
    j=0
    while j<i:
        j += 1
        print "hello"
```

```
def hello2(n):
    i=0
    while i<n:
        i += 1
        print "hello"
        j=0
        while j<i:
            j += 1
            print "hello"
```

2. **(Asymptotic Growth)** Rank the following functions by increasing order of growth; that is, find an arrangement $g_1, g_2, \ldots, g_{11}$ of the functions satisfying $g_1 = O(g_2)$, $g_2 = O(g_3)$, …, $g_{10} = O(g_{11})$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. All the logs are in base 2.

$$\binom{n}{2}, \quad 3^n, \quad n^{100}, \quad 1/n, \quad 373n^2 + 700n$$

$$10^{100}n, \quad 3^{\sqrt{n}}, \quad 1/5, \quad n \log n, \quad 100 \log n.$$

3. **(Stress Testing Glass Jars – Extra Credit)** Textbook Chapter 2 Exercise 8.

Programming Problems. due Feb 11 (Thu), 11.59 pm.

1. **(Large Contiguous Sums)** 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.)

   We saw in class an algorithm to solve the problem in time (n^3). In this problem, we want to design that an algorithm that solves the problem in time $O(n^2)$.

   Download `ps1_maxsums.zip` and look at `maxsums1.py`.

   (a) First, implement `partialsums`, which computes a list containing the partial sums of the input. For instance, if the input is $1, -2, 3$, `partialsums` should return the list $1, 1+(-2), 1+(-2)+3$, i.e., $1, -1, 2$. The algorithm should run in $O(n)$ time.

(b) Next, implement `maxsums`, which returns the largest sum of contiguous integers. In the first example, `maxsums` should simply return 7. The algorithm should run in $O(n^2)$ time.

(c) Finally, implement `maxsums_list`, which returns the list of contiguous integers with the largest sum. In the first example, `maxsums_list` should return the list $4, 2, -2, 3$. The algorithm should run in $O(n^2)$ time.

Make sure that your algorithms return the correct answers for the test suite in `test_maxsums.py`. Submit `maxsums1.py` by copying the file to your `submit` directory on `owl`.

2. **(Merging Sorted Lists)** Recall that we presented pseudocode for an algorithm `merge` to merge sorted lists in class, and showed how it can be used to implement the `mergesort` divide-and-conquer algorithm for sorting a list of n numbers.

   Download `ps1_merge.zip`.

   (a) Write `merge(a, b)`. It should run in time $O(n)$. Submit `mergesort.py` by copying the file to the `submit` subdirectory. Make sure it passes all the tests in `test_merge.py`.

   (b) Determine experimentally the running time of `mergesort(a)` by running it with different sized lists using the `mergetest(n)` subroutine. Note that `mergetest(n)` simply runs `mergesort` on the list $[n, n-1, \ldots, 2, 1]$.

   Let $T(n)$ denote the time it takes to complete `mergetest(n)`. We know that $T(n) = O(n \log n)$. This means that if the size of the input increases by a factor of 2, the running time increases by roughly a factor of 2 (a little more, because of the $\log n$ factor).

   Execute `mergetest(n)` for $n = 1000, 2000, 4000, 10000, 20000, 40000$ and write down the running times (as commented code) in `mergesort.py`.