CSE 101 - ALGORITHMS - SUMMER 2000
Midterm

Wednesday, July 19th.

Problem 1: Sorting

(10 points) Given an array of real numbers S, find a pair of numbers x, y in S that *minimizes* $|x + y|$. Give your best algorithm for this problem, argue that it is correct and then analyze it.

For partial credit you can write an $O(n^2)$ algorithm.

Proof: A solution in $O(n^2)$ time would be to generate all pairs x, y, to calculate $|x + y|$ and to select those that give the minimum.

The $O(n \log n)$ solution based on sorting that I expected is the following:

MIN-SUM(S)

1 sort S by the *abolute value* of its elements

2 $min \leftarrow \infty$

3 **for** $i \leftarrow 1$ **to** $length(S) - 1$

4   **if** $|S[i] + S[i+1]| < min$ **then** $\{min \leftarrow |S[i] + S[i+1]|;\ x \leftarrow S[i];\ y \leftarrow S[i+1]\}$

5 **return** x, y.

The idea is to sort the numbers by their absolute value. This can be easily done by modifying any sorting algorithm such that to compare $|S[i]|$ with $|S[j]|$ instead of $S[i]$ with $S[j]$; the running time remains the same.

This algorithm is correct because if two numbers x and y minimize the expression $|x+y|$, then one of the following cases can appear:

1. x, y are both positive; then x, y must be the smallest two positive numbers, so they occur at consecutive positions in the sorted array.

2. x, y are both negative; then x, y must be the largest two negative numbers, so they are consecutive in the sorted array.

3. One of x, y is positive and the other is negative; then $|x + y| = ||x| - |y||$; but the expression $||x| - |y||$ is minimized by two consecutive elements in the sorted array.

Therefore, the expression $|x + y|$ is minimized by two elements x, y which occur on consecutive positions in the array sorted by absolute value of numbers.

The running time of this algorithm is given by the running time of sorting in step 1, because the other steps take linear time. Hence, the running time is $O(n \log n)$.

Problem 2: Dynamic Programming

(10 points) Write a dynamic programming algorithm for the following problem:

LARGEST-SQUARE

INPUT: A matrix $n \times m$ of bits, A.

OUTPUT: A largest squared submatrix of A containing only bits 1.

A bit can be either 0 or 1. A squared matrix is a matrix $k \times k$. Output the answer as a triple (i, j, k), where (i, j) are the coordinates of the upper left-hand corner of the squared submatrix and k is its dimension.

You algorithm should run in $O(n \cdot m)$ time. For partial credit, your algorithm can be slower.

Proof: Let $l[i, j]$ be the length of the largest square whose upper left-hand corner is positioned at coordinates (i, j) in A. To simplify writing, suppose that $l[n + 1, j]$ and $l[j, m + 1]$ are 0. Then one can see that the following recurrence holds:

$$l[i, j] = \begin{cases} 0, \text{ if } A[i, j] = 0 \text{ or } i > n \text{ or } j > m \\ 1 + \max\{l[i + 1, j + 1], l[i, j + 1], l[i + 1, j]\}, \text{ if } A[i, j] = 1 \end{cases}$$

A dynamic programming algorithm can be easily devised now, either using recursion with memoization or using the iterative method. The following one, for example, uses the iterative method (suppose that $l[1..n+1, 1..m+1]$ initially contains only 0):

LARGEST-SQUARE(A)

$K \leftarrow 0$
for $i \leftarrow n$ **downto** 1 **do**
    for $j \leftarrow m$ **downto** 1 **do**
        if $A[i, j] = 1$ **then** $\{l[i, j] \leftarrow 1 + \min\{l[i + 1, j + 1] + l[i, j + 1] + l[i + 1, j]$
                                if $K < l[i, j]$ **then** $\{K \leftarrow l[i, j];\ I \leftarrow i;\ J \leftarrow j\ \}\ \}$
output (I, J, K).

The running time of this algorithm is obviously $O(n \cdot m)$.