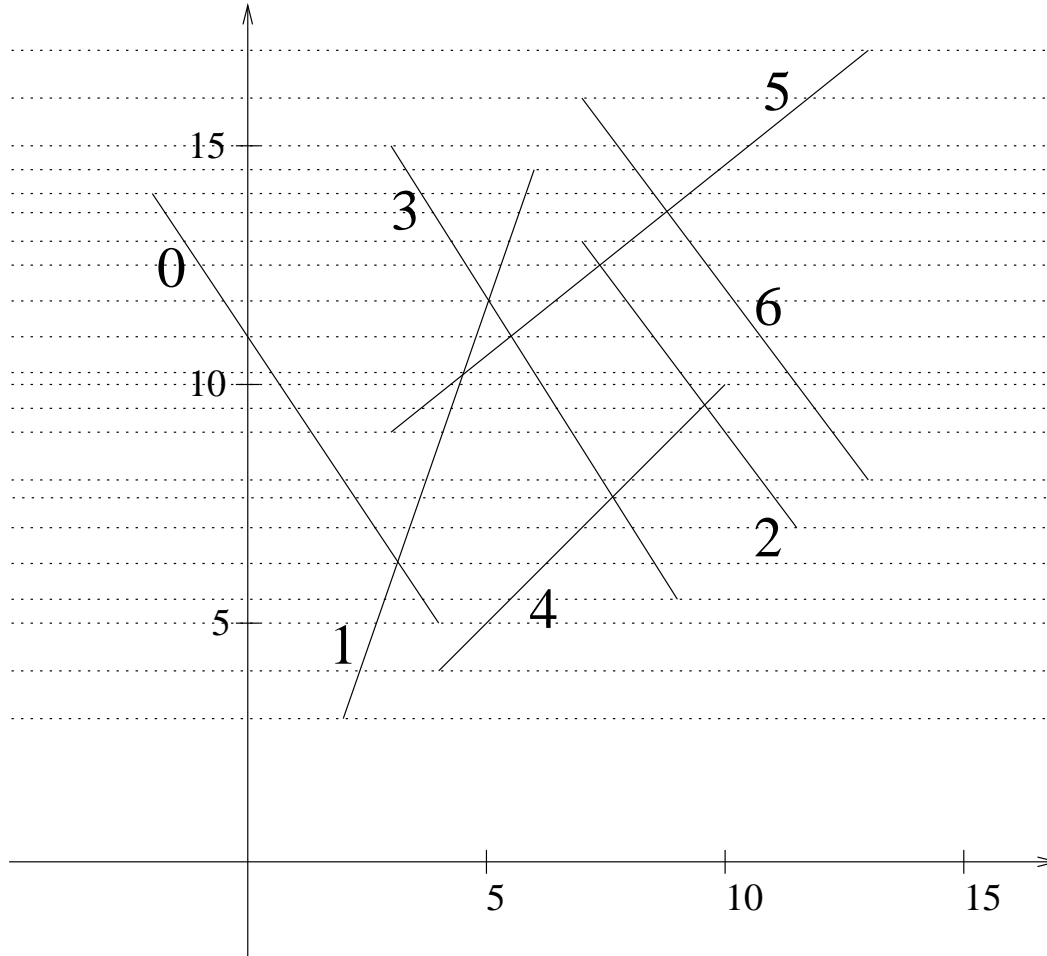


## COMPUTATIONAL GEOMETRY Homework Set # 8 – Solution Notes

(1). Consider the set  $S$  of 7 line segments given by  $S = \{s_0, s_1, \dots, s_6\} = \{((-2,14),(4,5)), ((6,14.5),(2,3)), ((7,13),(11.5,7)), ((3,15),(9,5.5)), ((10,10),(4,4)), ((13,17),(3,9)), ((7,16),(13,8))\}$ , where each segment  $s_i = (a_i, b_i)$  has upper endpoint  $a_i$  and lower endpoint  $b_i$ . I draw horizontal lines through the endpoints and the intersection points in the figure, to help you order the events.



Apply the Bentley-Ottmann sweepline algorithm to  $S$ . Give the event queue  $Q$  and the sweep status  $\mathcal{L}$  just after each event. (Use the notation as in the text that  $x_{ij}$  denotes the point (if any) at the intersection of segment  $s_i$  and segment  $s_j$ .)

Show the chart, just as in the handout given in class. (You need not compute the actual intersection points  $x_{ij}$ .)

I show the execution of the algorithm in the table below. I place an event point  $x_{ij}$  in square brackets (“ $[x_{ij}]$ ”) if it would not be present if we follow the modified Bentley-Ottmann algorithm, which deletes crossing events from the queue whenever the corresponding segments stop being adjacent in the sweep status (they are reinserted later, when the segments are again adjacent).

Event	Event Queue, Q	Sweep Status, $\mathcal{L}$
-	$(a_5 a_6 a_3 a_1 a_0 a_2 a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$()$
$a_5$	$(a_6 a_3 a_1 a_0 a_2 a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_5)$
$a_6$	$(a_3 a_1 a_0 x_{56} a_2 a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_6 s_5)$
$a_3$	$(a_1 a_0 x_{56} a_2 a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_3 s_6 s_5)$
$a_1$	$(a_0 x_{56} a_2 x_{13} a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_3 s_1 s_6 s_5)$
$a_0$	$(x_{56} a_2 x_{13} a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_0 s_3 s_1 s_6 s_5)$
$x_{56}$	$(a_2 x_{13} x_{15} a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_0 s_3 s_1 s_5 s_6)$
$a_2$	$(x_{25} x_{13} [x_{15}] a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_0 s_3 s_1 s_2 s_5 s_6)$
$x_{25}$	$(x_{13} x_{15} a_4 b_5 b_6 b_2 b_3 b_0 b_4 b_1)$	$(s_0 s_3 s_1 s_5 s_2 s_6)$
$x_{13}$	$(x_{35} [x_{15}] a_4 b_5 b_6 b_2 x_{01} b_3 b_0 b_4 b_1)$	$(s_0 s_1 s_3 s_5 s_2 s_6)$
$x_{35}$	$(x_{15} a_4 b_5 b_6 b_2 x_{01} b_3 b_0 b_4 b_1)$	$(s_0 s_1 s_5 s_3 s_2 s_6)$
$x_{15}$	$(a_4 b_5 b_6 b_2 [x_{01}] b_3 b_0 b_4 b_1)$	$(s_0 s_5 s_1 s_3 s_2 s_6)$
$a_4$	$(x_{24} b_5 b_6 b_2 [x_{01}] b_3 b_0 b_4 b_1)$	$(s_0 s_5 s_1 s_3 s_2 s_4 s_6)$
$x_{24}$	$(b_5 b_6 x_{34} b_2 [x_{01}] b_3 b_0 b_4 b_1)$	$(s_0 s_5 s_1 s_3 s_4 s_2 s_6)$
$b_5$	$(b_6 x_{34} b_2 x_{01} b_3 b_0 b_4 b_1)$	$(s_0 s_1 s_3 s_4 s_2 s_6)$
$b_6$	$(x_{34} b_2 x_{01} b_3 b_0 b_4 b_1)$	$(s_0 s_1 s_3 s_4 s_2)$
$x_{34}$	$(b_2 x_{01} b_3 b_0 b_4 b_1)$	$(s_0 s_1 s_4 s_3 s_2)$
$b_2$	$(x_{01} b_3 b_0 b_4 b_1)$	$(s_0 s_1 s_4 s_3)$
$x_{01}$	$(b_3 b_0 b_4 b_1)$	$(s_1 s_0 s_4 s_3)$
$b_3$	$(b_0 b_4 b_1)$	$(s_1 s_0 s_4)$
$b_0$	$(b_4 b_1)$	$(s_1 s_4)$
$b_4$	$(b_1)$	$(s_1)$
$b_1$	$()$	$()$

(2). *O'Rourke, problem 4, section 7.11.5, page 293.* Let the  $n$  intervals be  $I_1, \dots, I_n$ , where  $I_i = (l_i, r_i)$ . We spend  $O(n \log n)$  time sorting the  $2n$  numbers that are the left and right endpoints. Then, in  $O(n)$  time we advance through the sorted list, from left to right. We keep track of a counter,  $K$ , which says how many intervals are present at the particular  $x$  coordinate. Initially,  $K = 0$ . Each time we encounter a left endpoint ( $l_i$ ), we increment  $K$  ( $K \leftarrow K + 1$ ); each time we encounter a right endpoint ( $r_i$ ), we decrement  $K$  ( $K \leftarrow K - 1$ ). Each time that  $K$  goes from being 0 to being positive, we have encountered a left endpoint of one of the intervals of the union,  $U = \cup_i I_i$ . Each time that  $K$  goes from a positive number to 0, we have encountered a right endpoint of one of the intervals of  $U$ . This allows us to keep track of the union  $U$  as we walk through the sorted list. Once we have the intervals that make up the union,  $U$ , we just sum up their lengths (or do it as we discover them), in  $O(n)$  time. The total time is  $O(n \log n)$  (dominated by the sorting step).

Is this the best we can do? Can we avoid the  $O(n \log n)$  and possibly do it in linear ( $O(n)$ ) time? The answer is “no”. We can show a lower bound from Element Uniqueness (which is  $\Omega(n \log n)$ ). The input to EU is an unordered set of  $n$  integers,  $x_1, \dots, x_n$ . We create an instance of our interval problem by turning each point into a small interval:  $I_i = (x_i, x_i + 0.5)$ . If we could output the length of the union  $U$  in time faster than  $O(n \log n)$ , then we would be able to solve EU faster than  $O(n \log n)$ , since the total length of  $U$  is  $n/2$  if and only if the numbers  $x_i$  are all unique. Thus, we could use our union algorithm to solve EU. Thus,  $\Omega(n \log n)$  is a lower bound for computing the union too. (Note: I have assumed here that the output of our “Length-of-Union” problem is a single number (just the length). If we demand that the intervals of  $U$  be *reported in sorted* order, then we easily get an  $\Omega(n \log n)$  lower bound, from SORTING (we do not need to know about the lower bound on EU).)

(3). *O'Rourke, problem 5, section 7.11.5, page 293.* We are given a set  $S$  of  $n$  points in the plane. First, we construct the Voronoi diagram for  $S$ ; this takes time  $O(n \log n)$  and results in a data structure (e.g., winged-edge data structure) of size  $O(n)$ . An empty circle query for a point  $q$  can be answered by finding the point of  $S$  that is closest to  $q$ : if point  $p \in S$  is closest to  $q$ , then the radius of the largest empty circle centered on  $q$  is just the distance,  $d(q, p)$ .

Thus, we want to preprocess the Voronoi diagram,  $VD(S)$ , for point location queries: This takes time  $O(n)$ , since each face of  $VD(S)$  is a simple polygon (in fact, a convex polygon), implying that each can be triangulated in linear time, and then Kirkpatrick’s point location method can be applied with additional  $O(n)$  preprocessing time. A query for point  $q$  is answered in time  $O(\log n)$  and reports back which face of  $VD(S)$  contains  $q$ . This tells us the closest point,  $p \in S$ , to  $q$ , since  $p$  is the “owner” of the face of  $VD(S)$  containing  $q$ .

In total, we spend  $O(n \log n)$  preprocessing time to build a data structure of size  $O(n)$  for which empty circle queries cost  $O(\log n)$  each.

(4). *Build the Kirkpatrick point location hierarchy for the triangulation shown below. At each step, when you identify*

an independent set, apply Algorithm 7.4 on page 277, breaking ties when you select a node in favor of the lowest numbered vertex. When you retriangulate a hole, use the simple ear-clipping algorithm (Triangulate, page 39), starting at the (rightmost) bottommost vertex of the hole (as “v0” in Triangulate, the first one tested for earity), and proceeding counterclockwise.

(a). List the independent sets corresponding to each stage of the algorithm. Also, for each stage, draw the corresponding triangulation.

The independent sets at each stage are given by: {2, 4}; {3, 7}; {5}; {6}.

See the sequence of figures below.

(b). Draw the final hierarchy as a DAG, with each node of the hierarchy labeled by the triangle to which it corresponds. (When you label a node, please list the triangle as a triple with the vertex indices in order; e.g., the triangle with vertices “1”, “8” and “9” should be written as “189” (not as “819” or “918”, etc).)

See the DAG below.

(c). Highlight in the final hierarchy those nodes that are explored when point location is performed for point p, as shown in the figure.

The nodes are circled in red below.

