Homework Assignment #1 (revised)

CS 410/584 Algorithm Design & Analysis: Spring 2011

This assignment is due Wednesday, 7 April, at the beginning of class. You should work alone on this assignment. However, you are free to discuss the problems on the class mailing list. (Or you can send us email questions directly; please put "CS 584" at the beginning of the subject line.)

Reading: You should review Chapters 1-14. Topics covered in detail in class correspond to Section 3.1 (order notation) and Ch. 6 (heapsort). Read Chapter 15 (15.4 is Longest Common Subsequence).

Office Hours: I will not be at office hours on Tuesday, 5 April. I will have alternate office hours on Wednesday, 6 April, 1:30 – 3:30p.

Note: On any homework exercise where you are asked to give an algorithm, in addition to describing the algorithm, you must also provide an English description of how it works ***and*** at least one example execution on a non-trivial input.

Exercises:

1A (10 points): Is 3^{n+2} in $O(3^n)$? Is 3^{2n} in $O(3^n)$? Explain.

1B (10 points): Illustrate Build-Max-Heap on the array
A = [6, 2, 18, 11, 60, 20, 7, 21, 10]
Show the heap after each call from Build-Max-Heap to Max-Heapify. (See Figure 6.3.)

1C (15 points): The operation Heap-Delete(A, *i*) that deletes item in node *i* from heap A. Give an algorithm for Heap-Delete where time complexity in $O(\lg n)$ for a heap of size *n*.

1D (20 points): Prove or disprove:
(a) $f(n)$ in $O(g(n))$ implies $g(n)$ in $O(f(n))$.
(b) $f(n)$ in $O(g(n))$ implies $O(f(n) + g(n)) = O(g(n))$
(c) $f(n)$ in $O(g(n))$ implies $2^{f(n)}$ in $O(2^{g(n)})$.
(d) $f(n)$ in $O(g(n))$ implies $\lg f(n)$ in $O(\lg g(n))$.

1E (10 points): This question is about sorting multiple lists. You have *k* lists, each with *n* records in it. The keys for the records are integers in the range 1 to *m*. (For example, each list might contain records for students in one department, where the key for each student record is the number of courses the student has taken.) Give an algorithm to sort all the lists in total worst-case time of $O(kn + m)$. **Note:** Sorting each list separately will not give you the desired time bound. **Hint:** Review sorting methods.

1F (15 points, 584 students only): Modify the **Freq** algorithm that I passed out in class so that it has O(n log(n/k)) worst-case time complexity, where n is the size of collection S and k is the frequency of the most frequent element m (the *mode*). You may assume a function **Median**(S) that returns the middle element of S (in terms of sort order) and runs in O(n) time. Explain why your algorithm has the claimed time complexity.