Computational Geometry **Module 7-8** **NAME:**
PROBLEM SET 3

Scoring Rubric:

Points Earned on Problem	**Criteria**
4	Excellent. All work is correct. Neatly written and legible. Clearly presented and worked through without a fault.
3	Good. Only slight error in work. Neatly written and legible. Clearly presented and clear understanding of the concepts. Only possesses mistakes in calculations.
2	Fair. Errors in work. Illegible and not written neatly. Work is not presented in a way that shows understanding of the concepts. Calculation is not clear or correct.
1	Poor. Work is filled with errors, work is not clear and concepts are confused. Needs work to bring their work up to this courses expectations.
0	Immediate action needs to be taken to correct your understanding and/or effort in this class.

Directions: Each problem from the Problem Set will be worked out in its entirety. Your solution method may vary and will not be used against you if you get the right answer. You are expected to honor the following rules:

- Do not read any homework solutions or "sample solutions", including those produced by students (except yourself), instructors, or TAs, related to this or any other course on Computational Geometry or a closely related topic. Some of the homework questions I'll use this semester have been used before by me or other teachers at other institutions, and you're on your honor not to consult the solutions.
- Do not copy solutions or work from classmates or any work you've seen online. The purpose of the problems is to grow your understanding of the subject manner and refine your skills at writing and implementing algorithms. • Do not give solutions to each other. Learning is not done by copying solutions.

You may consult any books, papers, web pages, or other inanimate information sources you please, so long as they're not other people's homework solutions or sample solutions. If you use information from such a source, please cite it. I don't believe that the solutions to many of the problems I'll give are available in easy-to-find references, but if you do find one, it's fair game as long as you cite it. After all, a good literature search should be rewarded. You may not, however, let your classmates know about your discovery or sources until the deadline has passed.

You are welcome to ask me to clarify anything in the readings, the lectures, or this homework.

Computational Geometry **Module 7-8** **NAME:**
PROBLEM SET 3

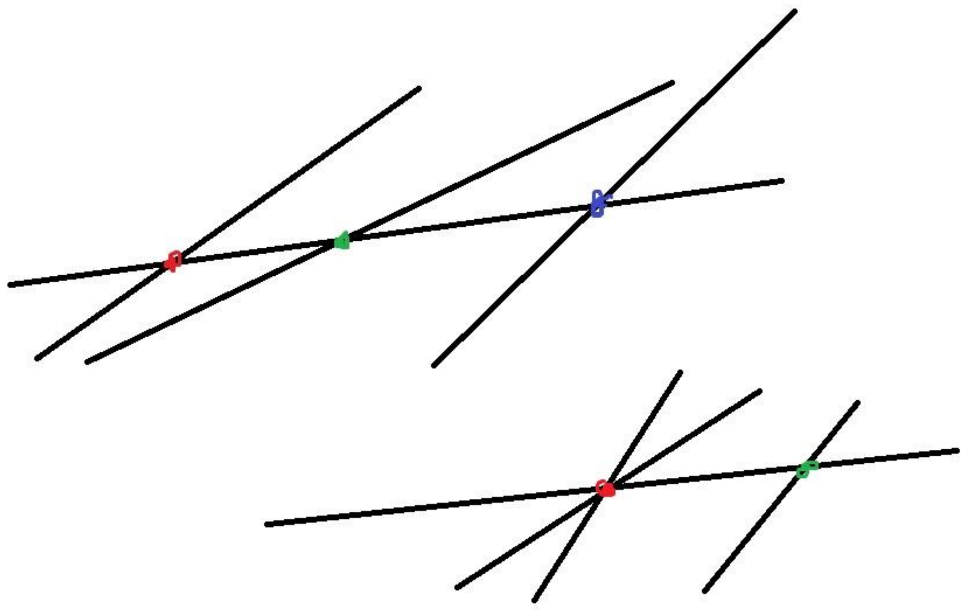
I prefer typeset solutions, on paper. I will accept handwritten solutions with the proviso that I reserve the right to give a zero to a solution if there is any word or variable I can't read clearly. (Sorry, I just don't have time to infer from context whether that vague squiggle is an i or a j.) Any such decision is final and irrevocable. By contrast, typeset solutions reserve the right to argue with my grading after the fact.

On the other hand, handwritten figures will be gratefully accepted. Anything to encourage you to draw figures! A simple figure can often mean the difference between my spending five minutes or two hours grading an answer. Thank you in advance for every figure you draw.

1. (Levels in Arrangements) Is it possible for an arrangement of lines, no two parallel, to have a vertex of level 2 but no vertex of level 1? Explain.

My immediate reaction based on considering levels of a tree or graph is no. I think this could apply if we consider lines to be edges in the context of a tree. In a tree, it is trivial to see that it is impossible to have a level 2 vertex without a level 1 vertex.

However, I think I can draw a trivial example that shows a level 0 vertex and level 2 vertex in an arrangement:
In the lower arrangement, the green vertex should have a level of 2 and the red arrangement should have a level of zero. The green vertex has two lines that are strictly above it, the red has zero. Is it possible for an arrangement of lines, no two parallel, to have a vertex of level 2 but no vertex of level 1?



2. (Stabbing vertical segments) Let *S* be a set of *n* vertical line segments in the plane. Describe an algorithm that determines whether there exists a line that intersects every segment in *S*, and identifies such a line if one exists. For full points, your algorithm should run in linear time.

Describing in words, what I think I want to do is a horizontal sweep across the plane.

1. I probably want to sort the segments from top to bottom in order of their top most point. I expect this to be linear. Additionally, I want each segment to have a member that holds the count of intersections.
2. Sweep from top to bottom with a structure that will manage the order of segments it currently intersects.
    a. Early exits:
        i. Once a segment is encountered if we ever have zero segments intersected during the sweep before the end we don't need to continue and know there is not a segment that intersects all segments
    b. Record the times lines intersect.
    c. I'm not sure if this sweep can be implemented in a way that is linear in time.(I think this part will fail to be linear in time)
        i. We will encounter 2n points
        ii. Every time we encounter a start point we need to put it in the right order
        iii. Every time we encounter an end point we need to figure out the overall change in order and increment the respective counts

Once the last segment is encountered, if any of the segments has the value n-1 for the intersection count we know that segment intersects all of the other line segments.(Search through n segments in linear time)

3. (Packing disks) Let *D* be a set of disks of radius *r* in the plane, which represent atoms. (A disk is a set of points consisting of the points on a circle and all the points inside it.) The disks in *D* may partly overlap each other (due to molecular bonds). Let *R* be an axis-aligned rectangle.
We wish to determine whether it is possible to place another disk d of radius *s* (representing a catalyst) such that *d* lies inside *R* and does not overlap any disk in *D*. (It is okay if the boundary of d intersects disks in *D*, or the boundary of *R*, but the interior of d may not.)
Give an $O(n\log n)$-time algorithm that answers this question, where *n* is the number of disks in *D*. (Hint: what geometric structure makes this computation straightforward?)

My thought is to use a half edge or quad edge data structure, not sure about which but I'm picking half edge.

Based on the prompt it seems we can assume that D fits in the plane but we are not guaranteed to have any data structure built up around D, so we'd have to build that.

I feel I need to gloss over a lot of important implementation details just because there would be a lot to cover and I'm not sure how to do everything. To summarize in case I leave anything out, I want to construct a half edge data structure that will be a Voronoi diagram. Centers of the circles are faces, from the faces we can use the radius to determine the maximum circle size given a vertex as the center. During construction we calculate this by iterating over the faces of a vertex. We use the rectangle as the outer bound which I'm not entirely sure how to do. (Hence glossing over)

Based on what I understand as the time complexity to generate Voronoi/Delaunay this should be O(nlogn) and during construction we can keep track of the size of potential circles that could be inserted. Keeping track of the potential circles should not change the time complexity and should let us check whether we can insert a circle of a given size.