CSCI 5454 **Prof. Aaron Clauset**
Problem Set 1, due Feb. 4 **Spring 2013, CU-Boulder**

This is a long problem set. I encourage you to discuss the problems with each other, but remember that you *must* write up your solutions independently *and* list the names of your collaborators. A footnote will suffice.

Note: the programming problems (9 and 10) will take considerable time. Plan accordingly.

1. (10 pts total) Acme Corp. has asked Professor Flitwick to develop a faster algorithm for their core business. The current algorithm runs in $f(n)$ time. (For concreteness, assume it takes $f(n)$ microseconds to solve a problem of size exactly n.) Flitwick believes he can develop a faster algorithm, which takes only $g(n)$ time, but developing it will take t days. Acme only needs to solve a problem of size n once. Should Acme pay Flitwick to develop the faster algorithm or should they stick with their current algorithm? Explain.

   (a) (5 pts) Let $n = 41$, $f(n) = 1.99^n$, $g(n) = n^3$ and $t = 17$ days.

   (b) (5 pts) Let $n = 10^6$, $f(n) = n^{2.00}$, $g(n) = n^{1.99}$ and $t = 2$ days.

   (Hint: start with the *largest* n Acme can solve with Flitwick after t days of waiting.)

2. (5 pts) Show (prove) that for any real constants a and b, where $b > 0$, the asymptotic relation $(n + a)^b = \Theta(n^b)$ is true.

3. (5 pts total) Using the mathematical definition of Big-O:

   (a) (3pts) Is $2^{n\,k} = O(2^n)$ for $k > 1$?

   (b) (2pts) Is $2^{n+k} = O(2^n)$, for $k = O(1)$?

4. (10 pts) In the classic version of Quicksort, the pivot is always chosen as the last element in the input array. When the pivot is compared to another element, we can use any valid comparison operator. Thus, "sorting" generalizes to any set of inputs over which we can mathematically define comparisons.

   Use this version of the algorithm to sort the following *functions* by order of asymptotic growth such that the final arrangement of functions $g_1, g_2, \ldots, g_{12}$ satisfies the ordering constraint $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, $\ldots$, $g_{11} = \Omega(g_{12})$.

n	n^2	$(\sqrt{2})^{\lg n}$	$2^{\lg^* n}$	$n!$	$(\lg n)!$	$\left(\frac{3}{2}\right)^n$	$n^{1/\lg n}$	$n \lg n$	$\lg(n!)$	e^n	1

(a) (**10 pts extra credit**) For each branch of the recursion tree, identify the chosen pivot element; for each level of the recursion tree and after all pivot elements at that level have been moved into their final location, give the global array ordering. (See Section 2.3 of Lecture 1 as an example.)

(Hint: it may be easier to work out the final ordering by hand and then backtrack through the Quicksort operations to produce the recursion tree.)

(b) (10 pts) Give the final sorted list and identify which pair(s) functions $f(n), g(n)$, if any, are in the same equivalence class, i.e., $f(n) = \Theta(g(n))$.

5. (18 pts total) Consider the following recursive function `f`, which takes an integer argument `n` and returns some other integer `f(n)`:

```
f (n) {
   if (n==0) return 3;
   else if (n==1) return 5;
   else {
      val = 3*f(n-1);
      val = val - 2*f(n-2);
      return val;
   }
}
```

(a) (2 pts) Write down the recurrence relation for the *value* returned by `f(n)`; identify the base cases.

(b) (6 pts) Using the method of characteristic polynomials, solve the value recurrence relation exactly.

(c) (2 pts) Show (prove) by induction that your solution is correct.

(d) (2 pts) Under the `RAM` model of computation, write down the recurrence relation for the *running time* of `f(n)`.

(e) (3 pts) In three sentences or less, explain why $O(2^n)$ is a trivial upper bound on the running time.

(f) (6 pts) Using the method of characteristic polynomials, solve the time recurrence relation for a *tight* upper bound. (Hint: $O(2^n)$ is not tight.)

6. (12 pts total) Solve the following three recurrence relations using the method specified. Show all your work.

(a) (3 pts) $T(n) = T(n-1) + n$ by "unrolling" (tail recursion).

(b) (3 pts) $T(n) = 2T(n/2) + n^3$ by the Master method.

(c) (6 pts) $T(n) = 2T(n/2) + n^3$ by the recurrence tree method; include a diagram of your recurrence tree.

7. (15 pts total) Professor Snape has n computer chips that are supposedly both identical and capable of testing each other's correctness. Snape's test apparatus can hold two chips at a time. When it is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the answer of a bad chip cannot be trusted. Thus, the four possible outcomes of a test are as follows:

Chip A says	Chip B says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

(a) (5 pts) Show (prove) that if $n/2$ or more chips are bad, Snape cannot necessarily determine which chips are good using any strategy based on this kind of pairwise test. Assume that the bad chips can conspire to fool Snape.

(b) (5 pts) Consider the problem of finding a single good chip from among the n chips, assuming that more than $n/2$ of the chips are good. Show (prove) that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.

(c) (5 pts) Show (prove) that the good chips can be identified with $\Theta(n)$ pairwise tests, assuming that more than $n/2$ of the chips are good. Give and solve the recurrence that describes the number of tests.

8. (5 pts) Professor Septima Vector thinks she has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three (possibly empty) sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Vector claims that any three keys $a \in A$, $b \in B$ and $c \in C$ must satisfy $a \leq b \leq c$. Give a *smallest possible* counterexample to the professor's claim.

9. (15 pts) Implement the skip list data structure.
No credit if you don't include your source code at the end of your file.

10. (10pts) Define the running time as the number of atomic operations (lookups), not clock time, used to complete a function call. Now, use your implementation from 9. to conduct a numerical experiment that verifies the asymptotic running times of the (i) insert, (ii) delete and (iii) find operations.

    The deliverable here is a single figure (like the one below) showing how the number of operations T grows as a function of n. Show two trend lines of the form $O(\log n)$ that bound from above and below your numerical results. Label your axes and the trend lines. Include a clear and concise description (1-2 paragraphs) of exactly how you ran your experiment.

    No credit will be given if you don't label your axes and trend lines.

    Hint 0: Define an "atomic operation" to be executing any simple mathematical operation (+,-,* or /), following a pointer, assigning or copying a value to a variable, comparing the values of two variables, or calling a random number generator.

    Hint 1: To count these operations, you'll need to implement some measurement code inside your skip list data structure to count and report the number of operations your code uses as it complete a function call. (Do not include the operations of your measurement code in your counts!)

    Hint 2: To clearly show the asymptotic behavior, choose at least 10 values of n spaced out logarithmically over several factors of 10, e.g., $\log_2 n = \{4, 5, ...13\}$.

    Hint 3: Because skip lists are a randomized data structure, each particular measurement is itself a random variable. Thus, for each value of $T(n)$, compute the *average* over several (> 10) *independent* measurements or "trials." The greater the number of trials you average over, the smoother the trend line. A smooth trend line is your goal. Note that for a trial to be independent, you cannot reuse a data structure that has previously been perturbed by an experiment.

    Hint 4: If your figure doesn't show *smooth* $O(\log n)$ behavior (like the figure below) over a wide range of n (like the figure below), there's almost surely a bug in your implementation.

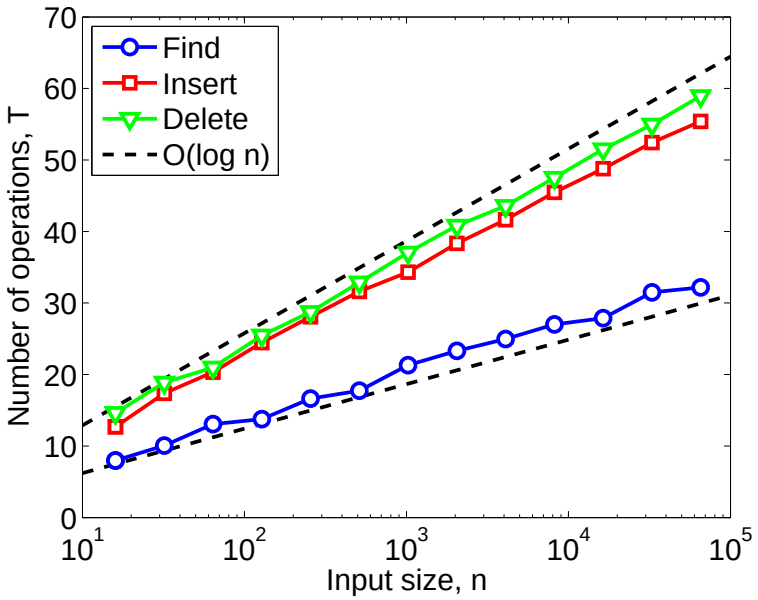

Figure 1: An example of what your skiplist results should look like. (Note: not real data.)