Problem Set 6

1. The Hadamard matrix H_n is a $2^n \times 2^n$ matrix which is defined by the recurrence $H_0 = [1]$ and

$$H_{n+1} = \begin{bmatrix} H_n & H_n \\ H_n & \mathbf{0}_n \end{bmatrix}$$

where $\mathbf{0}_n$ is a $2^n \times 2^n$ all zeros matrix.

   (a) What is the usual runtime of multiplying a vector by an $N \times N$ matrix?

   (b) Write down H_3.

   (c) Let $N = 2^n$. Give an $O(N \log N)$ algorithm to compute $H_n v$ where v is a given vector of length N. (This is sometimes used to produce fast sketches for numerical linear algebra.)

Solution. We use a divide-and-conquer approach. Let v_1 and v_2 be the the first and last $N/2$ coordinates of v respectively. Then observe that

$$H_n v = \begin{bmatrix} H_{n-1}v_1 + H_{n-1}v_2 \\ H_{n-1}v_1 \end{bmatrix}.$$

Our algorithm is to recursively use our multiplication algorithm to compute $H_{n-1}v_1$ and $H_{n-1}v_2$ and then to use these values to compute $H_n v$ as above. (For the base case $n = 0$ we simply compute the product directly.)

Our algorithm recursively calls itself twice on problems of half the size. The "conquer" step requires adding two vectors of size $n/2$ which takes $O(n)$ time. So we have the recurrence $T(n) = 2T(n/2) + O(n)$ for the runtime T. As with Mergesort, this solves to give $T(n) = O(n \log n)$.

2. You're given a collection of lines $y = m_i x + b_i$ in the plane such that no three lines meet in a point. Say that line i is "visible from above"

if there there is a vertical line for which line i intersects it at greater y-coordinate than any other line.

Give an $O(n \log n)$ algorithm to find all the lines that are visible from above.

3. (a) Suppose that you have a black-box algorithm which for arrays of size at least 4 can produce can produce an element between the first and third quartile in $O(1)$ time. Using this black box, show how to find the median of an array in linear time. Assume the list has odd length for simplicity.

   (b) How could you approximate this black-box in practice?

   Solution. We'll implement a more general function `findKthSmallest(k, A)` that finds the kth smallest element in the array A. If A has fewer than 4 entries then we can return the answer in $O(1)$ time. Otherwise our algorithm works as follows. We apply the black box to obtain an element x in the middle half of A. Then make a linear scan through A, splitting A into two arrays A_S and A_L consisting of the elements of A that are at most x and larger than x respectively. If $\text{len}(A_S) \geq k$ then the kth smallest element is in the smaller half and so we return `findKthSmallest(k,`A_S`)`. Otherwise it's in the larger half and so we return `findKthSmallest(k - len(`A_S`),`A_L`)`.

   Now let $T(n)$ be the worst case runtime of `findKthSmallest(k, A)` applied to an array A of length at most n. Both of A_S and A_L have size at most $(3/4)n$ by the guarantee of the black box. Also the linear scan through A took $O(n)$ time. So we have the recurrence $T(n) \leq T((3/4)n) + Cn$, where C is a constant. Expanding this out, we get

$$T(n) \leq Cn + C(3/4)n + C(3/4)^2 n + C(3/4)^3 n \ldots + O(1).$$

   This is a geometric series so $T(n)$ is $O(n)$ as desired.

4. You're given an $n \times n$ array containing a number x_{ij} in row i column j. We say that (i, j) is a local minimum of the array if x_{ij} is smaller than or equal to each of its horizontally and vertically adjacent neighbors. Show how to find a local minimum in $O(n)$ time. (Note that this is faster than linear, since there are n^2 entries in the array.)