Problem 1

I consider the problem of finding lines crossing a rectangle. Given n vertical and horizontal segments in the plane, I build a size $O(n \log n)$ data struture allowing me to output all segments that intersect a query rectangle in $O(k+\log^2 n)$ using an interval tree data structure that takes n intervals on the real line, uses $O(n)$ space and outputs the set of all intervals intersecting with a given query interval in $O(k + \log n)$ time.

I treat horizontal segments and vertical segments separately. I will describe how to find intersecting vertical segments (the case for horizontal segments is symmetric).

I consider the x-coordinate of each vertical line and build a binary search tree sorted by the x-coordinate, x-BST. For each substree in the x-BST, I create an interval tree data structure with the vertical lines belonging to this subtree. Now, every line belongs to $O(\log n)$ subtrees, so space is $O(n \log n)$.

For search, I find the subtrees of the x-BST whose union is the x-interval. I union the result from the related substructure for the y-interval query. There are $O(\log n)$ subtrees to search, and $O(\log n)$ work per subtree, so the total work is $O(\log^2 n)$. So the total work is $O(k + \log^2 n)$ if we need to output the k intersecting segments.

Problem 2

I sort the endpoints by their angle to the player point. The events are the starts and ends of segments, sorted by the angle. I maintain the positions of the segments on the half-line sweep. If, at any event, some wall point is first on the half-line sweep, I add the wall to the visible walls. At the start, at angle 0, I have to add all the segments that are active at this angle.

The bottleneck is sorting the endpoints, so the total running time is $O(n \log n)$.

Problem 3

I show that the randomized incremental construction algorithm for convex hull presented in class runs in $O(n \log n)$ with high probability, i.e. a probability $\geq 1 - O(1/n^c)$ for any desired $c > 0$.

Let $X_i(p)$ be an indicator random variable denoting whether p's pointer is changed in the ith step. We know that $\sum_i X_i(p) = c' \log n$. Using the Chernoff bound, $\mathbb{P}\left[\sum_i X_i(p) \geq (1+\epsilon)c' \log n\right] \leq e^{-\epsilon^2 c' \log n/4}$ or $\leq 2^{-(1+\epsilon)c' \log n}$ depending on whether ϵ is $<$ or $\geq 2e - 1$. In both cases, the probability comes out to $O(1/n^c)$ for c, a suitable function of ϵ.

Now, for a point p, the probability that $\sum X_i(p) = O(\log n)$ is $1 - O(1/n^c)$.

Using the union-bound, the probability that for any point p, $\sum X_i(p) > O(\log n)$ is $\leq nO(1/n^c) = O(1/n^{c-1})$. So the probability that, for all points p, $\sum X_i(p) = O(\log n)$, is $\geq 1 - O(1/n^c)$.

Problem 4

Part (a)

When the sweep line encounters a new point p, if p is one point in the closest pair behind the sweep line, the other point in the cloest pair must be at distance $\leq d$ of p. So I can simply consider the portion of the strip consisting of a rectangle of size $2d$ by d, where p is at the center of one of the $2d$ side.

Part (b)

The portion of the strip can contain only a constant number of points. Indeed, each point needs to be at least d away from any other point. So each point in the portion can claim a territory of size $\geq \pi(d/2)^2/4 = \pi d^2/16$. Since the portion has size $2d^2$, at most $32/\pi$ points can be in the portion. This is a very rough upper bound, and it's a constant.

Part (c)

In order to sweep, I keep an array of the points sorted by the x-coordinate. I keep a pointer in the array to the point at the beginning of the strip. I keep the points in the strip in a BST sorted by the y-coordinate. Then, when I get a new point p, the candidates for closest pair can be identified by considering the interval from $p.y - d$ to $p.y + d$ in the BST. I update the strip by changing the pointer of the beginning of the strip in the array, and in the BST deleting all points in between the old array pointer and the new array pointer. Since I am deleting each point once at most, the deletion will cost me $O(n \log n)$ for the n events. Querying the BST also cost me $O(\log n)$ per query, and the initial sorting of the array is $O(n \log n)$. Therefore, the total time bound is $O(n \log n)$.

Part (d)

The algorithm generalizes to higher dimensions, because part (b) holds in higher dimensions using a similar argument. Instead of keeping the points in a BST, I keep it in an orthogonal range query structure. Assuming this structure allows me to do query and update in $O(\log^{d-1} n)$, the total time bound for the algorithm is $O(n \log^{d-1} n)$.

Part (e)

We've seen in class that a Voronoi diagram has a linear number of edges. It is clear that the closest pair will be separated by an edge in the Voronoi diagram. Indeed, in the limiting case, we have 3 equidistant points and they would still be separated by Voronoi edges. Therefore, it suffices to check, among all edges, for the one which is closest to its points on either side. Since there are $O(n)$ edges, the closest pair can be identified in $O(n)$ time. This gives an alternative

$O(n \log n)$ time algorithm in 2 dimensions, as it takes $O(n \log n)$ to construct the Voronoi diagram.

Problem 5

When I add a half-plane, if the half-plane doesn't conflict with all or 0 vertices, I must add it in between vertices that conflict and vertices that don't conflict. I will be adding two vertices to take this new half-plane into account. I must now update the conflict graph for these 2 new vertices. Each of these 2 new vertices are between an old vertex A (now removed) and an old vertex B (still part of the intersection polytope). To decide whether a half-plane conflicts with a new vertex, I can first simply look at its A and B vertices. If the half-plane conflicts with both A and B or with neither, then I know that it conflicts in the same way with the new vertex. The only cases I need to check explicitly is when the half-plane conflicts with A but not B or vice versa.

I use a backward analysis similar to the analysis of the randomized incremental algorithm for convex hull to conclude that this algorithm runs in $O(n \log n)$ expected time. When I remove a half-plane at a stage where the polytope has k vertices, the only case where I have to do work is if the half-plane goes through two particular sides of the polytope. Given that the half-plane cuts the polytope, the probability that it goes through two specific sides is $2/k$. In this case, the work is $O(n)$. So I get an expected running time of $O(n) \sum_k 2/k = O(n) 2H_n = O(n \log n)$.

Problem 6

For my final project, I want to investigate the problem of routing in the control layer of microfluidic chips. Concretely, the control layer is composed of N valves and N punches, and each valve needs to be connected to a punch. The routing needs to respect minimum distances between connection lines, between a valve and another connection line, and between a punch and another line. In addition, the routing must avoid some obstacles and minimize crossing of flow lines. In the final routing, we would like to minimize the total length of the connection lines, but also the number of corners.

If we don't minimize the number of corners, the routing problem can be represented as a min-cost max-flow problem on a manhattan grid with the resolution equal to the minimum distance between connection lines. Each edge and node has a capacity of 1, each edge has a cost of 1 (to minimize the total length of connection lines), and flow nodes have a certain cost (to minimize flow crossings). We connect the source to each valve and each punch to the sink, and find a min-cost max-flow between the source and the sink.

In order to minimize the number of corners, we can duplicate each node into a vertical node and a horizontal node, and have more expensive crossing edges. However, we now need to add constraints so as to avoid the intersection of a horizontal edge and a vertical edge. These constraints turn the problem into a linear program. I want to investigate the integrality gap in this linear program and explore ways to reach an integer solution.

There are other extensions to consider. For example, we might want a grid resolution finer than the minimum distance between connection lines, because the chip elements (valves, punches and obstacles) might not fit well into the coarser grid. This adds more constraints to the linear program, and might render it more difficult.