Homework 3 Out: March 2, 2017
Due: March 23, 2017

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail. As always, try to make your answers as clear and concise as possible.

1. Prove that there is a unique minimum spanning tree on a connected undirected graph when the edge weights are unique.

2. Consider the following scheduling problem: we have two machines, and a set of jobs $j_1, j_2, j_3, \ldots, j_n$ that we have to process one at a time. To process a job, we place it on a machine. Each job j_i has an associated running time r_i. The load on the machine is the sum of the running times of the jobs placed on it. The goal is to minimize the completion time, which is the maximum load over all machines.

   Suppose we adopt a greedy algorithm: each job j_i is put on the machine with the minimum load after the first $i-1$ jobs. (Ties can be broken arbitrarily.) Show that this strategy yields a completion time within a factor of 3/2 of the best possible placement of jobs. (Hint: Think of the best possible placement of jobs. Even for the best placement, the completion time is at least as big as the biggest job, and at least as big as half the sum of the jobs. You may want to use both of these facts.) Give an example showing that the bound 3/2 is tight for this algorithm.

3. Consider an algorithm for integer multiplication of two n-digit numbers where each number is split into three parts, each with $n/3$ digits.

   (a) Design and explain such an algorithm, similar to the integer multiplication algorithm (i.e., Karatsuba's algorithm) presented in class. Your algorithm should describe how to multiply the two integers using only six multiplications on the smaller parts (instead of the straightforward nine).

   (b) Determine the asymptotic running time of your algorithm. Would you rather split it into two parts (with three multiplications on the smaller parts) as in Karatsuba's algorithm?

   (c) Suppose you could use only five multiplications instead of six. Then determine the asymptotic running time of such an algorithm. In this case, would you rather split it into two parts or three parts?

   (d) Challenge problem– this is optional, and not worth any points. Solving it will simply impress the instructor. Find a way to use only five multiplications on the smaller parts. Can you generalize to when the two initial n-digit numbers are split into k parts, each with n/k digits? Hint: also consider multiplication by a constant, such as 2; note that multiplying by 2 does not count as one of the five multiplications. You may need to use some linear algebra.

4. Suppose we have an array A containing n numbers, some of which may be negative. We wish to find indices i and j so that

$$\sum_{k=i}^{j} A[k]$$

   is maximized. Find an algorithm that runs in time $O(n)$.

5. A challenge that arises in databases is how to summarize data in easy-to-display formats, such as a histogram. A problem in this context is the minimal imbalance problem. Again suppose we have an array A containing n numbers, this time all positive, and another input k. Consider k indices $j_1, j_2, \ldots j_k$ that partition the array into $k+1$ subarrays $A[1, j_1], A[j_1+1, j_2], \ldots, A[j_k+1, n]$. The weight $w(i)$ of the ith subarray is the sum of its entries. The *imbalance* of the partition is

$$\max_i \left| w(i) - \left(\sum_{\ell=1}^{n} A[\ell] \right) / (k+1) \right| .$$

   That is, the imbalance is the maximum deviation any partition has from the average size.

   Give an algorithm for determining the partition with the minimal imbalance given A, n, and k. (This corresponds to finding a histogram with k breaking points, giving $k+1$ bars, as close to equal as possible, in some sense.)

Explain how your algorithm would change if the imbalance was redefined to be

$$\sum_i \left| w(i) - \left(\sum_{\ell=1}^{n} A[\ell] \right) / (k+1) \right|.$$

6. Suppose we want to print a paragraph neatly on a page. The paragraph consists of words of length $\ell_1, \ell_2, \ldots, \ell_n$. The maximum line length is M. (Assume $\ell_i \leq M$ always.) We define a measure of neatness as follows. The extra space on a line (using one space between words) containing words ℓ_i through ℓ_j is $M - j + i - \sum_{k=i}^{j} \ell_k$. The penalty is the sum over all lines **except the last** of the **cube** of the extra space at the end of the line. This has been proven to be an effective heuristic for neatness in practice. Find a dynamic programming algorithm to determine the neatest way to print a paragraph. Of course you should provide a recursive definition of the value of the optimal solution that motivates your algorithm.

   For this problem, besides explaining/proving your algorithms as for other problems on the set, you should also code up your algorithm in python to print an optimal division of words into lines. Call the program neatness.py. The output should be the text split into lines appropriately, and the numerical value of the penalty. You should assume that a *word* in this context is any contiguous sequence of characters not including blank spaces.

   After coding your algorithm, download the text file containing a review of the Season 1 Buffy DVD posted at `http://people.cs.georgetown.edu/jthaler/BuffyReview.txt`, which was apparently written by Ryan Crackell for the Apollo Guide. Determine the minimal penalty for nearly printing the review, for the cases where $M = 40$ and $M = 72$.

7. Another type of problem often suitable for dynamic programming is problems on tree graphs. For example, suppose we have a graph $G = (V, E)$ that is a tree with a root r. Derive a recursion to find the size of the maximum-sized independent set of of G. (An independent set is a subset of graph vertices, such that no two have an edge between them.) For full credit, show that you can find the size of the maximum-sized independent set and the set itself in linear time.