Algorithms Core 15-750 PROBLEM SET # 1 Spring 1999

DUE: 1:00pm Wednesday 27 January 1999

1 Crossover point for Strassen's Algorithm

In class we presented Strassen's algorithm for multiplying n by n matrices in $O(n^{\log 7})$ time. Assume that all the work is in the arithmetic operations.

a) Compute the smallest k such that Strassen does fewer arithmetic operations than the naive matrix multiplications on matrices of size 2^k by 2^k.

b) Compute the smallest even integer n such that one level of Strassen's algorithm followed by naive matrix multiplication does fewer arithmetic operations than the naive matrix multiplications on matrices of size n by n. (The algorithm is: take two n by n matrices, split each into four $n/2$ by $n/2$ matrices, organize the multiplications of the blocks according to Strassen. For each of the smaller products do the computation in the naive, direct way, without calling Strassen recursively.)

c) Compute the smallest integer n divisible by 4 such that two levels of Strassen followed by naive matrix multiplication does fewer arithmetic operations than one level of Strassen followed by naive matrix multiplication on matrices of size n by n.

2 Space for Strassen's Algorithm

In class we presented Strassen's algorithm for multiplying n by n matrices in $O(n^{\log 7})$ time. Show how to organize his algorithm so that it will use at most $2n^2$ space for the input, n^2 space for the output, and at most n^2 extra work space.

In general this problem can be messy because it needs space for the recursion stack as well as the space for the pointers to maintain the matrices as quad-trees. You can assume that the only space needed is to store scalars. Further assume that you may not overwrite the input, but you may use the output space for scratch space.

3 Counting in a Redundant system

Consider the following *redundant ternary number system*. A number is represented by a sequence of *trits*, each of which is 0, $^+1$ or $^-1$. The value of $t_k t_{k-1} \cdots t_0$ (where each t_i is a trit is defined to be)

$$\sum_{0 \le i \le k} t_i 2^i$$

The process of incrementing a ternary number is analogous to that operation on binary numbers. One is added to the low order trit. If the result is 2, then it is changed to 0, and a carry is propagated to the next trit. This process is repeated until no carry results. Decrementing a number is similar. One is subtracted from the low order trit. If it becomes $^-2$ then it is replaced by 0, and a borrow is propagated.

a) The cost of an increment or a decrement is the number of trits that change in the process. Starting from 0, a sequence of n operations (intermixed increments and decrements) are done. Find a constant c (the smallest you can) such that the cost of the sequence is at most cn. Prove your result by the use of an appropriate potential function.

b) Now suppose that the cost of changing a trit t_i is $i + 1$. That is, the higher order trits are more expensive to change. Answer the question in part a for this situation.

4 A selection algorithm

Here's an algorithm for finding the element of rank k in a file of n numbers. (The smallest number has rank 1, the second smallest has rank 2, etc. The term "file" in this problem refers to a multi-set, or a set that can contain duplicates.)

select(k, S)

1. If $n < 50$ sort the file using insertion sort. Return the kth smallest. (Note that insertion sort does at most $n(n + 1)/2$ comparisions.)
2. Divide S arbitrarily into $t = \lceil n/7 \rceil$ subfiles of size 7 (padding with infinities to make all the files be of size 7).
3. Sort each group of 7 using insertion sort.
4. Form a new file T consisting of the middle element of each of the files of 7. Compute $m = \text{select}(\lceil t/2 \rceil, T)$
5. Compare each element of S to m (this is n comparisons). Using these comparisions, partition S into three new files S_0, S_1 and S_2, which are the elements of S that are less than m, equal to m, and greater than m respectively.
6. If $k <= |S_0|$ then compute select(k, S_0) and return the answer. Else, if $k <= |S_0| + |S_1|$ return m. Otherwise compute select$(k - |S_0| - |S_1|, S_2)$ and return the answer.

Let $T(n)$ be the worse-case number of comparisions done by this algorithm on a file of size n. (That is, for fixed n, run the algorithm on all files of size n, and run it for all values of k from 1 to n. Take the maximum number of comparisions done in all these runs. This is $T(n)$.)

Prove that $T(n) = O(n)$. Note: Your proof should be rigorous. One way to do this is to find a linear polynomial $an + b$, and prove by induction that $T(n) \leq an + b$ for all $n \geq 7$.