CS314: Algorithms

Homework 1, due Monday, Feb. 2 at the beginning of class

This homework will be submitted in written format. Remember, you can submit homework in pairs; just be sure both names are written on all pages submitted.

Also recall that when asked to design an algorithm, you must also include a proof of correctness and running time for that algorithm. In particular, if the problems asks you to design an algorithm with a particular run time, you must still analyze *your* algorithm to prove that it meets that running time.

Required Problems

1. Consider the following sorting algorithm:

```
STUPIDSORT(A[0..n − 1]):
  if n = 2 and A[0] > A[1]
      swap A[0] and A[1]
  else if n > 2
      m ← ⌊2n/3⌋
      STUPIDSORT(A[0..m − 1])
      STUPIDSORT(A[n − m..n − 1])
      STUPIDSORT(A[0..m − 1])
```

   (a) Prove that STUPIDSORT actually sorts its input.

   (b) State a recurrence (including the base cases - there are two of them!) for the number of comparisons executed by STUPIDSORT.

   (c) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?

2. A *subsequence* of a sequence A consists of a (not necessarily contiguous) collection of elements of A, arranged in the same order as they appear in A.

   Describe and analyze a **simple** recursive algorithm to compute, given two sequences A and B, the length of the *longest common subsequence* of A and B. For example, given the strings `ALGORITHM` and `ALTRUISTIC`, your algorithm would return 5, the length of the longest common subsequence `ALRIT`.

3. Chapter 5, Exercise 2 from the textbook:

   Recall the problem of finding the number of inversions. As in the text, we are given a sequence of numbers $a_1, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$.

   We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is two sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ time algorithm to count the number of significant inversions.

4. Chapter 5, Exercise 6 from the textbook:

   Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the roots are all distinct. A node v of T is a *local minimum* if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

   You are given such a complete binary tree T, but the labeling is only specified in the following implicit way: for each node v, you can determine the value x_v by *probing* the node v. Show how to find a local minimum of T using $O(\log n)$ probes to the nodes of T.

5. Extra Credit problem: Chapter 5, Exercise 7 from teh textbook:

   Now suppose you're given an $n \times n$ grid graph G. (An $n \times n$ grid graph is just the adjacency graph of an $n \times n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural number (i, j) where $1 \leq i \leq n$ and $1 \leq j \leq n$; the nodes (i, j) and (k, l) are joined by an edge if and only if $|i - k| + |j - l| = 1$..)

   We use some of the terminology of the previous question. Again, each node v is labeled by a real number x_v; you may assume that all these labels are distinct. Show how to find a local minimum of G using only $O(n)$ probes to the nodes of G. (Note that G has n^2 nodes.)