15-451: Algorithm Design and Analysis

Homework 2 (due Thursday, February 02)

Directions: *Write up carefully argued solutions to the following problems. The first task is to be complete and correct. The more subtle task is to keep it simple and succinct. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. You may use any results proven in lecture without proof. Anything else must be argued rigorously. Unless otherwise specified, all answers are expected to be given in closed form.*

0. Finding Well Spaced Triples of Ones (20 points)

In this problem we are given a string of zeros and ones of length n, $S = a_0 \cdots a_{n-1}$.We are asked to find if there exists three ones in S, say, $a_i = 1$, $a_j = 1$, and $a_k = 1$so that $k - j = j - i \geq 1$. The string 10100101 does not have such a pattern while thestring 010100110011 does.

(a)Give a simple $O(n^2)$ algorithm to find at least one such triple of ones in S.
The goal in the next few parts is to use FFT to find an algorithm that runs in $O(n \log n)$ timeto find a well spaced triple of ones. In class we gave two way to view FFT's. The first was as a way to compute theproduct of two polynomials and the second was as a convolution.
(b)Suppose we convolve S with itself, $P = CONVOLVE(S, S)$. Give an interpretationof P_t both in the case when t is odd and even.
(c)Use your interpretation to count the number of well spaced triples in $O(n \log n)$ time.
(d)Give an algorithm to find a well spaced triple in $O(n)$ time given the convolution P if one exists.

Open Question:
Here is a question we do not know the answer:
Give an algorithm to find ALL well spaced triples in $O(n + T)$ time given the convolution$P$ where T is the number of well spaced triples in S.
Note: By Part (c) one gets $O(n \cdot T)$ time.

1. Matrix Transposition In-Place (10 points)

There are a number of circumstances in which it is necessary to reordera matrix in memory to its transposed form. Depending on how we representthe matrix, the cost to transpose changes.Consider two possible representations:
1) (**row-major order)** Here we store the matrix as an arrayconsisting of the concatenation of the rows.
2) (q**uadtree order)** If we assume that n is a power of twothen we can represent the matrix as a four way tree, with each child as one of thefour $n/2$ by $n/2$ sub-matrices.
The time to lookup an element in row-major order will be constant,while in quadtree it will be $O(\log n)$.
Let's consider matrix transpose which is useful for 2D FFT's.With a matrix stored in row-major order, the rows of the matrix are contiguous inmemory and the columns are discontiguous. If repeated operations are neededto be performed on the columns, for example in a fast Fouriertransform algorithm, transposing the matrix in memory will improve analgorithm's performance.
The time to perform the transpose for row-major ordercan easily be done in $O(n^2)$ using "pointer arithmetic".While in quadtree form the naive algorithm is $O(n^2 \log n)$since we lookup all the elements and write them into a new quadtree.
(a)Given a square matrix of $n \times n$ size. Derive an eagerdivide-and-conquer $O(n^2)$ time algorithm to

compute matrixtransposition for a matrix given in quadtree format.
(b)Improve the above algorithm to run in $O(1)$ time.