1. Write algorithm to check if 2 lines intersect -
2. Write an Algorithm to check if a point is within polygon
3. Write an algorithm to check if 2 polygons intersect each other
4. Implement the APPROX-TSP-TOUR algorithm for the traveling-salesman problem with the triangle inequality. (Page 1112)

**33. Computational Geometry**

Sweeping - O(nlgn) time

Graham’s scan - O(nlogn)

Jarvis’ march O(nh), h number of vertices of convex hull, n number of line segment

Closest Pair of Points - O(nlogn)

**33.1 Line-Segment Properties**

P3 is any point on the line passing through point 1 and point 2

Given two distinct points p1 and p2, the line segment p1p2 is the set of convex combinations of p1 and p2.

1. Given two directed segments is first segment counter clockwise or clockwise

If directed segment P0P1 is closer to P0P2 in clockwise or counter clockwise, with respect 0,0

P1-P0 = P’1 = (X1-X0, Y1-Y0)

P2-P0 = P’2 = (X2-X0, Y2-Y0)

(P1-P0) X (P2-P0) = (X1-X0)( Y2-Y0) - (X2-X0)(Y1-Y0)

(P1-P0) X (P2-P0)= positive, P0P1 is clockwise from P0P2

1. Given P0P1 and P1P2, left turn at point P1?

(P2-P0)X(P1-P0) = negative, P0P2 is counter clockwise with respect to P0P1 and left turn

1. Line segments p1p2 and p3p4 intersect?

p1p2 straddles a line if point p1 lies on one side of the line and point p2 lies on the other side

Boundary case: if p1 or p2 lies directly on the line

Two line segments intersect if and only if either (or both) of the following conditions holds:

1. Each segment straddles the line containing the other.

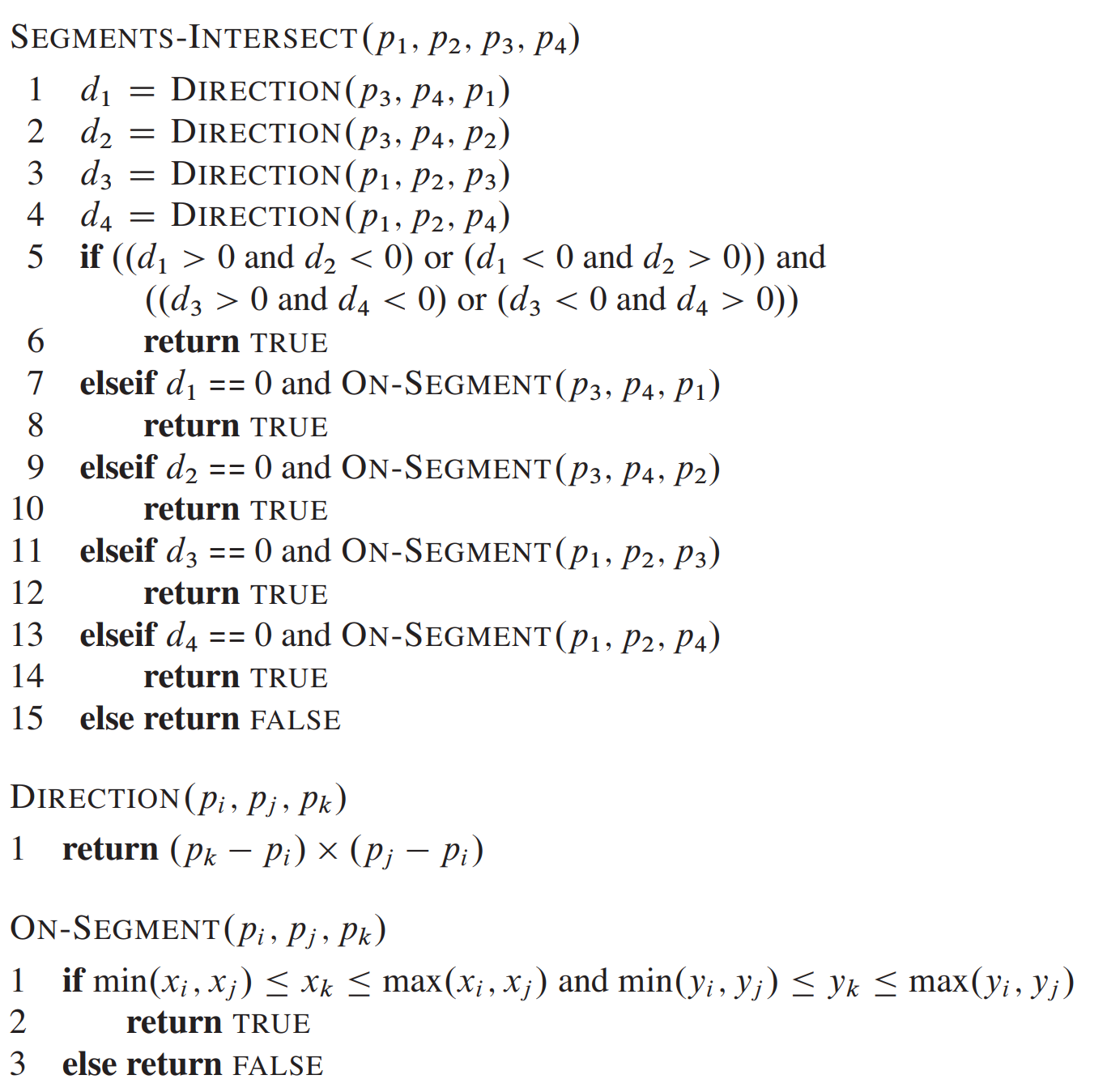
2. An endpoint of one segment lies on the other segment. (This condition comes from the boundary case.)

SEGMENTS-INTERSECT

DIRECTION, which computes relative orientations using the cross product method above

ON-SEGMENT, which determines whether a point known to be colinear with a segment lies on that segment.

-All O(1) time determination



d1=direction of p1 in relation to segment P3P4

d2= direction of p2 in relationt to segment P3P4

d3=direction of p3 in relation to segment P1P2

d4=direction of p4 in relation to segment P1P2

If d1 and d2 opposite signs and d3 and d4 opposite signs

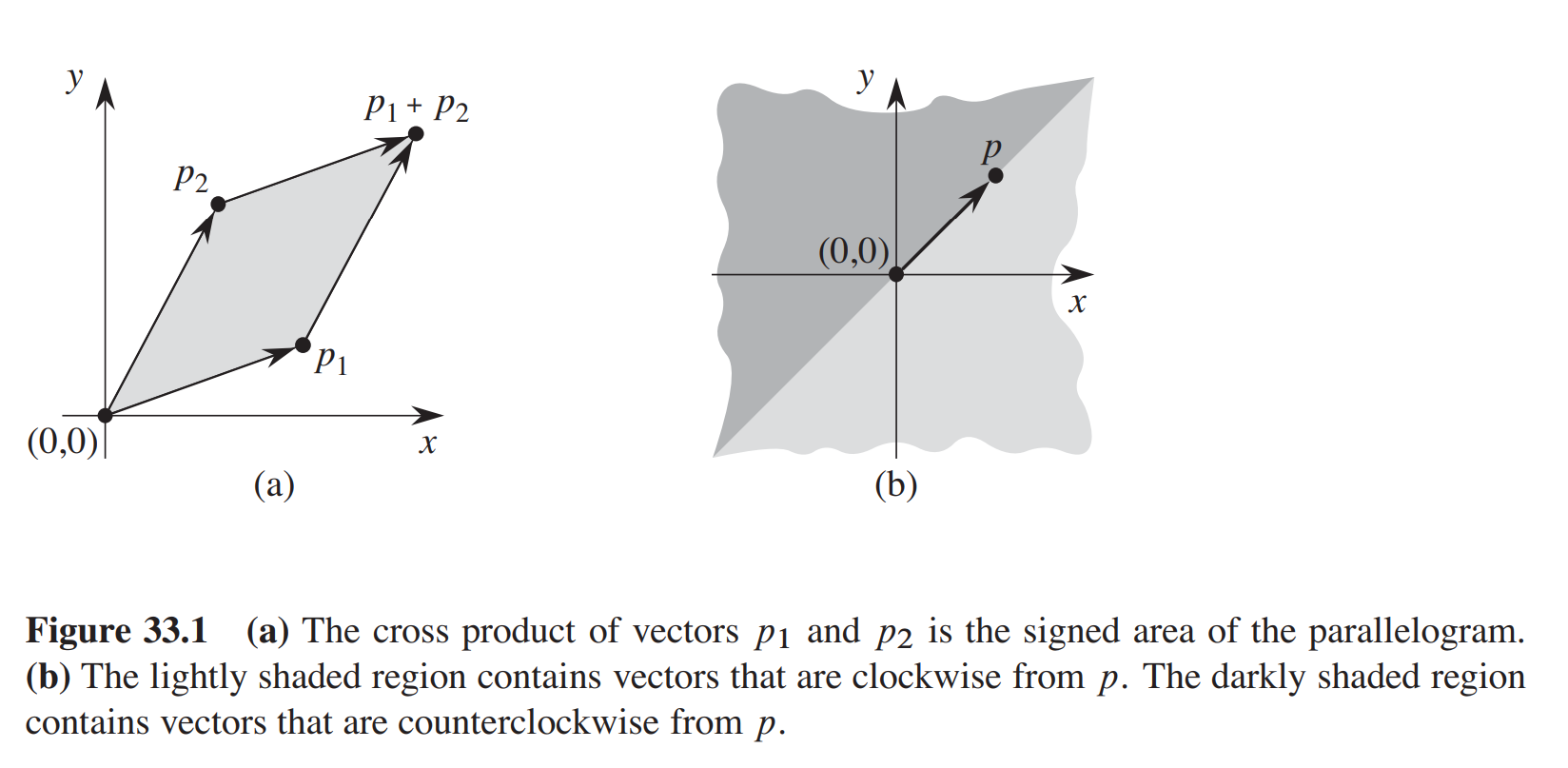
-return true

If d1==0, on-segment

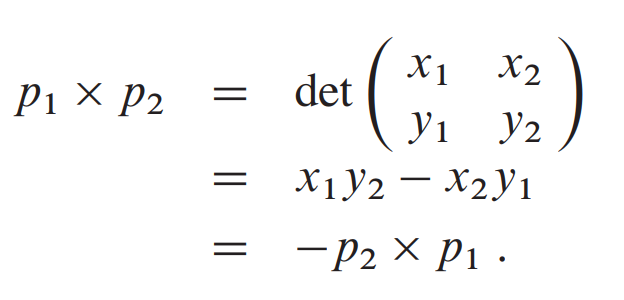
if (check p1 on p3p4),

-above= if x of segment <= X of point <=Max of x of segment and Y is inbetween min of y segment and max of y segment

-point on line



cross product p1Xp2 as the signed area of the parallelogram formed by the points 0,0, p1, p2, and P1+P2=(x1+x2, y1+y2)



Clock-wise or counterclockwise

P1 X P2 = positive, p1 clockwise from p2 with respect to 0,0

P1 X P2 = 0 colinear, pointing in same or opposite directions

1. Write algorithm to check if 2 lines intersect
2. Write an Algorithm to check if a point is within polygon
3. Write an algorithm to check if 2 polygons intersect each other
4. Implement the APPROX-TSP-TOUR algorithm for the traveling-salesman problem with the triangle inequality. (Page 1112)

Test Case:

Input: G={V,E} is a fully connected graph, while V={a: (1,3), b:(1,1),c: (0,0), d: (3,3), e: (4,2), f: (3,1), g: (5,1)}

Output: {a, d, e, g, f, h, c, b}

—-------------------------------------------------------------------------------------------------------------------

**33.2: Determining whether any pair of segments intersects with Sweeping**

O(n lg n)time, n is number of segments

-does not show what intersections exist, O(N^2) is time complexity to find all intersections

Left to right sweep with intersection check every time it encounters end point

-Assumptions: no vertical segments, no three lines intersect at 1 point

+order the segments that intersect a vertical sweep line according to the y-coordinates of the points of intersection.

+s1 and s2 segmens comparable at x if sweep with x-coordinate intersects both lines

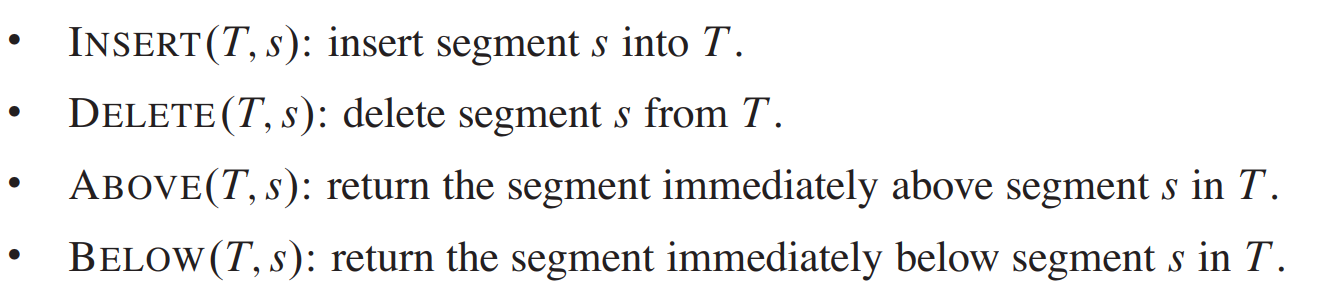
+s1 is higher than s2 if y value of s1 is higher than s2 at same x value

+ enters ordering when encountered from left to right, left endpoint (enter) and right endpoint(leave)

*Moving the sweep line*

+sweep-line status: gives the relationships among the objects that the sweep line intersects.

-sweep-line status is a total preorder T

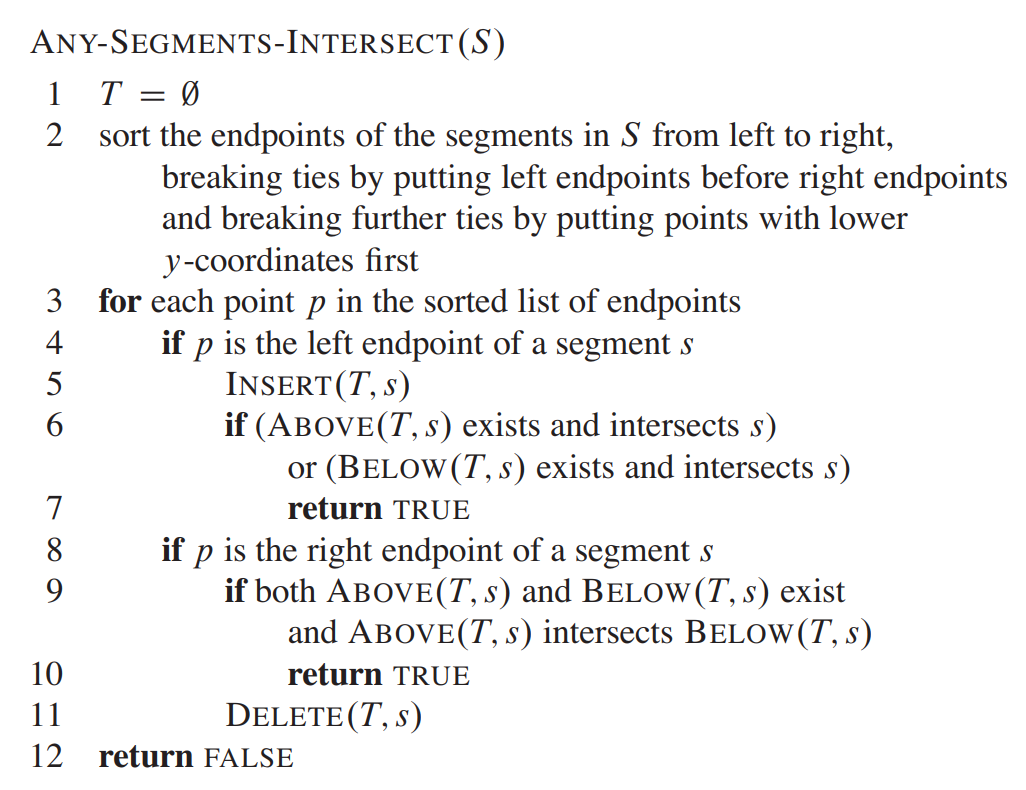


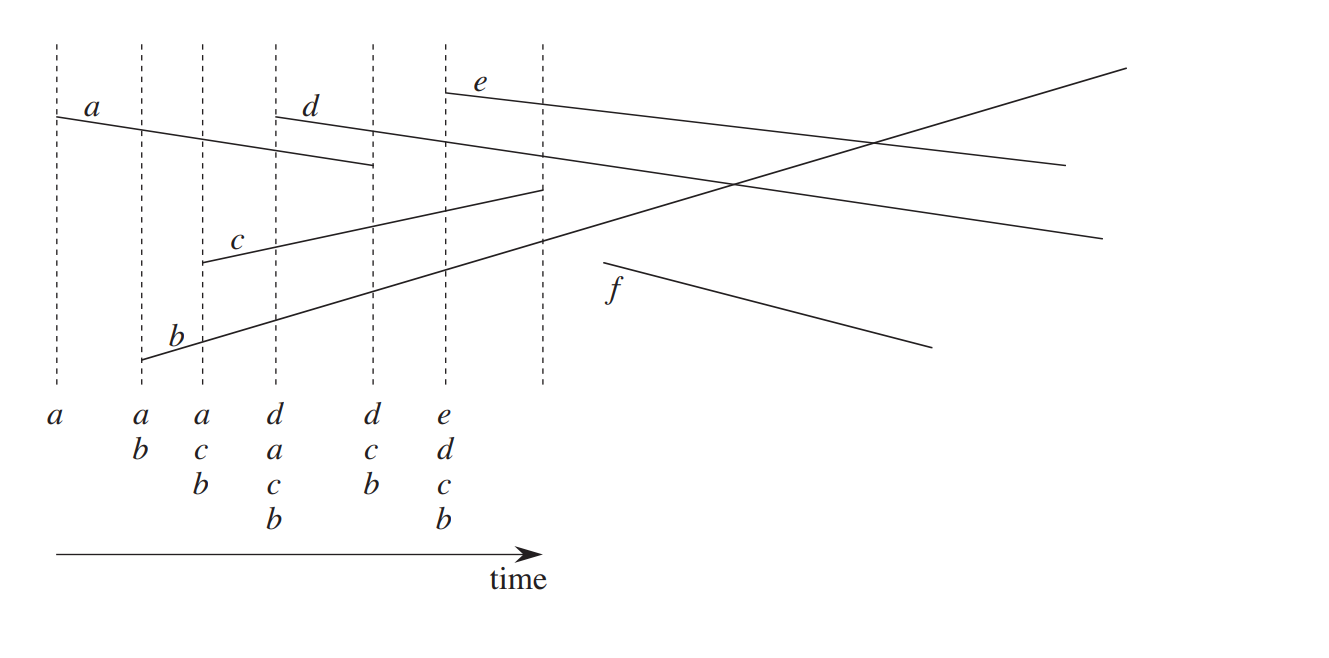
sINSERT, DELETE, ABOVE, and BELOW in O(lg n)time using red-black trees.

+event-point schedule is a sequence of event points (end-points), which we order from left to right according to their x-coordinates. As the sweep progresses from left to right, whenever *the sweep line reaches the x-coordinate of an event point,* the sweep halts, processes the event point, and then resumes. Changes to the sweep-line status occur only at event points.

+ covertical endpoints: same x-coordinate, left (beginning) endpoints before right end-points, lower y before higher y -coordinates

+whenever two segments consecutive in total preorder, check if intersect





—-----------------------------------------------------------------------------------------------------------------

**33.3: Finding the convex hull**

CH(Q)= convex hull of set of Q points

The convex hull is then the shape formed by a tight rubber band that surrounds all the nails

=is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior

2 methods for output of the vertices of the convex hull in counterclockwise order.

Both use…

Rotational Sweep: processing vertices in the order of the polar angles they form with a reference vertex

-can also user incremental method O(nlogn), divide and conquer O(nlogn), and prune and search method O(nlogh)

farthestpair problem:O(nlogn) we are given a set of n points in the plane and wish to find the two points whose distance from each other is maximum.

n= number of points

h = number of vertices of the convex hull.

Graham’s scan O(nlogn) -

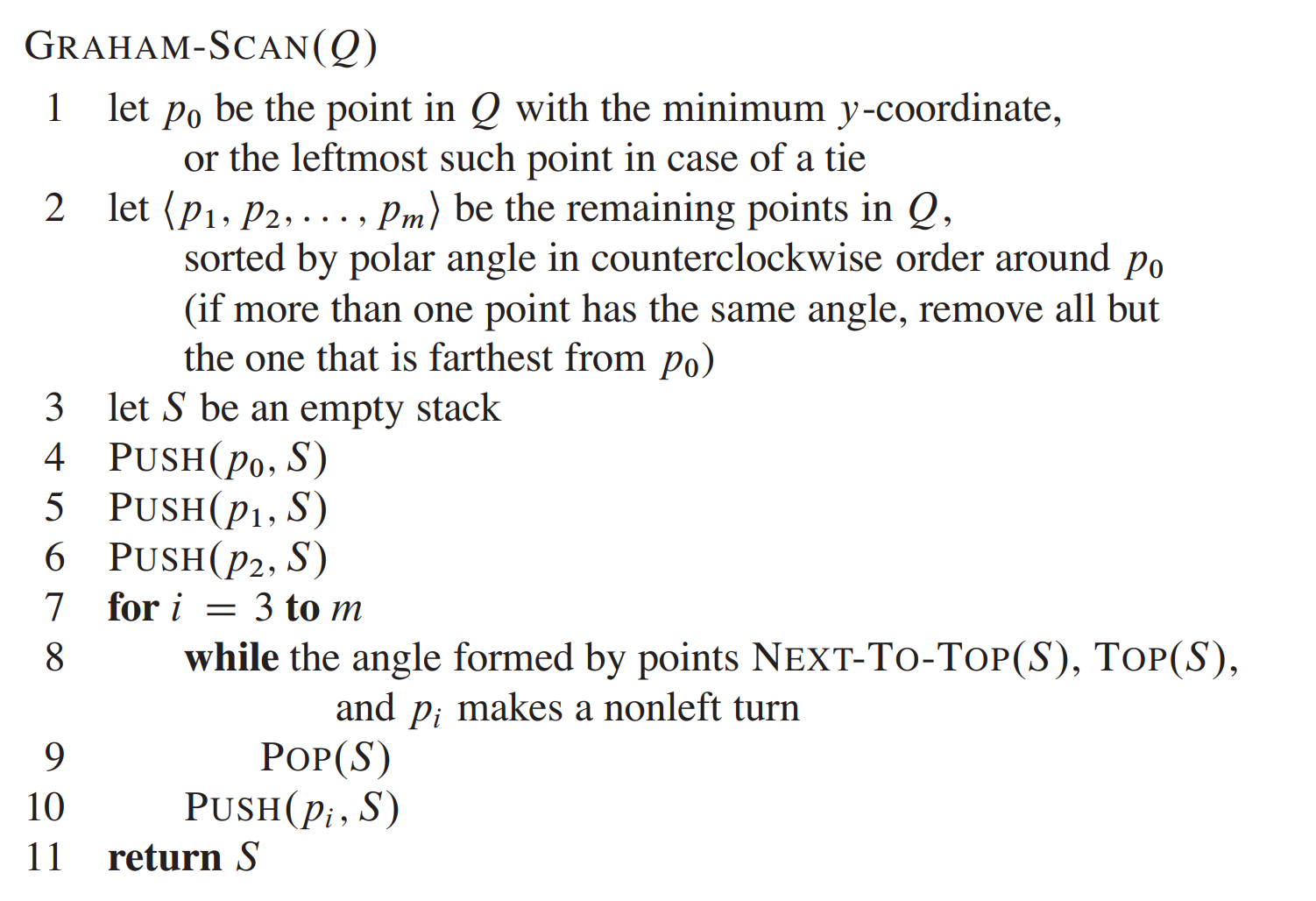
Stack S of candidate points

Q input set pushes point onto stack and pops from stack point not a vertex

-output: stack S contains exactly vertices of CH(Q) from bottom to top counterclockwise

TOP(S) which returns the point on top of stack S without changing S

NEXT-TO-TOP(S) which returns the point one entry below the top of stack S without changing S



1. Find most bottom, most left point
2. Find points from right to left
3. 4-6 add first point, furthest point on right and next point from righ to left
4. Stack =P2, P1, P0, P0 on bottom
5. Check next point from right to left, if p2 and p1 make a straight segment,
   1. left turn is anything left of the straight line create by p1 and p2
      1. If left turn add p3 to stack
      2. If not left turn, take out top and put in p3
         1. Replacing not most right point

Running Time analyis:

1. O(n) time to find p0
2. O(nlogn) to sort polar angles
3. 3-4 O(1) time
4. While O(n)

Jarvis’s march O(nh)

Packwrapping: asymptotically faster than Graham’s scan.

-the next vertex p1 in the convex hull has the smallest polar angle with respect to p0

- point farthest from p0

-Similarly, p2 has the smallest polar angle with respect to p1, and so on.

-highest vertex pk = right chain

-left chain, we start at pk and choose pk as the point with the smallest polar angle with respect to pk, but from the negative x-axis

-We continue on, forming the left chain by taking polar angles from the negative x-axis, until we come back to our original vertex p0.

-without separately constructing the right and left chains.

-ykeep track of the angle of the last convex-hull side chosen

-require the sequence of angles of hull sides to be strictly increasing (in the range of 0 to 2 radians)

-Time Complexity: Each comparison between polar angles takes O(1) time, using the techniques of Section 33.1. As Section 9.1 shows, we can compute the minimum of n values in O(n)time if each comparison takes O.1/ time. Thus, Jarvis’s march takes O(nh) time.

—-----------------------------------------------------------------------------------------------------------------

**33.4 Finding closest Pair of Points**

Distance between two points = Sqrt((x1-x2)^2+(y1-y2)^2)

Brute force: O(N^2)

Divide and Conquer: O(nlgn)

1st check is more than 3 points, if not just brute force

*Divide: divide map by one vertical line where Left side is PL and Right Side PR*

*-sort each side by increasing x value*

*-do same for y*

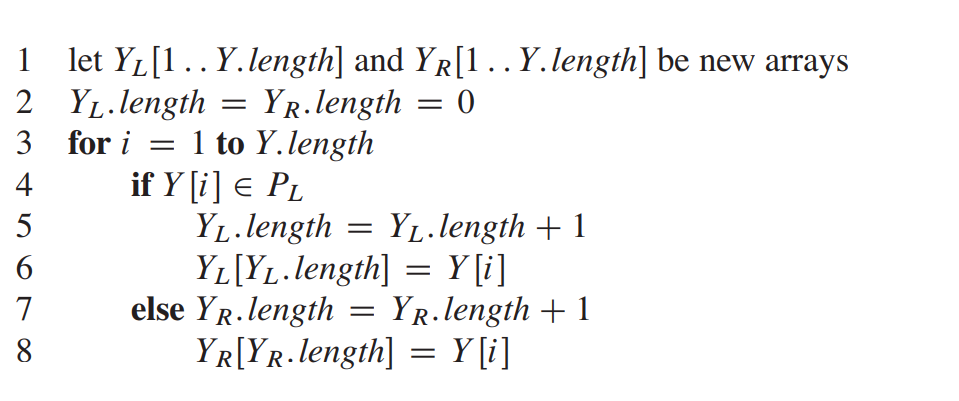
*=> four arrays*

*Conquer: recursive call for both sides for closes pairs of points in PL and PR*

*=> return SL= closest pair of distance in PL, SR = closest pair of distance in PR*

*Combine: closets pair is either 1. Pair within PL or PR or One point in PL and the other in PR*

1. *Create array Y’ with all points not in 2s-wide vertical strip removed* 
   1. *Sort array by y coordiante*
2. *For each point, see if you can find pair of points less than s minimum distance*
3. *If it exists, return pair and distance, otherwise use minimum of both recursive results*

**

O(nlogn)

Time Complexity: The only remaining question is how to get the points sorted in the first place. We presort them; that is, we sort them once and for all before the first recursive call. We pass these sorted arrays into the first recursive call, and from there we whittle them down through the recursive calls as necessary. Presorting adds an additional O.n lg n/ term to the running time, but now each step of the recursion takes linear time exclusive of the recursive calls. Thus, if we let T .n/ be the running time of each recursive step and T 0 .n/ be the running time of the entire algorithm, we get T 0 .n/ D T .n/ C O.n lg n/ and