1. Ch 2 problem 2.

10e10 ops/second \* 3600 seconds/hour = 3.6e13 ops/hour

a) n^2<=3.6e13; n<=6e6

b) n^3<=3.6e13; n<=33019

c) 100n^2<=3.6e13; n<=6e5

d) nlogn<=3.6e13; n<=1.29095e12 (From solving xlogx-3.6e13=0 in Mathematica)

e) 2^n<=3.6e13; n<=log(3.6e13)=45

f) 2^(2n)<=3.6e13; 2n<=log(3.6e13)=45; n<=22

2. Ch 2 problem 4

g1<g4<g3<g5<g2<g6<g7

3. Ch 2 problem 6

a) In general, the number of iterations these two loops will make together should be

equal to (1/2)(n^2-n). Now, during every loop iteration we must add up some linear

number of integers; (1/2)(n^2-n). is in O(n^2), and multiplying this by n gets (1/2)(n^2-n)(n) which is O(n^3). Thus, f(n)=n^3.

b) The term (1/2)(n^2-n)(n) we derived above is Omega(n^3) inherently;

3. Ch 2 problem 8

a)If we can divide n into intervals, the number of which grows in a slower-than-linear fashion, we will have solved the problem. I had originally wanted to use a logarithmic function, but upon further reflection (or, reading part b) it makes sense to divide the input n into intervals of size equal to some power of n less than 1.

For example, let's divide n into sqrt(n) intervals. Starting from the beginning of the first interval and iterating in steps of sqrt(n), we find the first interval at which the jar breaks. At this point we've found the interval in which the highest safe rung exists, an interval of size (at most) sqrt(n). We can iterate through this linearly.

In the worst case scenario the first iteration took sqrt(n) steps and the second iteration took another sqrt(n) steps, for a total of 2sqrt(n) steps, so our algorithm is O(sqrt(n)). This is better than O(n), which was the condition we were going for.

b) In the above case we got O(f(n)) where f(n)=sqrt(n). sqrt(n) can be written as n^(1/2), and since in a) k=2 we can see that this is actually n^(1/k). By increasing k we get smaller sub-intervals and reduce our iterations significantly.

However, we cannot use the same algorithm as above, though we would in the case of two jars. Instead, we can take advantage of a recursive procedure by subdividing into n^(1/k) intervals, finding the “highest safe rung” in that set, decrementing k by one, and repeating until k=2. Once k=2 we perform the same thing as in a.

Looking at this algorithm, we start by performing (worst case) n^(1/k) steps. We then get a new set, in which we perform at worst n^(1/(k-1)) steps. This takes k-2 or O(k) steps. Adding these k steps on to our procedure, we see that the number of steps is something like kn^(1/k) +n^(1/k), or n^(1/k)(k+1), which is O(kn^(1/k)+k)=O(2kn^(1/k)). It is clear that f(n,k-1)<f(n,k), as are its derivatives WRT k and n, so by L'Hospital's Rule it is inevitable that the limit in the problem=0.