Design and Analysis of Algorithms	**out:** February 26, 2015
Problem Set 4	
Lecturer: Daniel A. Spielman	**due: 2:30** PM, March 5, 2015

Instructions. Read Carefully

1. Make sure that your yale **netid** appears on **every** page of your problem set, and that it is written clearly.
2. The problem sets are due at the beginning of class. Solution sets will be provided in class. For this reason, late problem sets are not accepted.
3. You will **hand in each problem separately**, so make sure that they are not stapled together and that no problem appears on the back of another. There will be 4 boxes in the front of the class, one for each problem. Put each problem in the correct box.
4. Alternatively, you may submit the problem set electronically via the Classes server. If you do so, submit it as a **pdf** and make sure that your **netid** appears on **every** page. **Each problem must be in its own document.**
5. Cite any resources that you use, other than the textbook, TA and instructor.
6. You **must not** search the web for solutions to similar problems given out in other classes.

Collaboration Policy

You must write the solutions to the problems independently and in your own words.

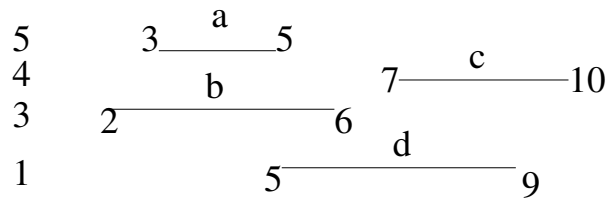
However, you are allowed to discuss the problem sets in small groups of no more than 4 students. To ensure that you understand the solutions you submit, you are forbidden from taking written notes during your discussions. You must list at the top of the problem set everyone (other than course staff) with whom you have discussed the problem set. Failure to list people with whom you have discussed a problem set is considered a violation of academic honesty.

Problem 1: Visible segments

In this problem, you are given as input a collection of horizontal line segments, and are asked to determine which segments are visible from above, and where. In particular, each segment has a height, h_i, and first and last x-coordinate, f_i and l_i. You may assume that all of the heights and x-coordinates are distinct.

For example, the following figure contains 4 segments, with data

name	h	f	l
a	5	3	5
b	3	2	6
c	4	7	10
d	1	5	9



You are asked to divide the x-coordinate into intervals and report which segment is highest in each interval. For example, the answer for this problem is

$$(2,3) : b \quad (3,5) : a \quad (5,6) : b \quad (6,7) : d \quad (7,10) : c$$

Present a Divide-and-Conquer based algorithm that solves this problem in time $O(n \log n)$. Prove that your algorithm is correct and explain why it runs in the stated time.

There are other ways of solving this problem, but you will only get full credit for a Divide-and-Conquer solution.

Hint: I know two different ways of solving this by a divide-and-conquer algorithm. One is basically like Mergesort: divide the intervals in half arbitrarily, solve the problem on each half, and then merge the solutions. In the other solution, one is more intelligent about how one divides the segments: divide them into those with the first and last $n/2$ start times, solve in each half, and then merge those solutions together.

Problem 2: Improving `GoodRandomSplit`

I (roughly) recall the algorithm `GoodRandomSplit` from the lecture on randomized divide and conquer.

$(S^-, x, S^+) =$ `GoodRandomSplit`$(a_1, \ldots, a_n)$.

Repeat.

1. Pick a $j \in \{1, \ldots, n\}$ uniformly at random.
2. Set S^- to contain all $a_i < a_j$.
3. Set S^+ to contain all $a_i > a_j$.
4. Set $x = a_j$.

Until $n/4 \leq |S^-| \leq 3n/4$

In our analysis of `GoodRandomSplit`, we used (the trivial fact) that a randomly chosen number from $a_1, \ldots, a_n$ has rank between $\lfloor n/4 \rfloor$ and $\lceil 3n/4 \rceil$ with probability at least $1/2$. We call such an element a *good splitter*. That is, x is a *good splitter* if

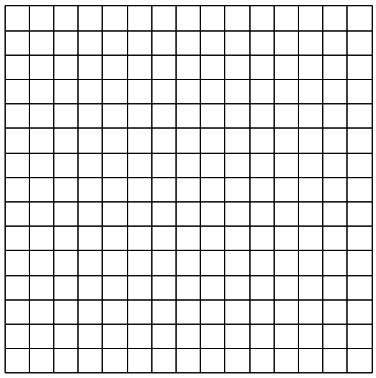
$$|\{i : a_i < x\}| \leq 3n/4 \quad \text{and} \quad |\{i : a_i > x\}| \leq 3n/4.$$

We could improve our chance of getting a good splitter if we chose more than one random element, and took the median of those we chose. For simplicity, assume that n is a multiple of 4 and that $a_1, \ldots, a_n$ are distinct.

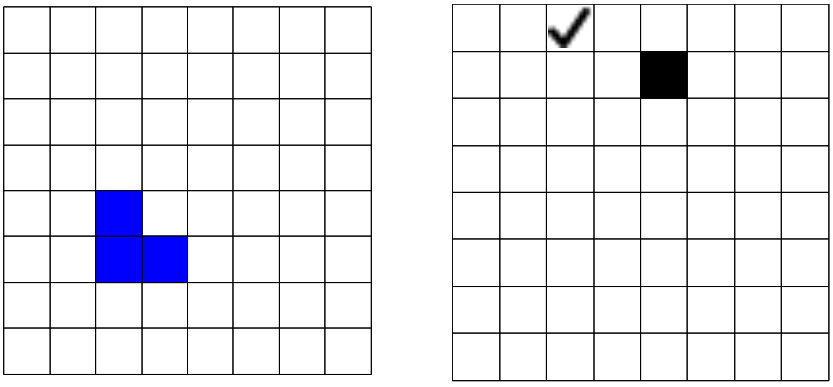
Prove that if we choose three elements at random and use the median of those three, then the probability we obtain a good splitter is at least $11/16$. This might be easiest if you choose the three elements independently, rather than requiring that they be distinct.

Problem 3: Covering the Blokus board

Blokus is a popular new board game. It is played on a grid-shaped board



Each player places pieces on the board that cover a few squares, like the "v"-shaped piece on the left below.



While the actual game has pieces of many shapes and sizes, we will only consider two types of pieces: a piece that covers exactly one square, which we call a *unit piece*, shown on the right above, and the piece shown on the left above which I will call a *v-piece*.

I claim that if the Blokus board is square and that the number of squares down each side is a power of 2, then it is possible to cover it with one unit piece and many non-overlapping v-pieces. Moreover, I claim that you can do this no matter where on the board the unit piece is located.

For example, if the board is 2-by-2, then it can be covered by one v-piece and one unit piece.

a. You will design an algorithm that shows how to do this. Your algorithm should take as input a number n giving the size of the board and coordinates x, y each between 1 and n giving the location of the unit piece. It should return a description of the locations and orientations of v-pieces that cover the rest of the board. Your algorithm should run in time polynomial in n, and you should explain why it does if it is not obvious.

   Please try to find a concise way of doing this. (6 Points)

b. Explain why your algorithm is correct. (4 Points)

Remember: Your algorithm only has to work when n is a power of 2. Otherwise a solution might not exist.

Problem 4: Finding the largest fraction

In this problem, you are given 4 lists of n **postive** numbers each.

$$a_1, \ldots, a_n$$
$$b_1, \ldots, b_n$$
$$c_1, \ldots, c_n$$
$$d_1, \ldots, d_n$$

You should think of a_i/b_i as representing a fraction, and c_j/d_j as another.

Your problem is to find the i and j for which

$$F(i,j) \stackrel{\text{def}}{=} \frac{a_i + c_j}{b_i + d_j} \tag{1}$$

is as large as possible. First, let me warn you that your instincts for this problem are probably wrong. My first instinct is to find the i for which a_i/b_i is as large as possible, and then the j for which c_j/d_j is as large as possible, and to use them. It turns out that there are inputs for which neither of these is part of the solution. For example, consider

$$a_1 = 3, b_1 = 1, (\text{so } 3/1 = 3), \quad a_2 = 11, b_2 = 4, (\text{so } 11/4 = 2.75),$$
$$c_1 = 6, d_1 = 5, (\text{so } 6/5 = 1.2), \quad c_2 = 1, d_2 = 1, (\text{so } 1/1 = 1).$$

The largest fraction of form (1) is

$$F(2,2) = \frac{a_2 + c_2}{b_2 + d_2} = \frac{11+1}{4+1} = 2.4.$$

Math is crazy like that.

The naive algorithm for solving this problem is to compute all $F(i,j)$, which takes time $O(n^2)$. You should state an algorithm that finds the i and j for which $F(i,j)$ is largest in expected time $O(n \log n)$, explain why it is correct, and explain why its expected running time is as claimed. In fact, one can get the expected time down to $O(n)$, but you don't need to do that. I will give you hints.

Hints:

1. Given a fixed i, one can find the j maximizing $F(i,j)$ in time $O(n)$. I suggest that you write a routine that does this. Call this index $f(i)$. That is $F(i, f(i))$ maximizes $F(i,j)$ over all j.

2. Given a number r, let S_r be the set of i for which there exists a j such that $F(i,j) > r$. Formally,

$$S_r = \left\{ i : \max_j F(i,j) > r \right\}.$$

It turns out that, given a number r, one can also compute S_r in linear time. To see this, note that

$$\frac{a_i + c_j}{b_i + d_j} > r \quad \iff \quad a_i + c_j > r b_i + r d_j \quad \iff \quad a_i - r b_i > r d_j - c_j.$$

So, one can compute a threshold $t_r = \min_j r d_j - c_j$, and $i \in S_r$ if and only if $a_i - r b_i > t_r$.

3. The last idea for the algorithm is that you should compute S_r for well-chosen values r. In particular, you should use values like $F(i, f(i))$ for well-chosen i. Choosing i at random from a set like S_r should work.

I begin by setting $S = \{1, \ldots, n\}$. I then repeat the process of choosing an i at random from S, and then updating $S = S_{F(i,f(i))}$.