The design of a VLSI chip consists of a set of polygons, each assigned to a layer.

In general, these polygons are NOT disjoint. Furthermore, many edges may intersect in a single point or in a segment.

The fabrication house must form the union of the polygons on each layer. Further processing is then performed on the contour.

- The fabrication house would prefer to restrict designers to rectilinear polygons for ease of processing

- Chips gain efficiency by minimizing wasted space, so the designers would prefer the freedom to choose an unlimited number of directions

A compromise of some fixed number of orientations more than two, but less than infinity, may be the solution.

The first step in the fabrication process involves computing the contour of the union of all polygons on a given layer. Next, the polygonal region(s) with holes must be decomposed into machine writable pieces (sometimes overlapping!), and appropriate doses must be assigned to these pieces so as to maintain the correct electrical properties. For example,

- regions near the contour need to be filled with higher doses than interior regions
- regions of one component that are close to regions from another component need to be filled at lower doses.

Institutions fabricating chips have had procedures for these processes for some time. Two changes have recently occured:

the size and complexity of VLSI chips has risen dramatically

more designers have decided to leave Manhattan geometric and use more orientations

Two principles at war:

- Fabrication preprocessing is easier if all polygons are rectilinear
- Chips gain efficiency by minimizing wasted space

In this particular application, it may be reasonable to forgo the attempt to write robust and efficient code to produce accurate instructions for the fabrication of a chip designed using edges of arbitrary slope.

A few companies are interested instead in restricting designers to some fixed collection of orientations, greater than two but far smaller than infinity, chosen compatibly so that all numbers can be represented as integers.

TODAY, I will report only on work done with Iliana Bjorling-Sachs and a couple of student programmers on the contour problem.

This paper studies the problem of finding the contour of an n-edged collection of polygons satisfying:

1) all vertices have integer coordinates
2) all slopes belong to a finite, fixed set

We study the problem from 2 perspectives.

1) which rectilinear-based algorithms can be generalized to handle polygons of 2 or more additional directions while still retaining efficiency

2) which algorithms for computing the union of arbitrary polygons can be revised to take advantage of the restricted number of orientations, improving robustness, efficiency, or both?

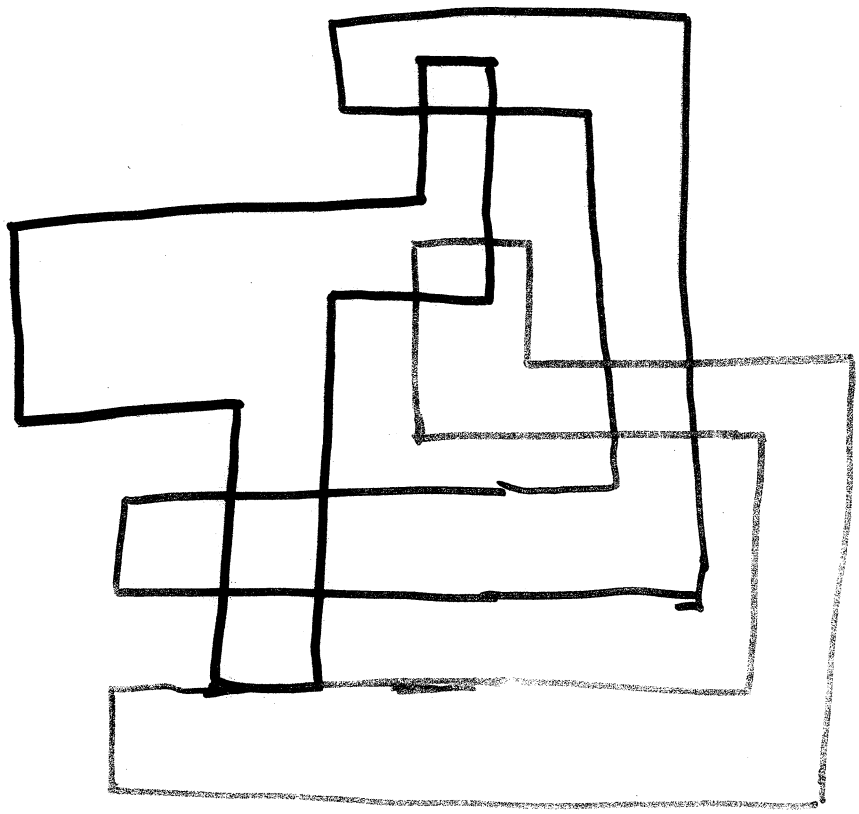
GENERAL COMMENTS

IF THE INPUT POLYGONS ARE RECTILINEAR, ALL X-COORDINATES (RESP. Y-COORDINATES) OF INTERSECTION POINTS BELONG TO THE SET OF X-COORDS (RESP Y-COORDS) DETERMINED BY THE POLYGON VERTICES!

① IF VERTICES ARE AT GRIDPOINTS, THEN SO ARE INTERSECTION POINTS
⇒ INTEGER ARITHMETIC

② X-COORDS (RESP. Y-COORDS) CAN BE PRESORTED AND NORMALIZED.
⇒ ANY FINAL SORTS ON THE POSSIBLY $\Theta(n^2)$ VERTICES OF THE CONTOUR CAN BE DONE IN TIME LINEAR IN THE SIZE OF THE CONTOUR

PLANE SWEEP



Vertical sweepline L originally located at $x = -\infty$.

L moves **continuously** across the plane to $x = +\infty$.

Devise appropriate data structures to maintain accurate representation of the current cross-section.

(CAN ONLY MAKE A FINITE NUMBER OF UPDATES)

Rectilinear Algorithms

p = # edges on final contour

Sweepline (segment tree)

(I) Lipski-Preparata, 1980 (RECTANGLES)

$O(n \log n + p \log \frac{n^2}{p})$ time — $O(n+p)$ space

(II) Wood, 1984 (RECTILINEAR POLYGONS)

$O(n \log n + p)$ time — $O(n+p)$ space

(Guting, 1984)

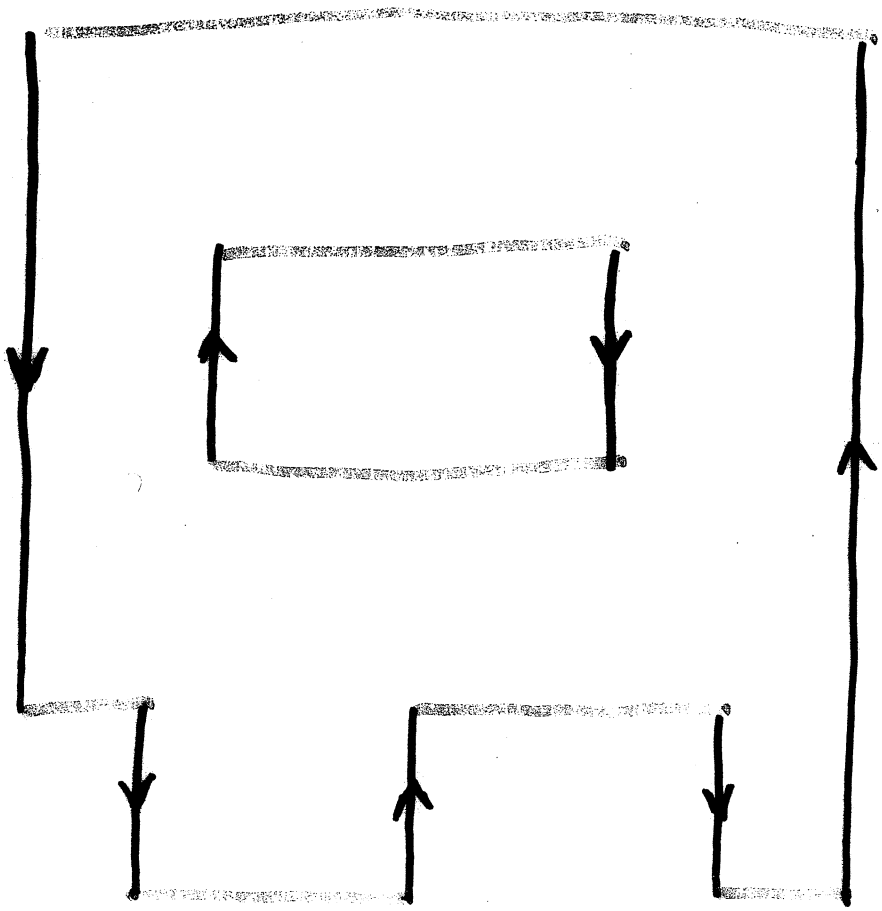
divide-and-conquer

(III) Guting, 1984 (RECTANGLES)

$O(n \log n + p)$ time — $O(n \log n + p)$ space

THE THREE ALGORITHMS SHARE 2 KEY TRAITS:

I) ONCE THE EDGES IN A SINGLE DIRECTION HAVE BEEN DETERMINED, THE OTHERS CAN BE DETERMINED IN LINEAR TIME



Lipski-Preparata - determine vertical edges

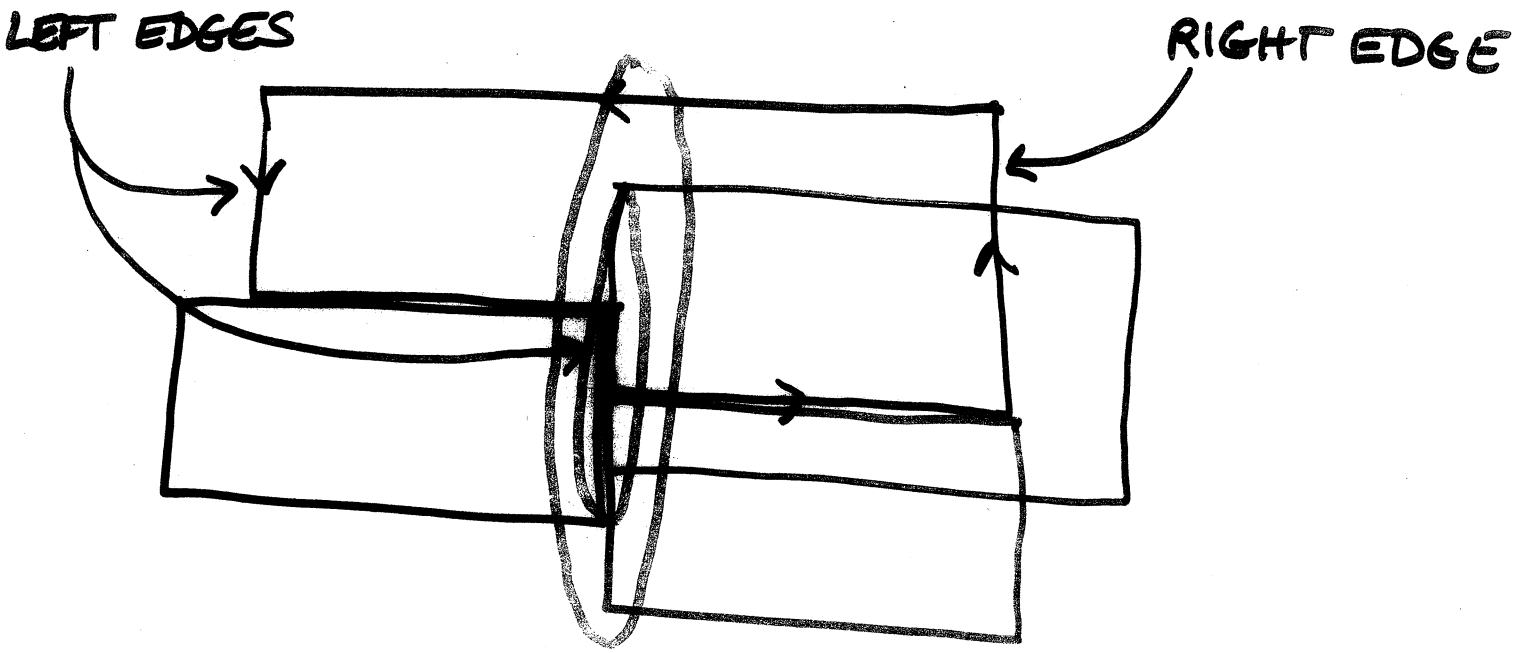
Wood
Guting } - determine horizontal edges

THE THREE ALGORITHMS SHARE 2 KEY TRAITS:

II) THE VERTICAL DIRECTION IS "SPECIAL."

TO RUN CORRECTLY, ALL VERTICAL EDGES MUST BE SORTED LEXICOGRAPHICALLY ON

1) X-COORDINATE
2) "LEFT" OR "RIGHT"
3) BOTTOM Y-COORDINATE



(EDGES ARE ORIENTED SO THAT THE INTERIOR OF THE POLYGON IS TO THE LEFT)

LIPSKI-PREPARATA BASED ALGORITHM (ALGORITHM I)

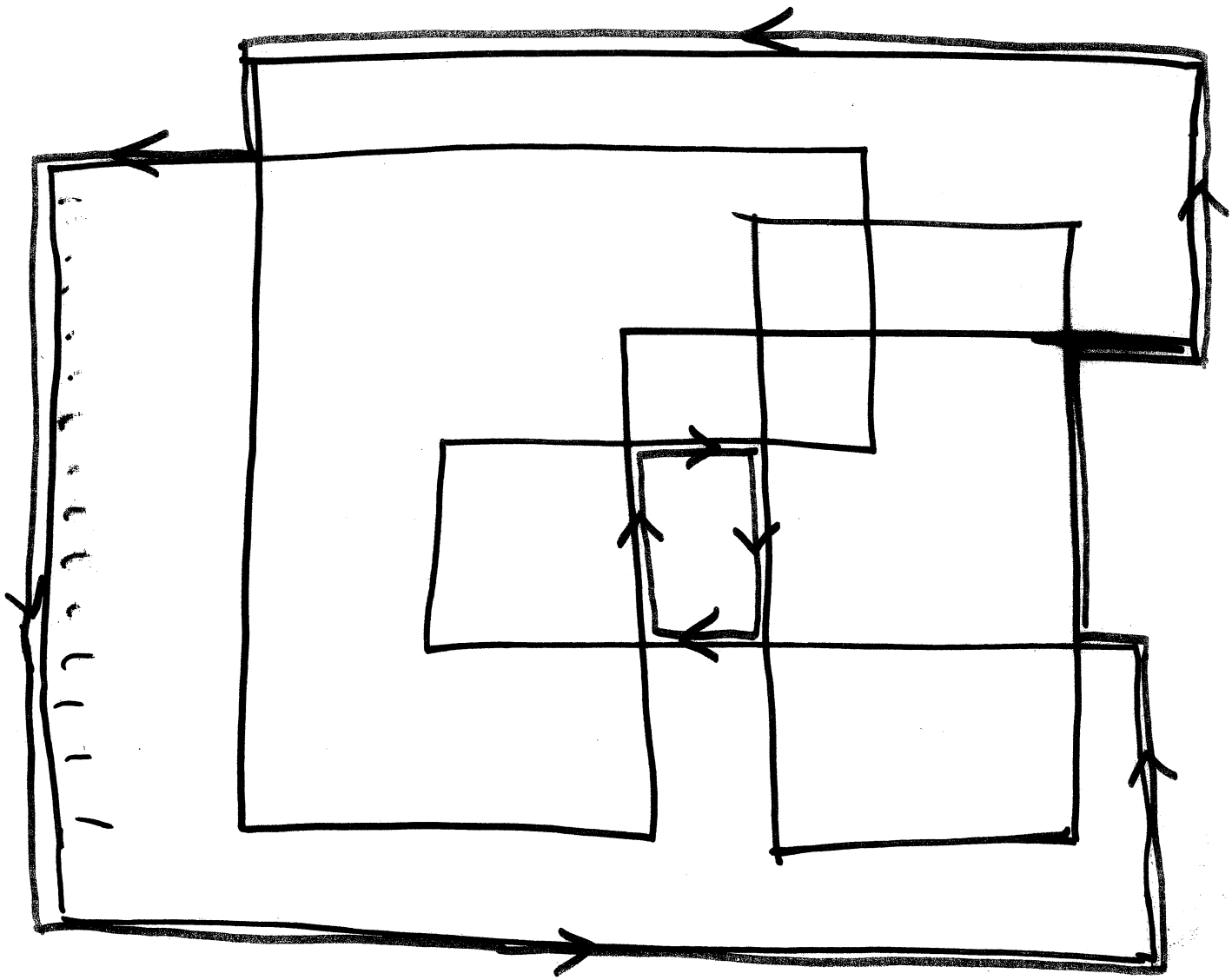
OUR CHANGES

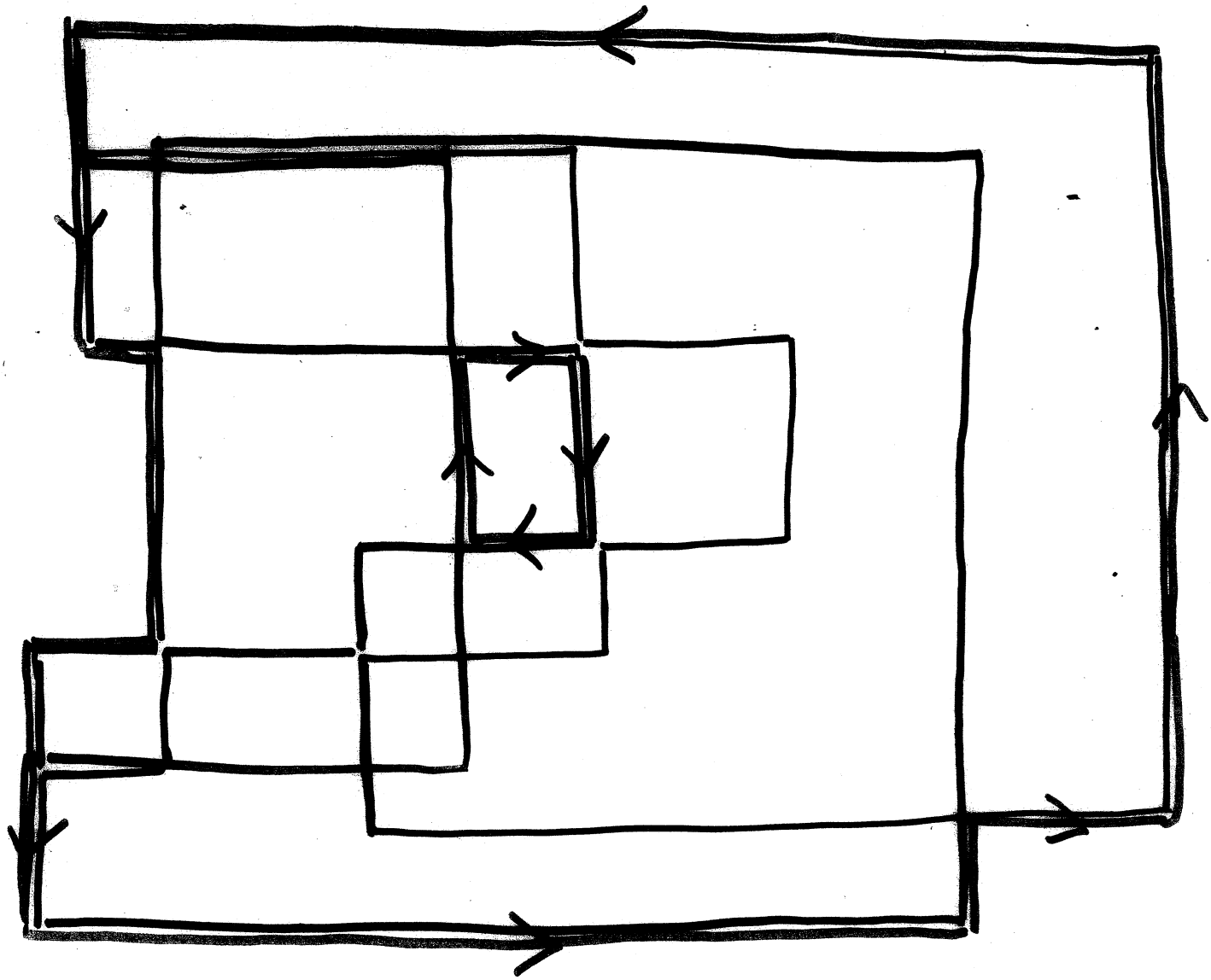
- CORRECT A BUG WHICH ALLOWS THEIR ALGORITHM TO REQUIRE $\Omega(n^2)$ TIME IN CERTAIN INSTANCES.

- GENERALIZE THE ALGORITHM TO APPLY FOR RECTILINEAR POLYGONS RATHER THAN JUST RECTANGLES.

(IMPLEMENTED BY DAMON ANTOS)

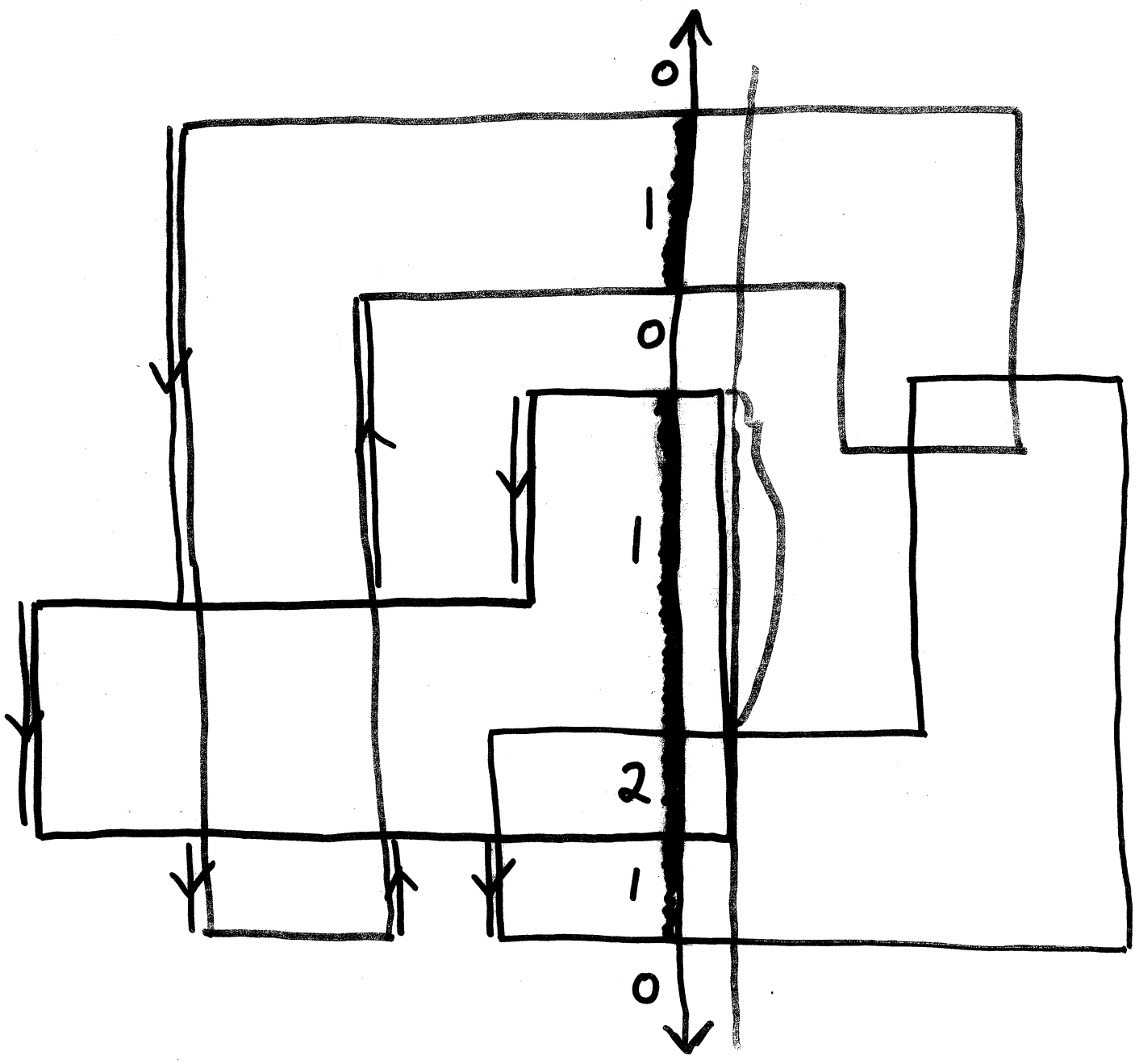
LEMMA: MATCHING EACH LEFT EDGE IN THE COLLECTION IN LEFT TO RIGHT ORDER WITH THE CLOSEST FRAGMENTS OF RIGHT EDGES PRODUCES A NEW COLLECTION OF POLYGONS WHOSE UNION IS IDENTICAL TO THE UNION OF THE ORIGINAL COLLECTION.





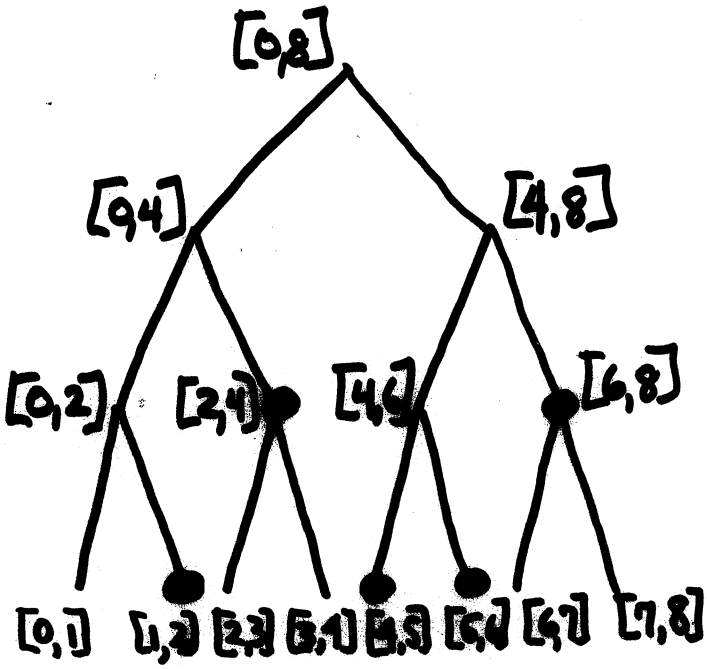
ALGORITHM 1 - $O(n \log n + p \log(n/p))$

1) SEGMENT TREE ON ALL Y-COORDS.
2) PROCESS VERTICAL EDGES
3) WITH EACH LEFT (RIGHT) EDGE, COMPUTE ITS INTERSECTION WITH THE COMPLEMENT OF THE CROSS-SECTION PRIOR TO (AFTER) ITS INSERTION (DELETION).
4) UNION COLLINEAR FRAGMENTS.

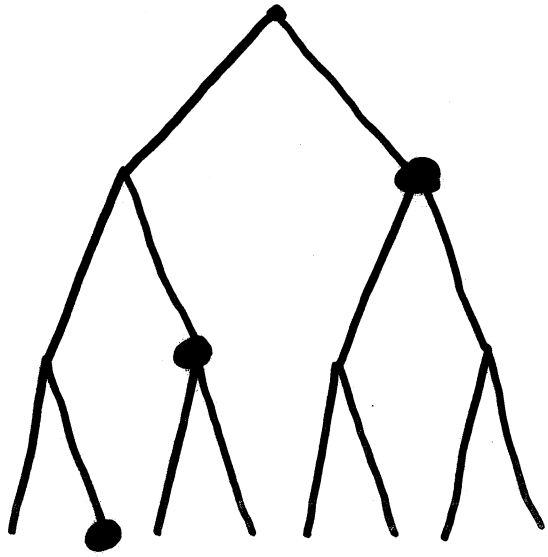


SEGMENT TREE ON $\{0,1,2,3,\ldots,8\}$

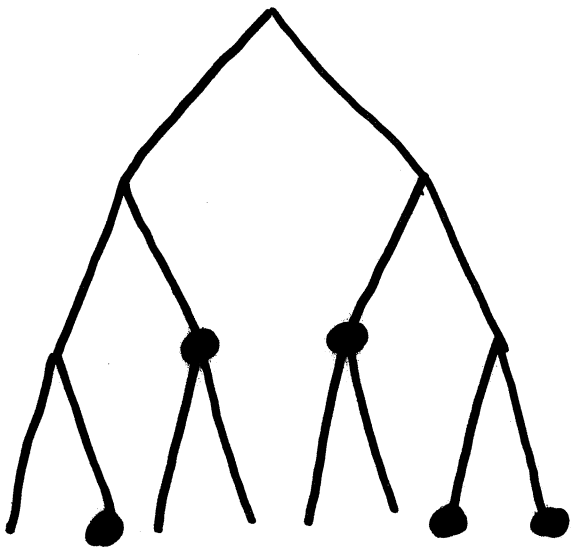
SHADED NODES v HAVE $C[v]=1$;
OTHERWISE $C[v]=0$.



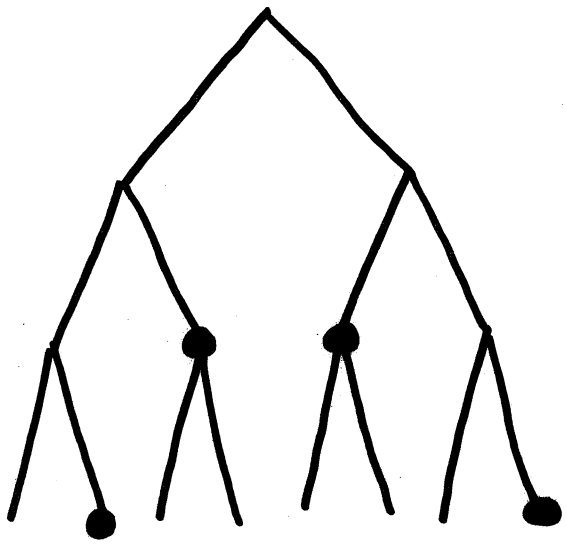
INSERT $[1,5]$ & $[5,8]$



PROMOTE



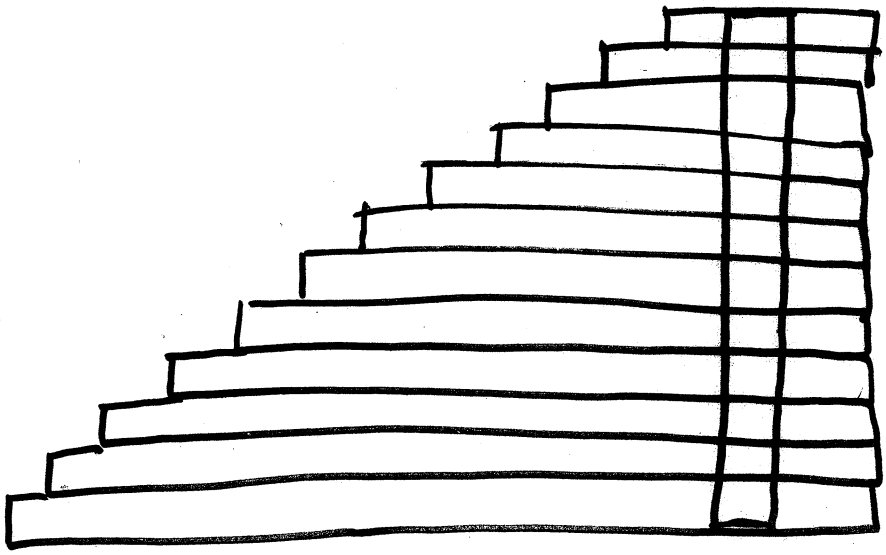
DEMOTE BEFORE DELETING $[6,7]$

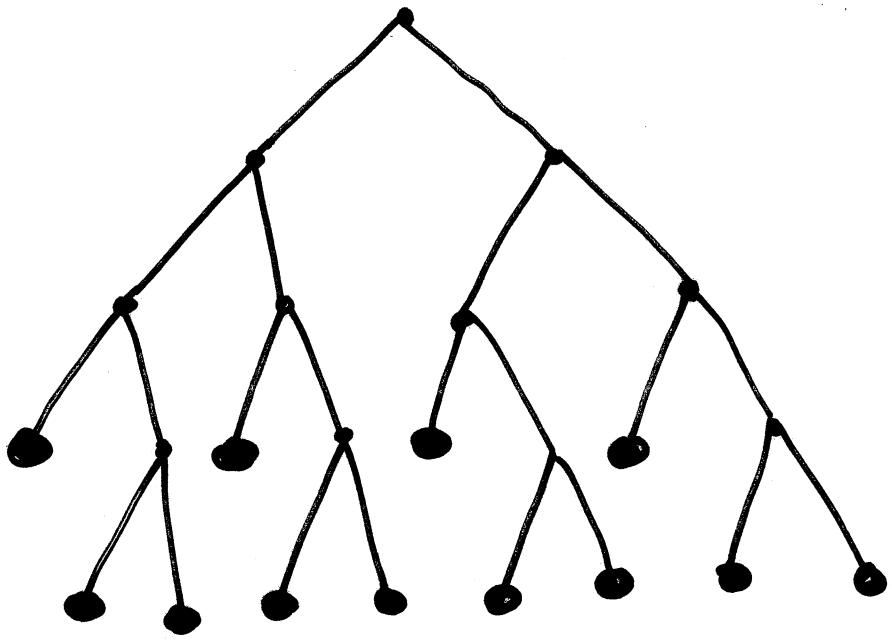


DELETE $[6,7]$

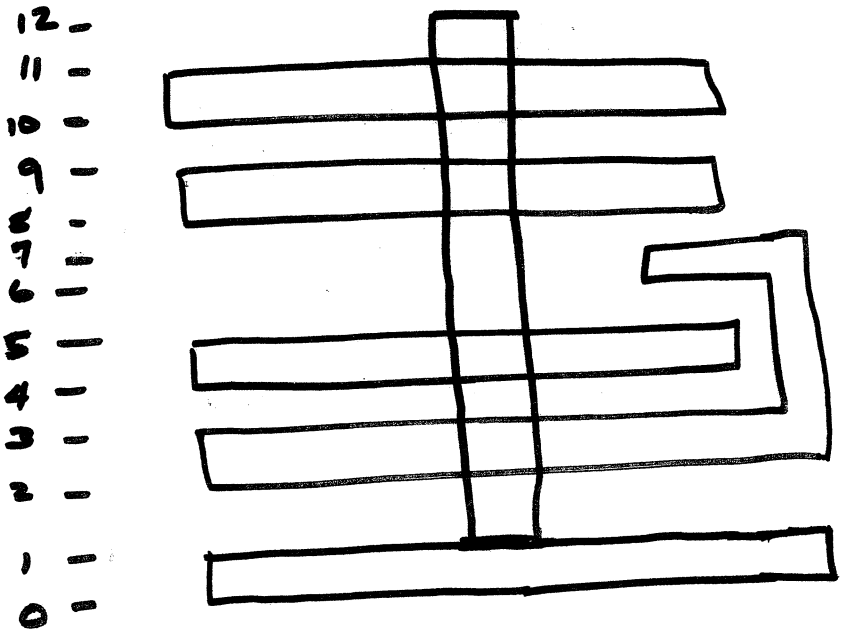
A NODE v HAS $P(v)=1$ IF SOME NODE u IN ITS SUBTREE HAS $C(u)=1$; ELSE $P(v)=0$.

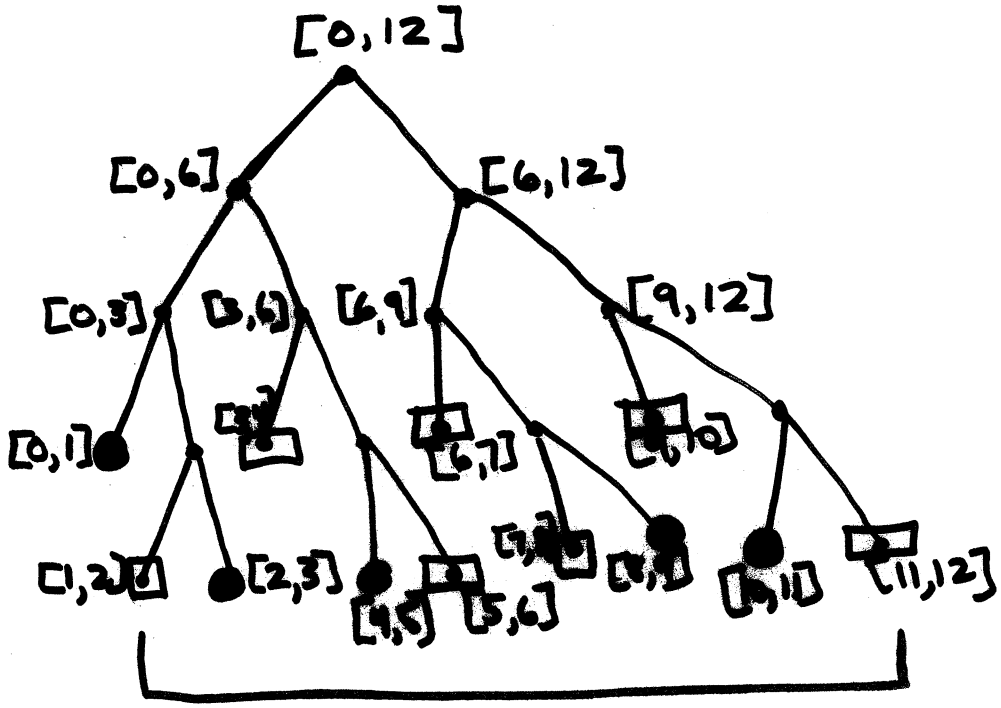
FLAW IN ORIGINAL ALGORITHM





ANALYSIS





LEMMA: If there are v end-nodes in a subtree containing n end-nodes, the total path length of the subtree is $O(v - \log \frac{16n}{v})$.

KEY FACT: Every other endnode is productive.

WOOD-BASED ALGORITHM

(ALGORITHM II)

OUR CHANGES

- APPLY THE LEMMA, PROMOTE, AND DEMOTE TO SIMPLIFY THE ALGORITHM.

- SIMPLIFY BOOKKEEPING AT THE LEAVES OF THE SEGMENT TREE BY HAVING ONE LEAF FOR $(i, i+1)$ AND ANOTHER LEAF FOR $[i, i]$.

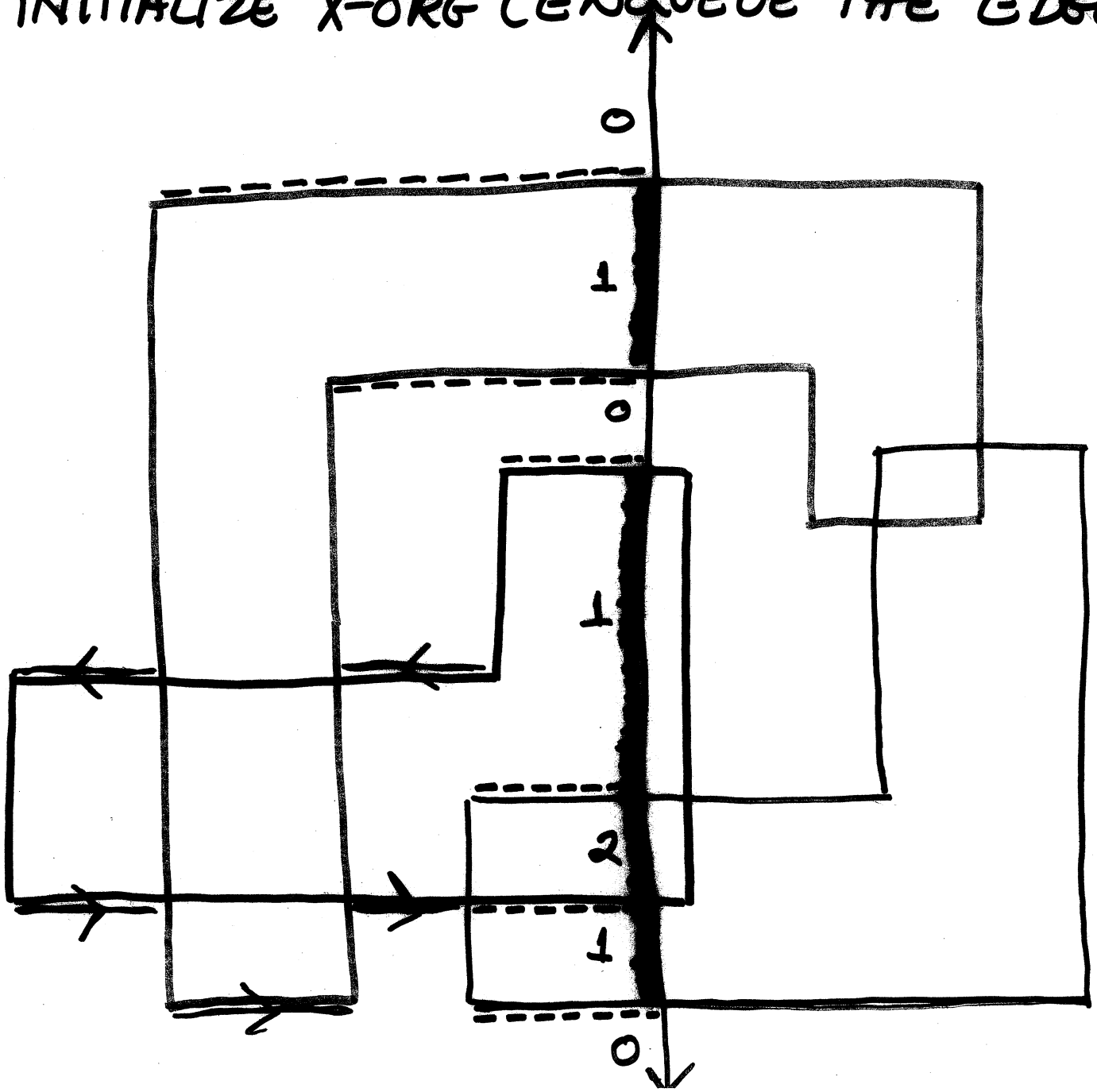
vertical interval

horizontal edge

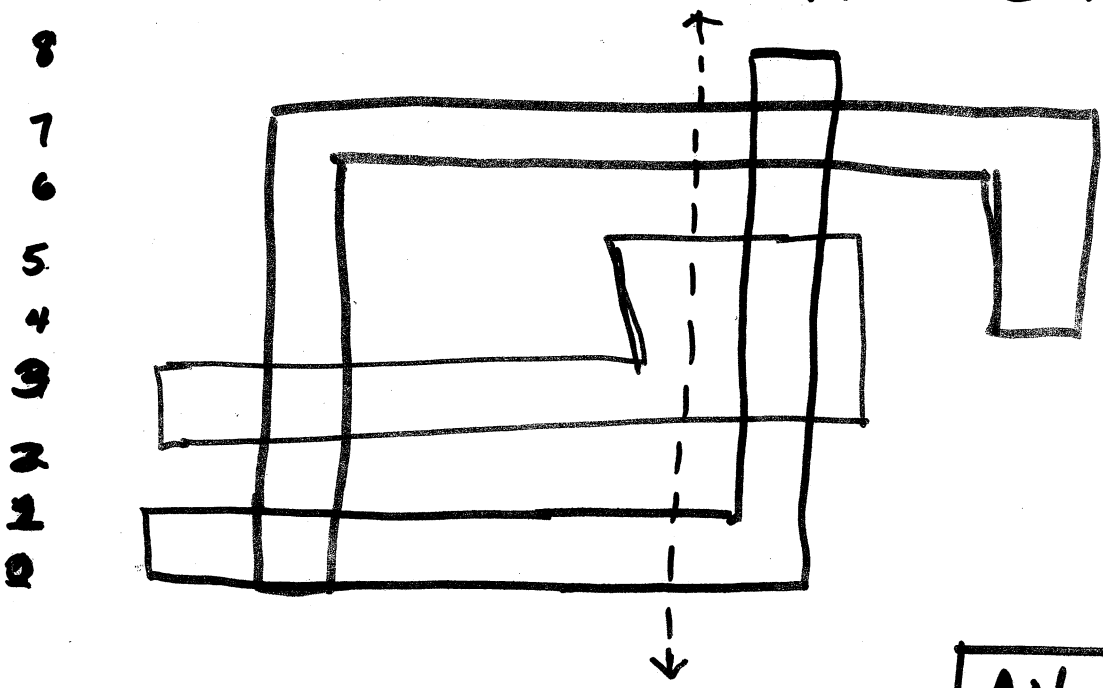
(IMPLEMENTED BY ASHOK KUMAR)

ALGORITHM II - $O(n \log n + p)$

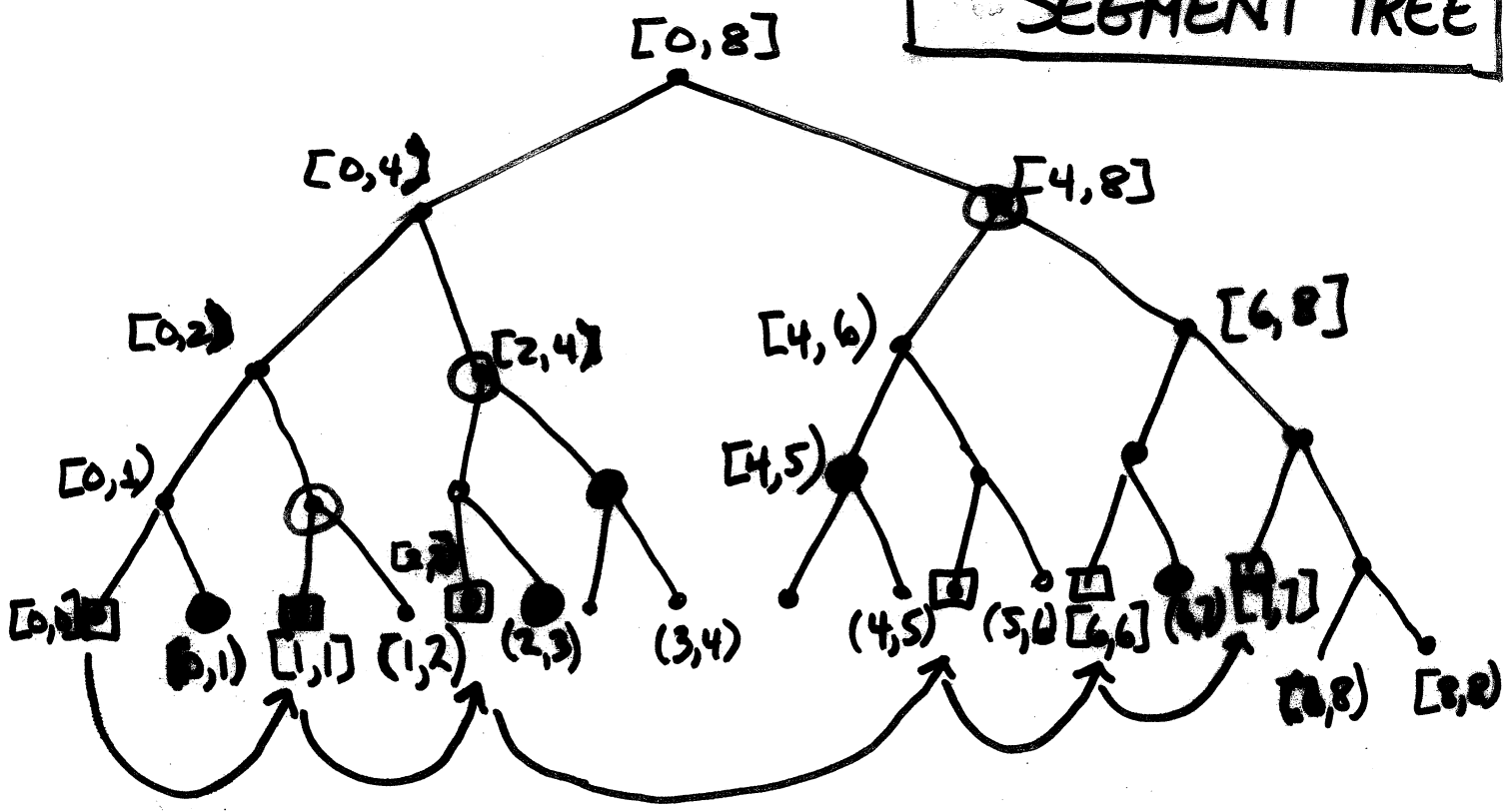
1) SEGMENT TREE WITH ADDITIONAL LEVEL + DATA
2) PROCESS VERTICAL EDGES AND THEIR ENDPOINTS
3) WHEN RUNNING HORIZONTAL EDGE GETS COVERED, TERMINATE IT AT CURRENT X AND ENQUEUE IT; GETS UNCOVERED, RESET X-ORG; WHEN HORIZONTAL EDGE IS FIRST INSERTED (LAST DELETED), INITIALIZE X-ORG (ENQUEUE THE EDGE).



WHY IS THIS ALGORITHM MORE EFFICIENT THAN THE FIRST?



AN AUGMENTED SEGMENT TREE



CONSTANT TIME PER OUTPUT EDGE

An external node v corresponding to a horizontal edge contains:

above[v] = difference between the number of polygons covering the region above v and the region below v, provided the first number exceeds the second.

below[v] — defined analogously.

hits[v] = number of horizontal edges with this y-coordinate overlapping at the current x-coordinate

V[v] = <xorg, dir, next>

xorg = first x-coord. where this horizontal edge might be visible

dir = orientation of the edge

next = pointer to next horizontal edge visible in the same subtree.

All nodes u, internal or external, contain:

fv[u] = the first horizontal edge (external node) visible in u's subtree.

lv[u] = the last external node (horizontal edge) visible in u's subtree.

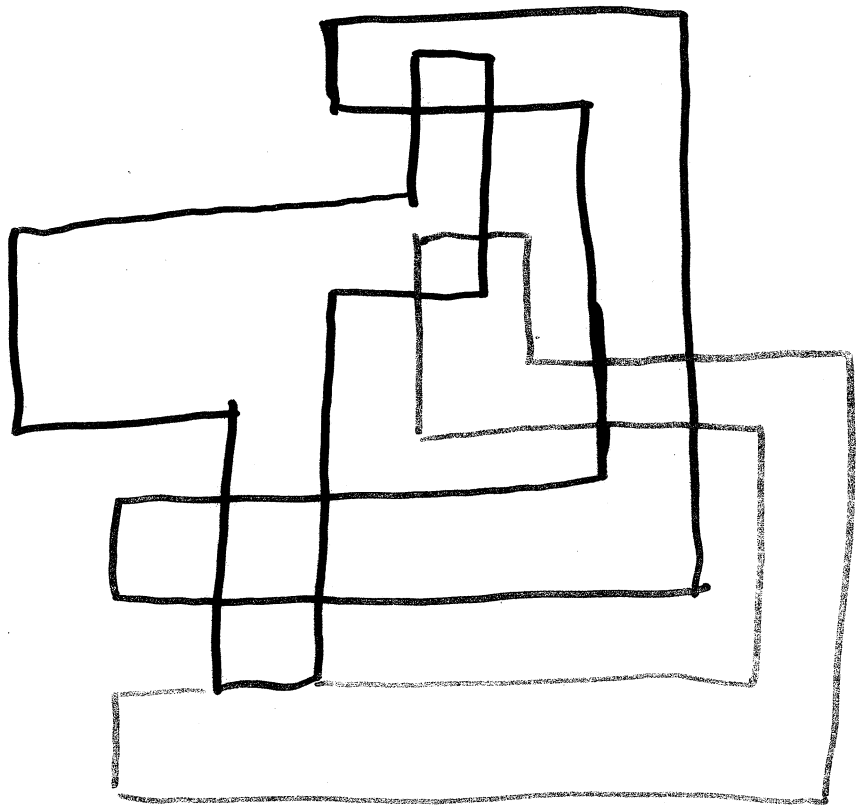
GUTING-BASED ALGORITHM
(ALGORITHM III)

OUR CHANGES

- ALTER THE BOOKKEEPING AND MERGE PROCEDURES SO THAT THE ALGORITHM APPLIES TO RECTILINEAR POLYGONS RATHER THAN JUST RECTANGLES.

(IMPLEMENTED BY THE SECOND AUTHOR)

DIVIDE AND CONQUER



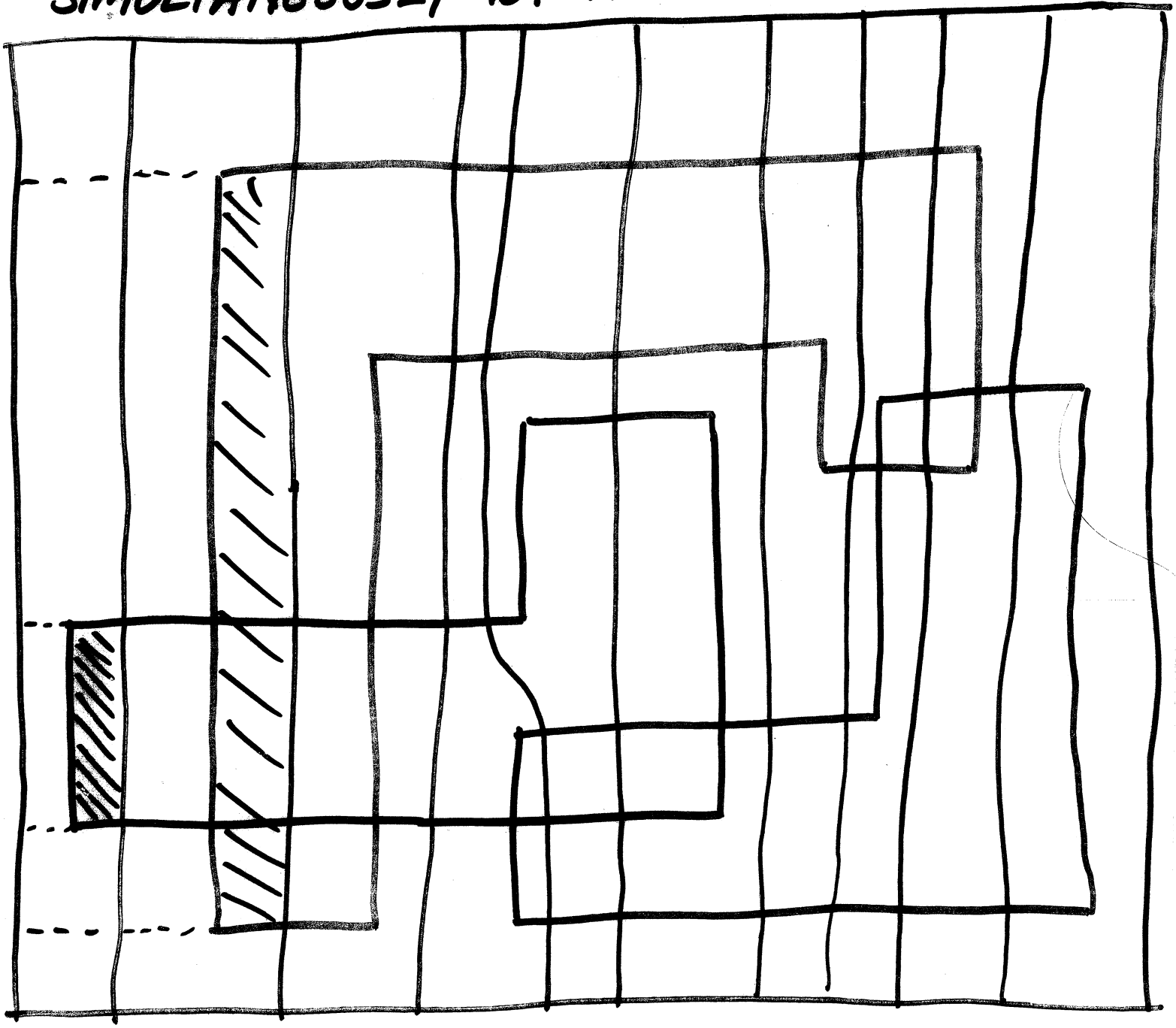
DIVIDE THE PROBLEM INTO SUBPROBLEMS

SOLVE (CONQUER) EACH SUBPROBLEM **RECURSIVELY**

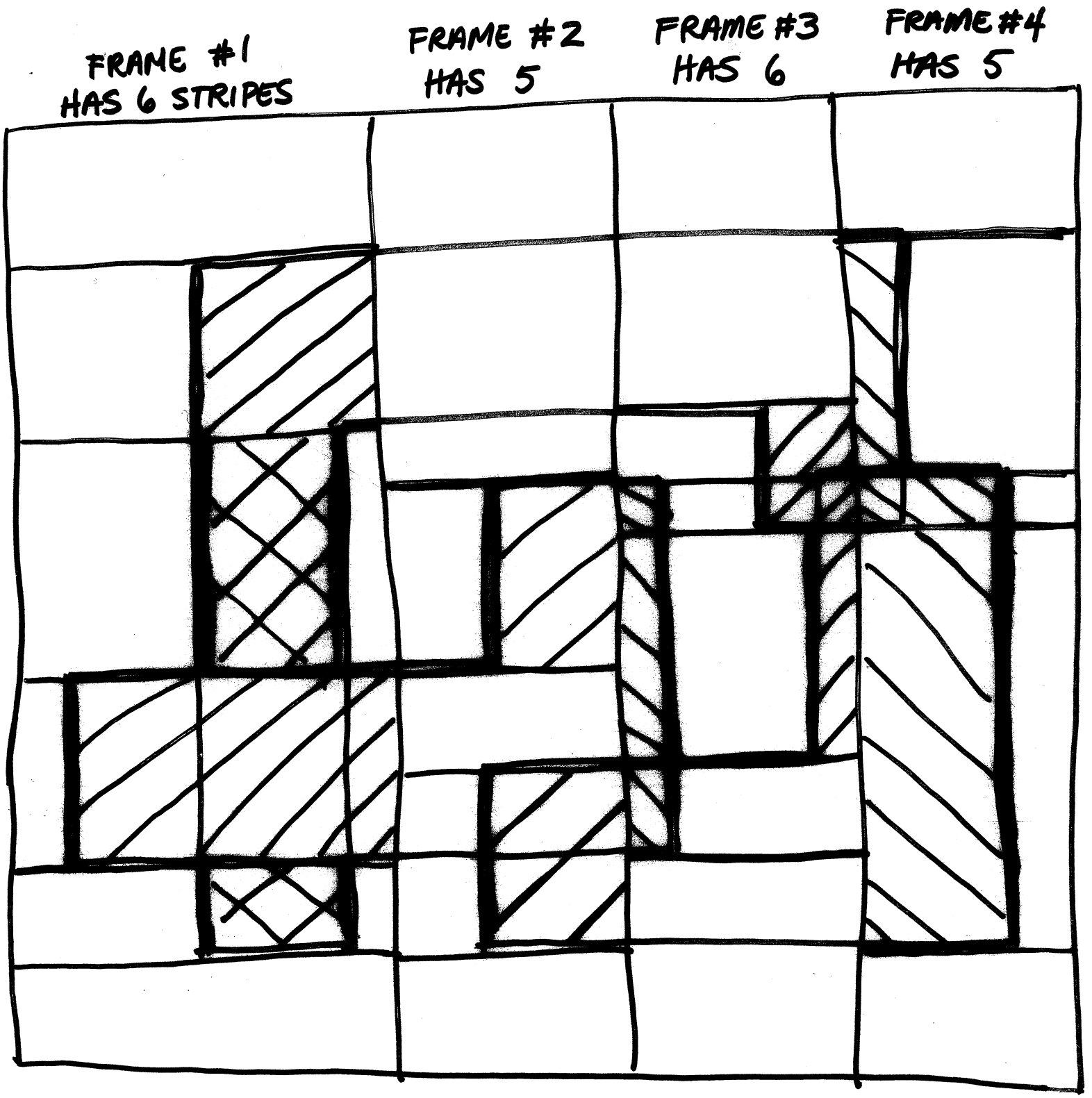
COMBINE THE PARTIAL SOLUTIONS TO FORM THE SOLUTION TO THE ORIGINAL PROBLEM.

ALGORITHM III - $O(n \log n + p)$

1) PUT A RECTANGULAR FRAME AROUND COLLECTION.
2) DIVIDE INTO VERTICAL SUBFRAMES OF 1 EDGE EACH
3) DIVIDE EACH SUBFRAME INTO STRIPES.
4) MERGE ADJACENT SUBFRAMES IN PAIRS.
5) SORT ORIGINAL HORIZONTAL EDGES, MERGING ADJACENT & OVERLAPPING EDGES
6) DETERMINE ALL FRAGMENTS NOT COVERED SIMULTANEOUSLY BY THE STRIPES ABOVE/BELOW.



ALGORITHM III - $O(n \log n + p)$



ANALYSIS

$O(n \log n + p)$ TIME

↑ sorting & divide-and-conquer

↑ output

BUT

$O((n \log n) + p)$ SPACE

- AT EVERY MERGE, NEW SETS OF STRIPES ARE CREATED
- EACH NEW STRIPE GETS ONE NEW NODE, WHICH THEN POINTS TO 0, 1, OR 2 OLD NODES.
- THE NUMBER OF NEW STRIPES IS LINEAR IN THE SIZE OF THE LISTS BEING MERGED

$\Rightarrow S(n) \leq 2S(\frac{n}{2}) + n = O(n \log n)$.

SLANTED EDGES

ADD ADDITIONAL ORIENTATIONS $\Rightarrow$ INTERSECTION POINTS HAVE "NEW" X & Y COORDINATES

SLOPES $0, \pm 1, \infty$ $\Rightarrow$ INTEGER MULTIPLES OF $1/2$
(multiply grid by 2)

SLOPES $0, \pm 1/2, \pm 2, \infty$ $\Rightarrow$ INTEGER MULTIPLES OF $1/3$, $1/4$ OR $1/5$
(multiply grid by 60)

$0^\circ, \pm 30^\circ, \pm 60^\circ, 90^\circ$ LINES $\Rightarrow$ SOME MULTIPLES OF $\sqrt{3}/2$.

A WISE SELECTION OF ORIENTATIONS ALLOWS THE CONTINUATION OF INTEGER OPERATIONS.

KEY TRAITS OF 3 RECTILINEAR ALGS:

1) LEFT-RIGHT ORDERING OF VERTICAL EDGES
2) DETERMINE CONTOUR EDGES OF A SINGLE ORIENTATION

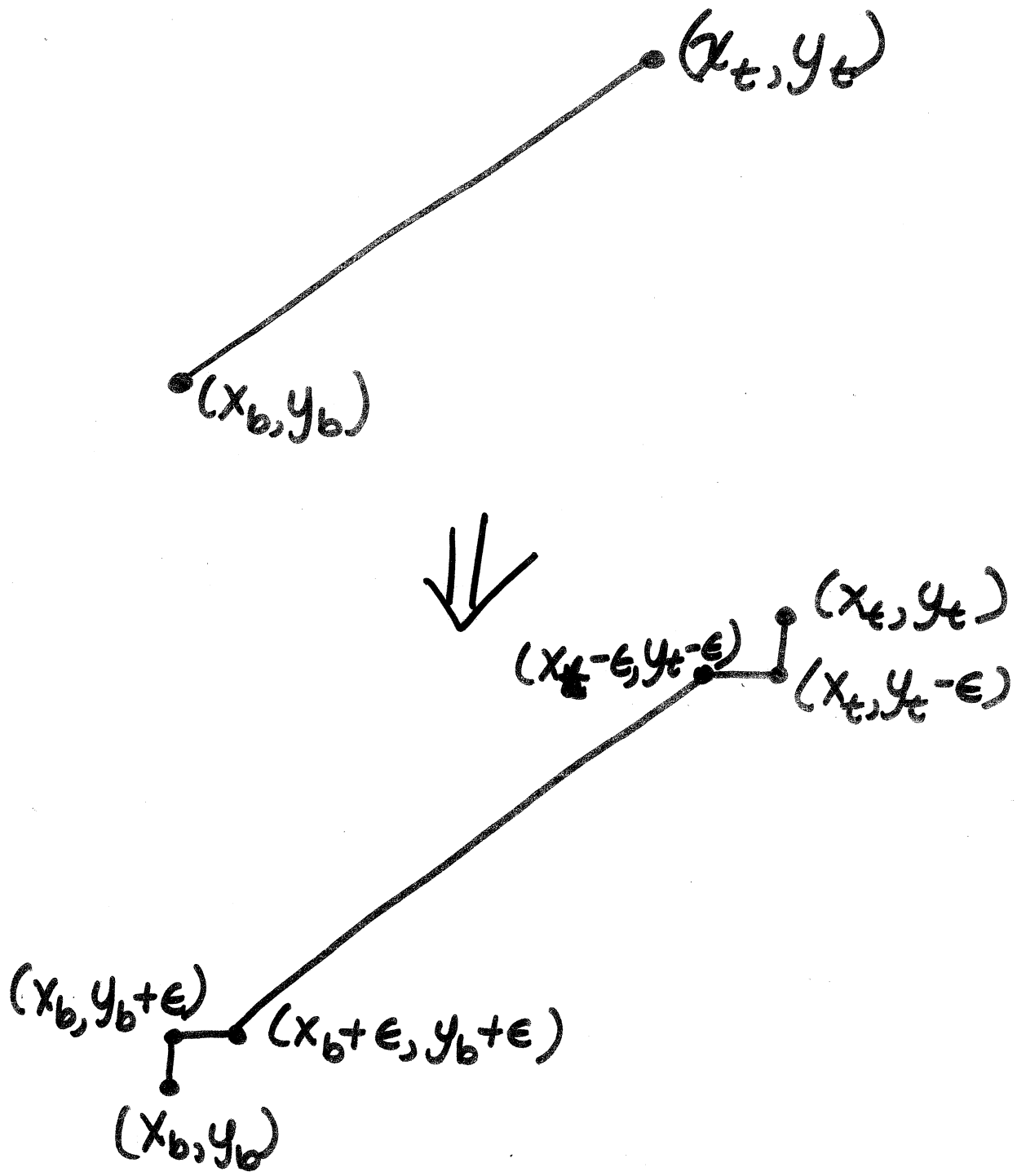
NEED TO MAKE SLANTED EDGES BEHAVE AS IF THEY WERE VERTICAL!

WHAT ARE THE CHARACTERISTICS OF VERTICAL EDGES?

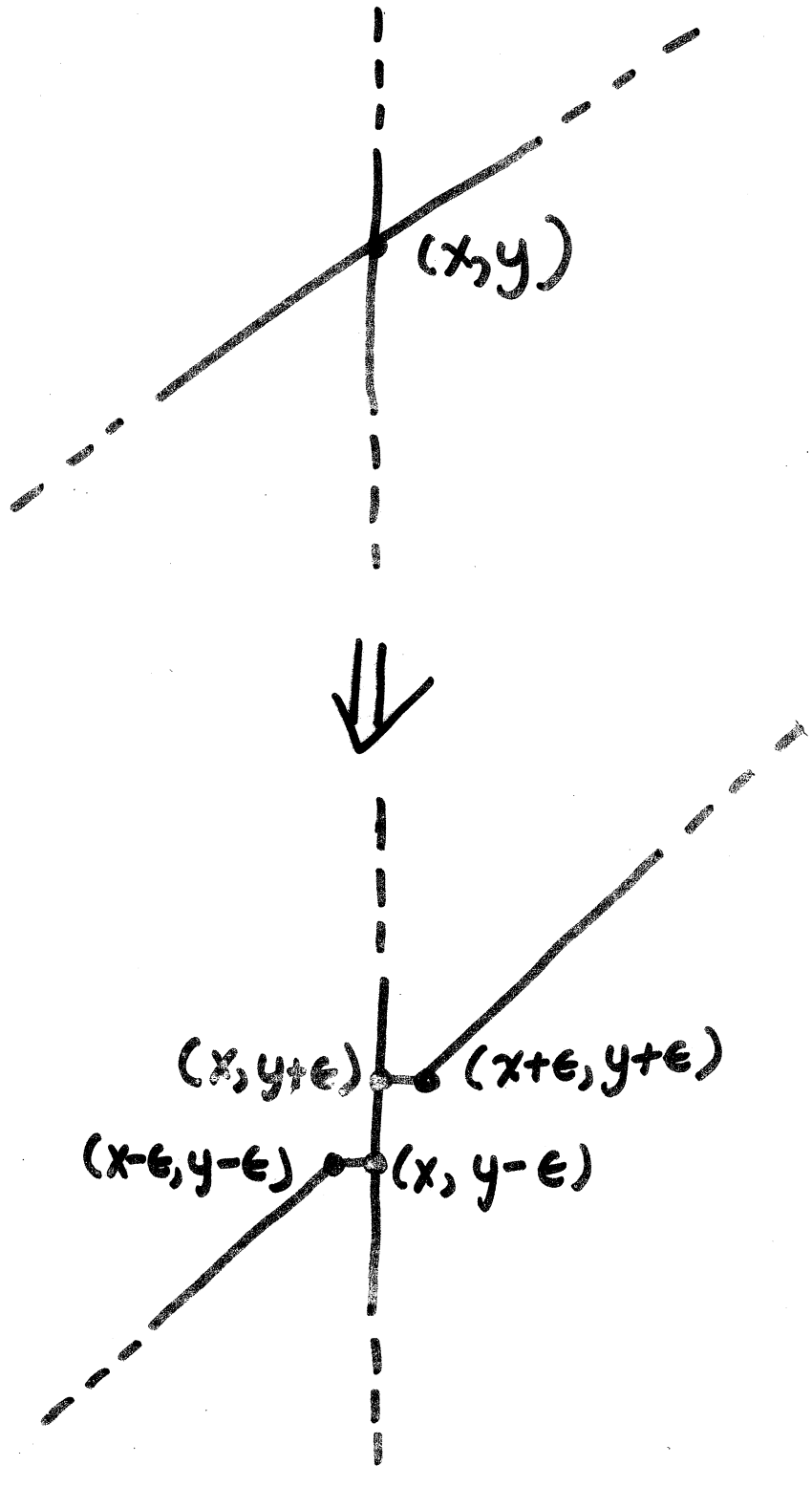
1) Each vertical edge adjoins a horizontal edge at each endpt.
2) No two vertical edges which are <u>not</u> collinear may intersect.

SOLUTION:

REPLACE EACH SLANTED SEGMENT BY A 5 SEGMENT CHAIN.

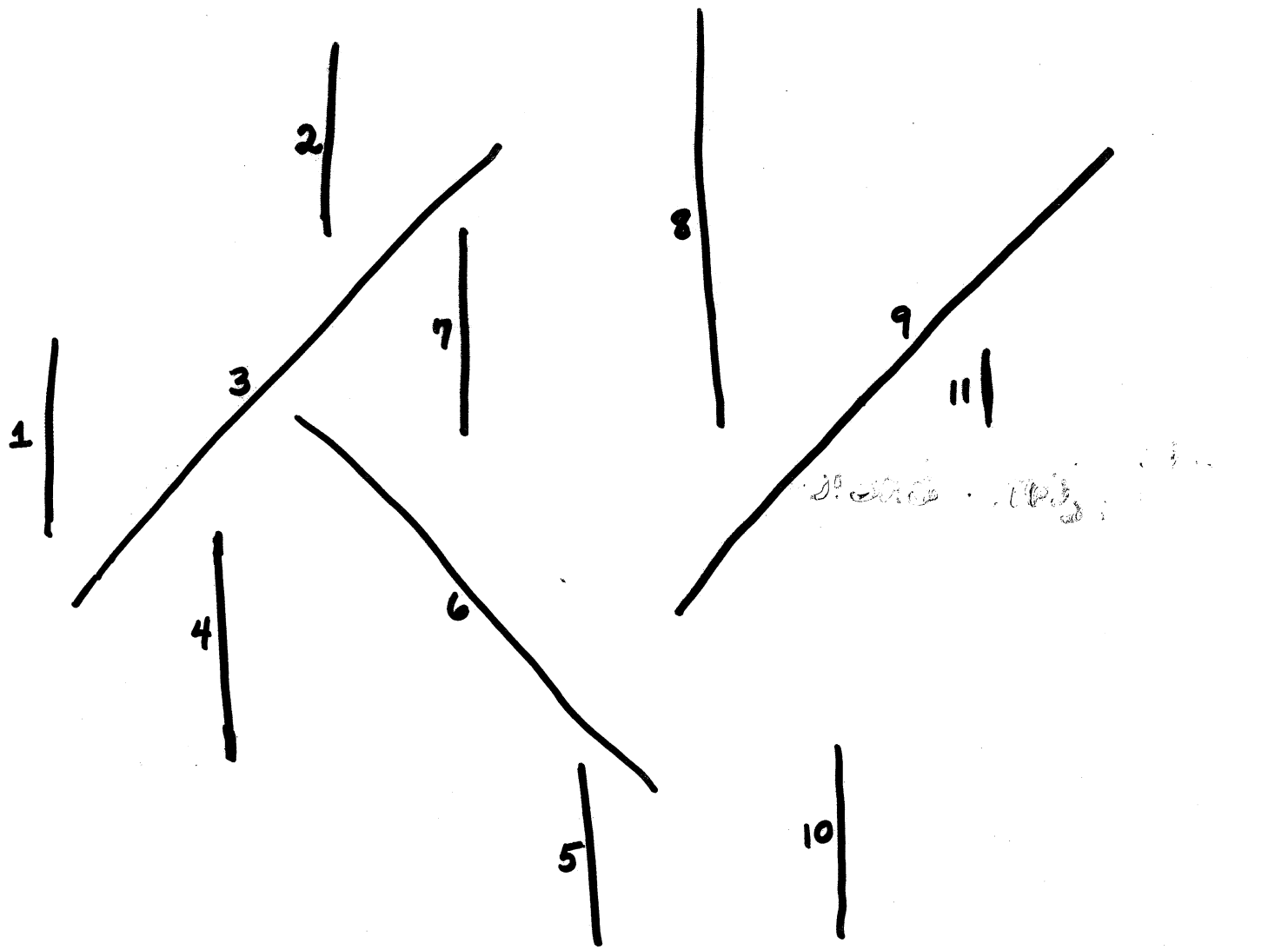


DETERMINE ALL INTERSECTIONS INVOLVING SLANTED EDGES.
LET S = # of such intersections.
DECOMPOSE SLANTED EDGES FURTHER.



POLYGONS NOW HAVE HORIZONTAL AND VERTICAL AND PSEUDO-VERTICAL EDGES. CONSIDER THE VERTICAL AND PSEUDO-VERTICAL EDGES

NEED LEFT-RIGHT ORDER



SORT THE EDGES OF EACH TYPE AND FIND MAXIMAL-SIZED PIECES ALONG EACH LINE.

CREATE A NEW DIRECTED GRAPH WITH ONE NODE PER SEGMENT.

SWEEP A LINE FROM $Y=-\infty$ TO $Y=\infty$, ADDING A DIRECTED EDGE FROM EACH SEGMENT (NODE) TO THOSE SEGMENTS (NODES) EVER DIRECTLY TO ITS RIGHT.

TOPOLOGICALLY SORT THE GRAPH, ASSIGNING AN INDEX TO EACH NODE.

TO EACH ORIGINAL SLANTED OR VERTICAL EDGE, ASSIGN THE INDEX AWARDED TO THE MAXIMAL SEGMENT ON WHICH IT LIES

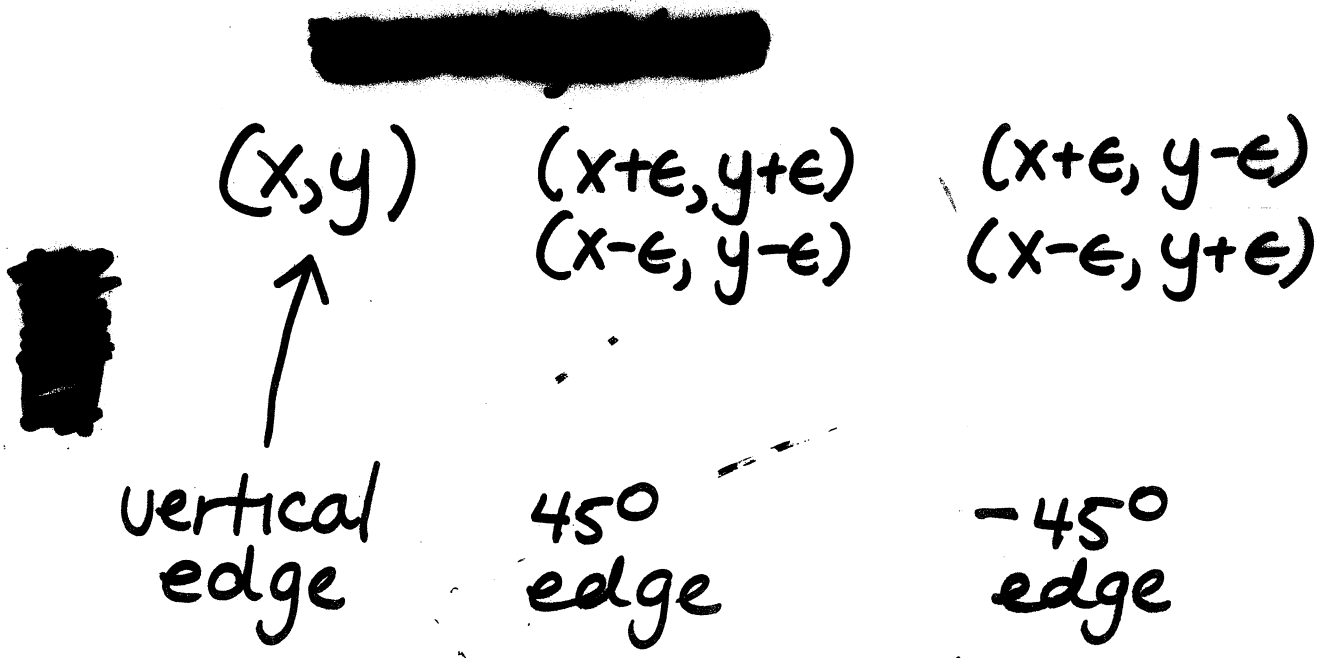
SORT EDGES IN ASCENDING ORDER OF INDEX & BOTTOM Y-COORD.

ALGORITHMS I, II & III RUN UNCHANGED EXCEPT FOR PRE- & POST-PROCESSING.

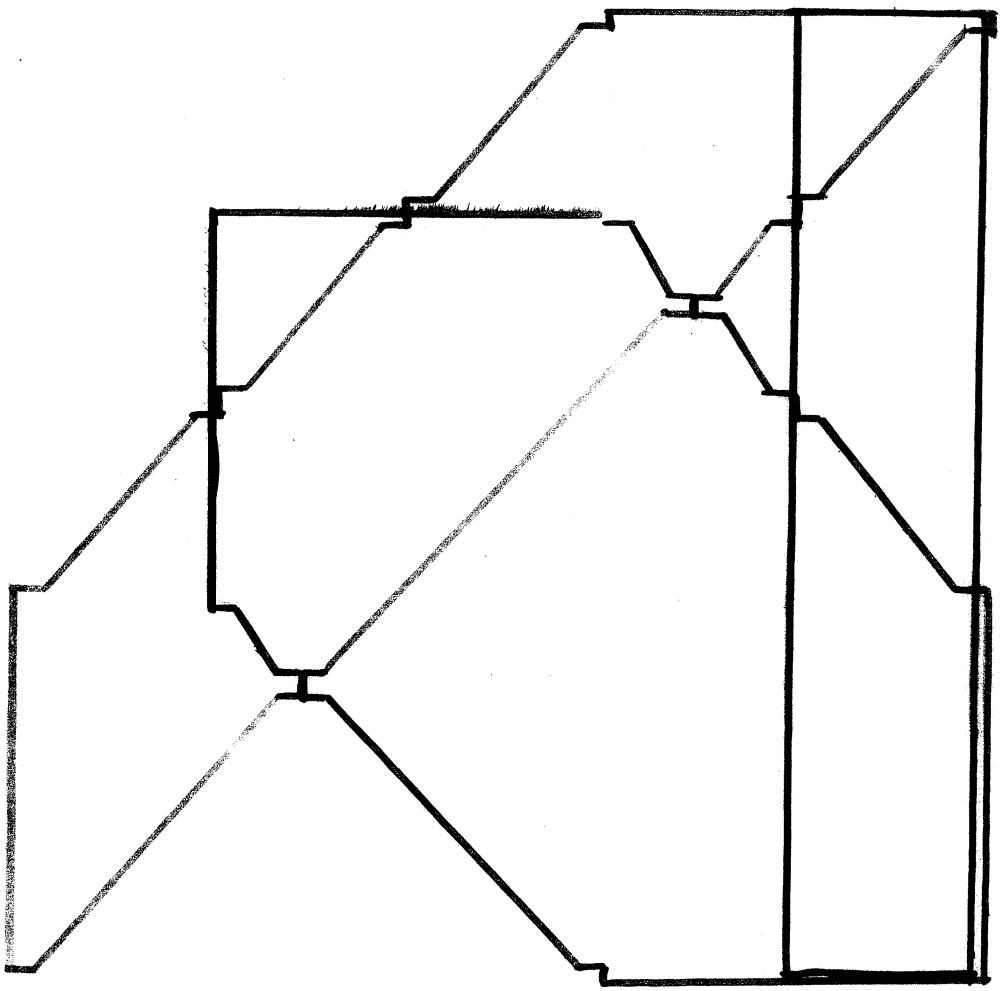
WHAT ABOUT POST-PROCESSING?

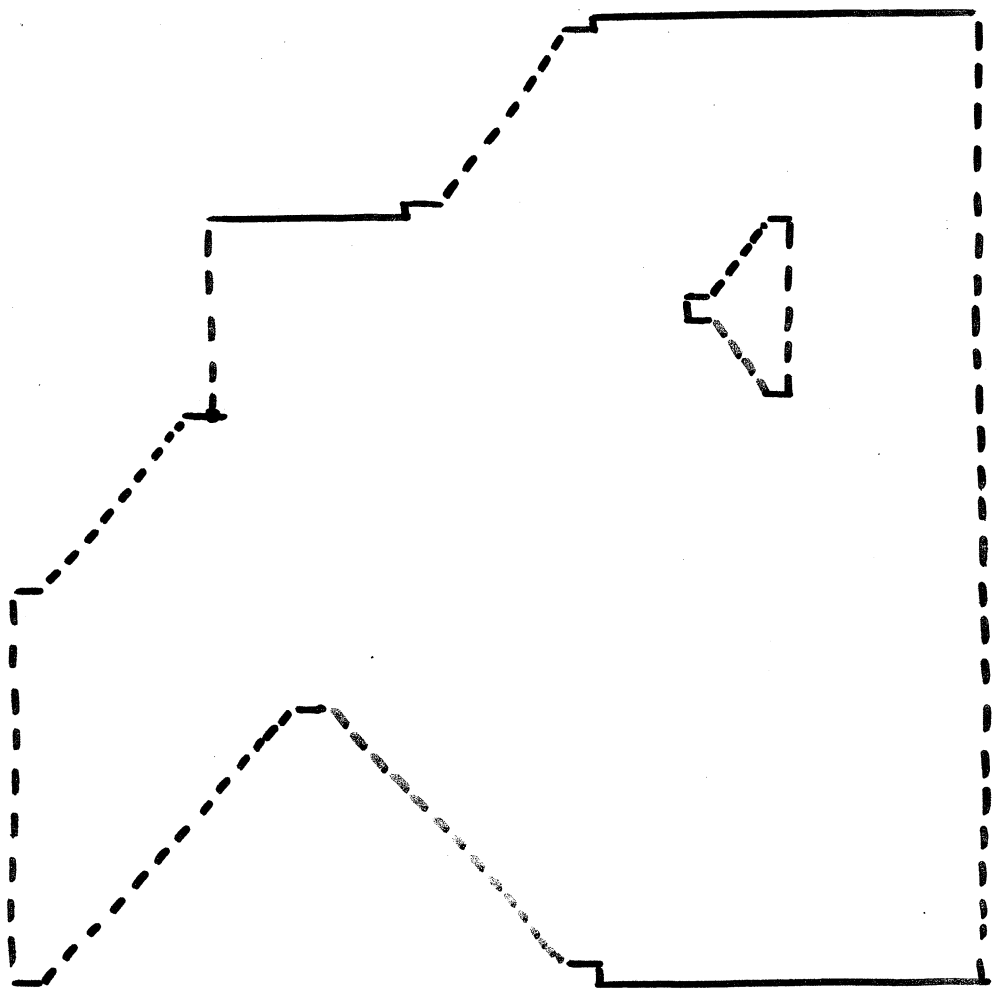
ALG. I DETERMINES ALL VERTICAL AND PSEUDO-VERTICAL CONTOUR EDGES ⇒ ORIGINAL POSTPROCESSING.

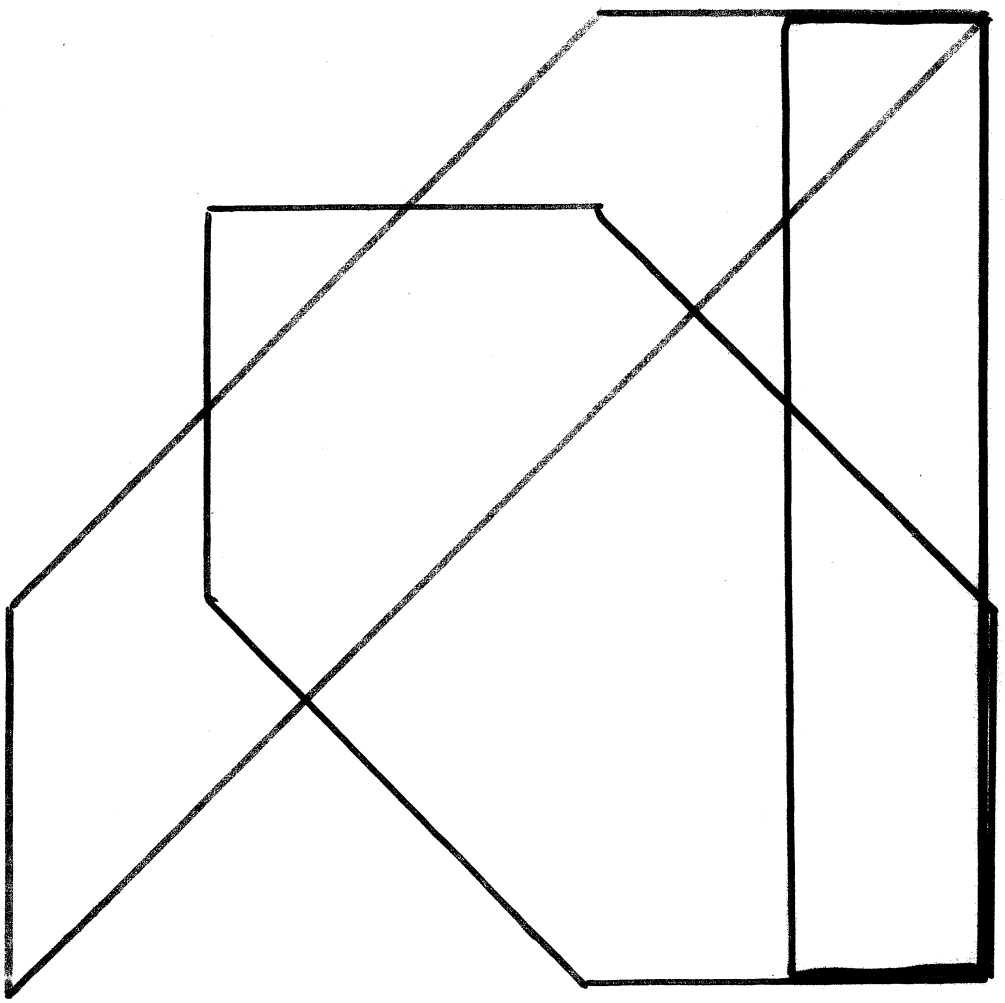
ALGS. II & III DETERMINE HORIZONTAL CONTOUR EDGES ⇒ NEED TO FIND VERTICAL, ±1 SLOPE EDGES BUT ENDPOINTS HAVE 3 TYPES:



SORT IN 3 PASSES!







CHANGE IN ASYMPTOTIC COMPLEXITY

REPLACE n IN RECTILINEAR BOUNDS BY $n+s$.

SOME RESEARCHERS HAVE ARGUED THAT s IS SMALL:

- MOST COMPONENTS ARE ALIGNED WITH CHIP BOUNDARIES
- CHIP BOUNDARIES FORM A RECTANGLE

???

What about adding additional orientations?

Suppose we have slopes $+1/2, +2, -1/2, -2$.

An edge of slope $+1/2$ from (x_b, y_b) to $(x_t, y_t) \Rightarrow$

$(x_b, y_b), (x_b, y_b+\epsilon), (x_b+2\epsilon, y_b+\epsilon), (x_t-2\epsilon, y_t-\epsilon), (x_t, y_t-\epsilon), (x_t, y_t)$

An edge of slope $+2$ from (x_b, y_b) to $(x_t, y_t) \Rightarrow$

$(x_b, y_b), (x_b, y_b+\epsilon), (x_b+\epsilon, y_b+\epsilon), (x_t-\epsilon, y_t-\epsilon), (x_t, y_t-\epsilon), (x_t, y_t)$.

TOTAL NUMBER OF EDGES ~~AFTER~~ DECOMPOSING:

$5n + 3s$

TOTAL NUMBER DISTINCT y-COORDINATES

$\leq 3n$

TOTAL NUMBER DISTINCT x-COORDINATES

$\leq (2d+1)n$

$\Rightarrow$ Size of the segment tree (number of stripes) does <u>not</u> change as d increases.

distinct x-coordinates affects the postprocessing phase: endpoints of horizontal edges will have the following forms: $x, x+\epsilon, x-\epsilon, \ldots, x+d\epsilon, x-d\epsilon$.

- endpoints of the form x are adjoined to other endpoints of the same type with vertical edges.
- an endpoint of the form $x+i\epsilon$ which came from a line with positive (negative) slope is connected by a segment of that slope to an endpoint of the form $x-i\epsilon$ above (below) and to the right of it.

The postprocessing phase can still be accomplished in $O(p)$.

TIME COMPLEXITIES:

ALGS II & III:	ALG I
$O((n+s)\log n + p)$	$O((n+s)\log n + p\log((n+s)^2/p))$

GENERAL ALGORITHMS

BASED ON LINE-SEGMENT INTERSECTION.

- BENTLEY-OTTMAN, 1979

  $O((n+k)\log n)$ time

  $O(n)$ computing space

  $O(k)$ reporting space

- CHAZELLE-EDELSBRUNNER, 1989

  $O(n \log n + k)$ time

  $O(n+k)$ computing space

  $O(k)$ reporting space

n = # segments

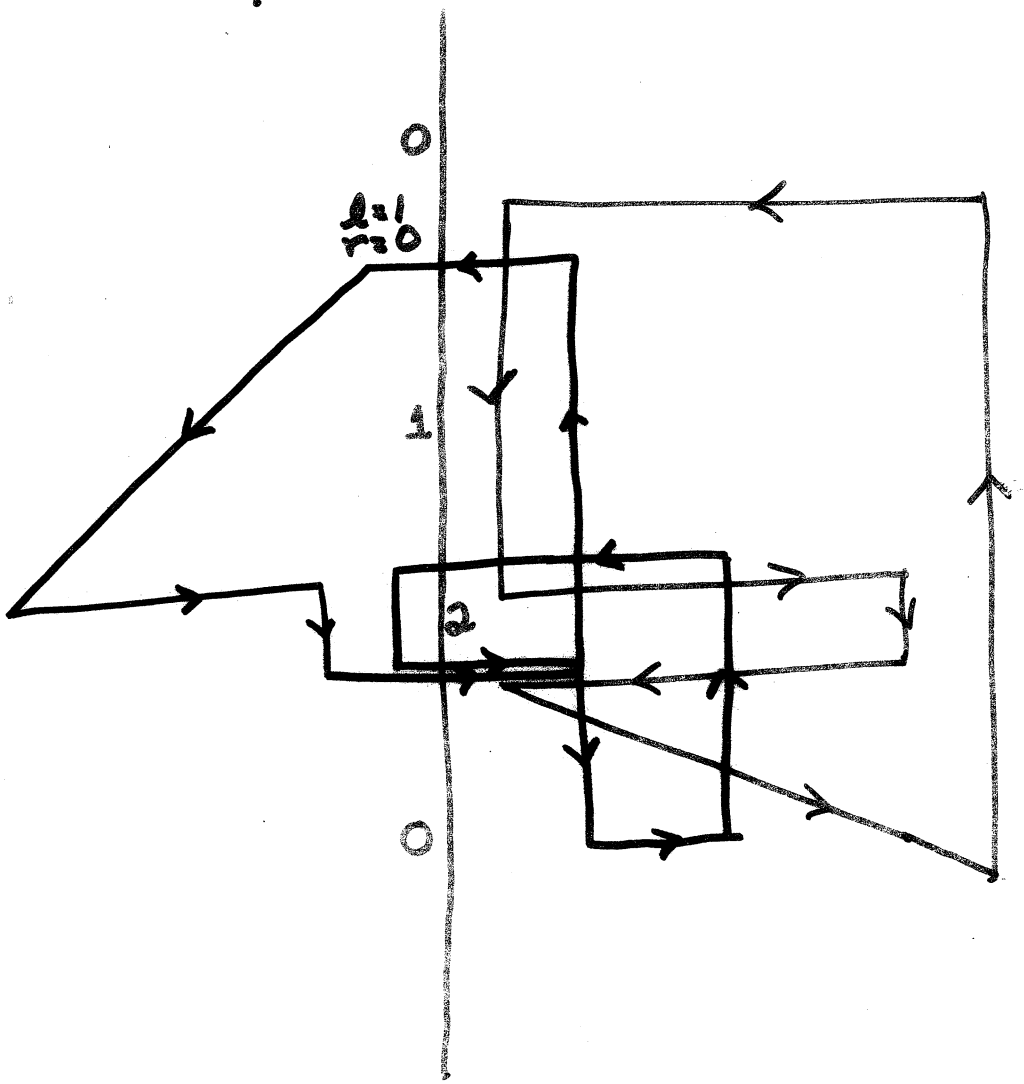
k = # intersections

ORIGINAL VERSIONS OF THESE ALGORITHMS ASSUME NO TWO EDGES OVERLAP AND NO THREE EDGES INTERSECT IN A POINT!

ALG. 1 SWEEPLINE L

MAINTAIN CROSS-SECTION AT CURRENT POSITION OF L IN DYNAMIC BINARY TREE.

EVENT POINTS STORED IN DYNAMIC PRIORITY QUEUE.



EVENT QUEUE INITIALLY CONTAINS BOTH ENDPOINTS OF EACH EDGE
(EVEN DUPLICATES!)

WHEN TWO EDGES ARE ADJACENT IN STATUS TREE, ANY INTERSECTION POINT BETWEEN THEM IN THE FUTURE IS INSERTED INTO EVENT QUEUE.

Several people have adapted the general sweepline algorithm to compute the contour of the union:

eg. Chiang, Nahar, Lo (1989)
Nievergelt, Preparata (1982)
Szymanski, Van Wyk (1983)

In our version:

- dynamic binary tree represents cross-section
- ignore vertical edges, keeping only slanted and horizontal edges in the tree
- each node keeps a counter for the region above the edge, and one for the region below
- each node keeps a leftcount l (rightcount r) indicating the number of overlapping left-directed (right-directed) edges.

PERFORMANCE

n = # edges
r = # intersections between slanted edges and horizontal edges or slanted edges
h = # intersections involving vertical edges
} $r+h = k$
p = # contour edges

- $n+r$ stopping points, $O(\log n)$ each
- h intersection points are processed at constant time each

$O((n+r)\log n + h)$ TIME

$O(n)$ COMPUTATION SPACE

$O(p)$ REPORTING SPACE

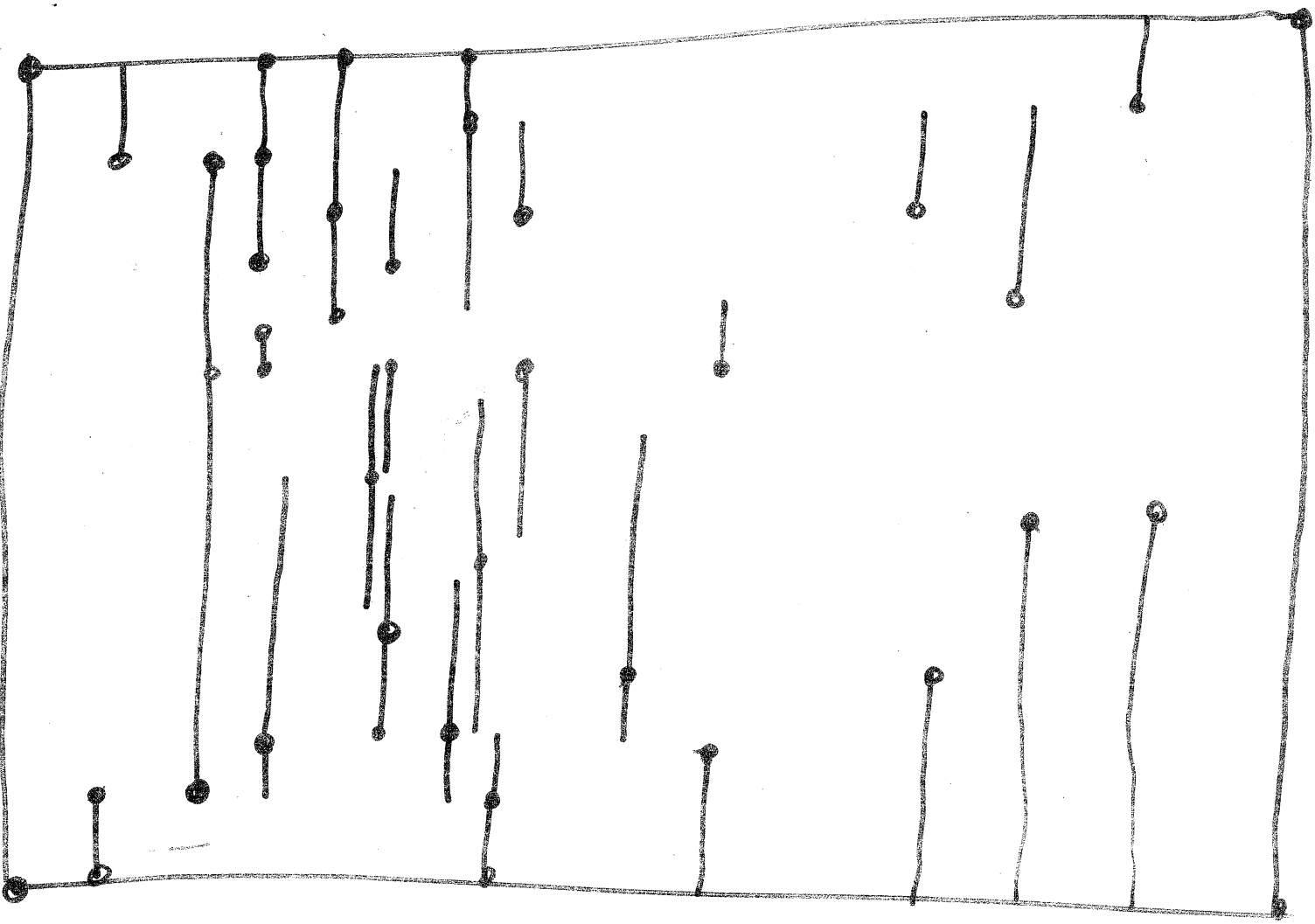
(Szymanski & Van Wyk use premise that any horizontal or vertical line crossing a chip intersects an expected $O(\sqrt{n})$ edges.

Present a general algorithm similar to alg. I with expected main memory space requirement of $O(\sqrt{n})$)

CAVEAT: Chazelle's implementation of the Chazelle-Edelsbrunner algorithm does not currently handle multiple intersection nor overlapping edges. Furthermore, it does not preserve the vertical map.

Levine's implementation of the Clarkson probabilistic algorithm does construct the vertical map and does handle multiple intersections but does not handle overlapping edges which occur frequently in VLSI design.

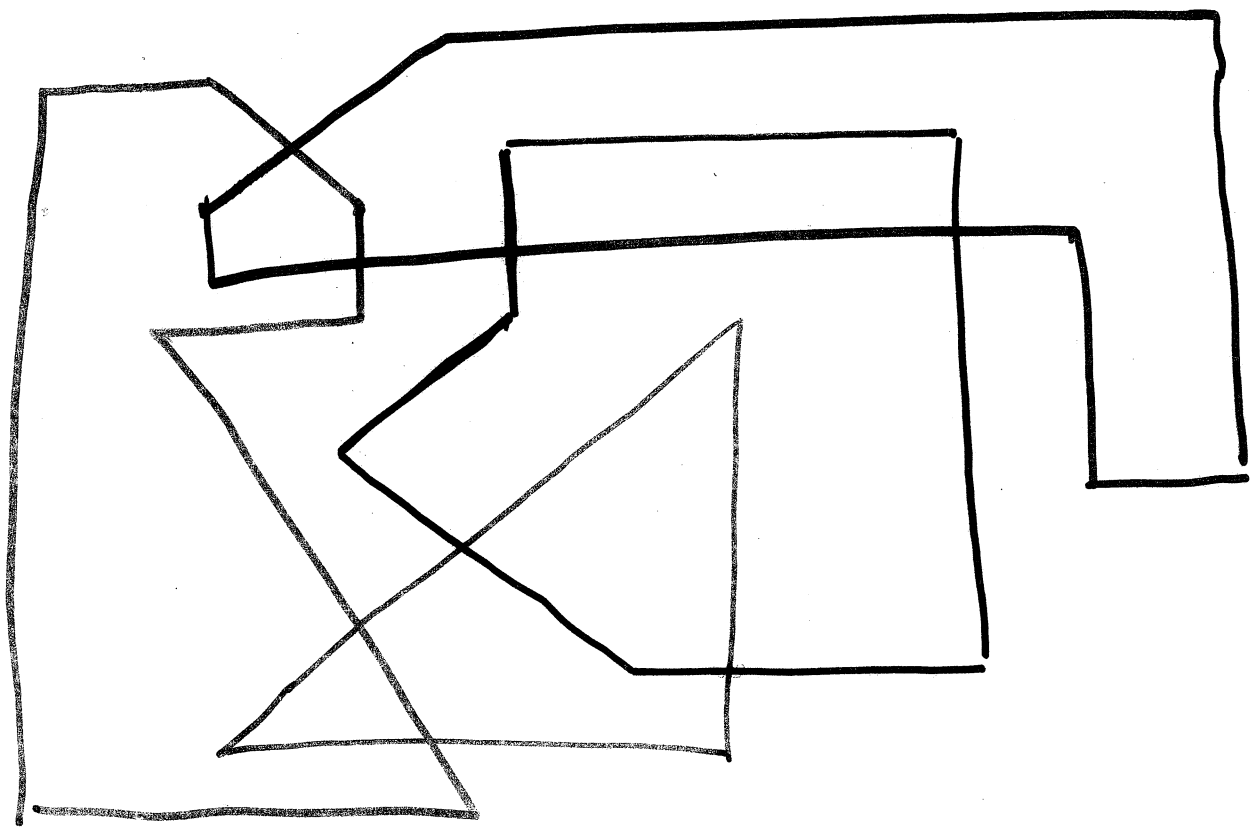
- ORIENT EACH EDGE UP OR TO RIGHT
- AUGMENT EACH NODE WITH # OF PREDECESSORS
- DOUBLY LINKED LIST KEEPS STATUS OF SWEEP.
- QUEUE KEEPS NEXT EVENT.
- INITIALIZE QUEUE WITH BOTTOM LEFT POINT (ONLY POINT WITH NO PREDECESSORS).



ALG. II - TOPOLOGICAL SWEEPLINE L (CURVED).

FIRST, APPLY (GENERALIZED) CHAZELLE-EDELS. ALGORITHM TO PRODUCE VERTICAL MAP.

THEN, SWEEP TOPOLOGICALLY.



Strange version of topological sweep:

topological sweep is normally applied to arrangements of lines given implicitly by equations of lines

upper horizon trees & lower horizon trees are maintained to give partial information about the underlying planar graph.

here, the planar graph is given explicitly

no horizon trees are necessary

- sweep a curved line
- use a doubly linked list to maintain the status of the sweep line
- process points from the FIFO event queue, outputting and linking contour edges as we go.

PERFORMANCE

$O(n \log n + k)$ TIME
$O(n+k)$ SPACE
} Chazelle–Edelsbrunner

$O(n+K)$ TIME
$O(n+k)$ SPACE
} TOPOLOGICAL SWEEP

$O(n \log n + k)$ TIME
$O(n+k)$ SPACE

TIME COMPLEXITIES

ALG. I: $O((n+s)\log n + p\log((n+s)^2/p))$

ALG. II: $O((n+s)\log n + p)$

ALG. III: $O((n+s)\log n + p)$

GEN. ALG. I: $O((n+r)\log n + h)$

GEN. ALG. II: $O(n\log n + k)$

LINE SEG. INT → Bentley-Ottman 1979

LINE SEG. INT → Chazelle-Edelsbrunner 1988

S = # intersections involving slanted edges
r = # intersections of slanted edges with other slanted or horizontal edges
h = # intersections involving vertical edges
k = # intersections of any type
p = # contour edges.

SPACE COMPLEXITIES

ALG I: $O(n+p+s)$

ALG II: $O(n+p+s)$

ALG III: $O((n+s)\log n + p)$

GEN. ALG. I: $O(n+p)$

GEN ALG II: $O(n+k)$

- post-sorting of $O(p)$ edges
- no processing of the p edges after computation
- $O(k)$ space is inherent to the algorithm.

QUESTIONS

IS $s << k$?

IS $p << k$?

Which algorithm is best in practice?

(On our small samples, ALG I outperforms ALGS II & III.)