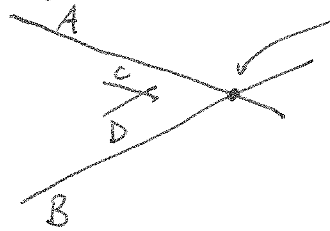
CS 410/584 Spring 2011 HW 5 Key

33.2-5

Left-to-right issue

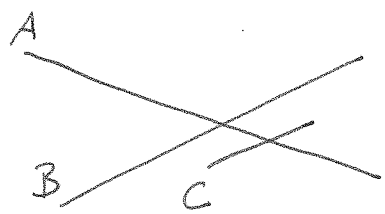


Finds the A-B intersection first, which is to the right of the C-D intersection.

All intersections issue

a) Counting: The for-loop in Any-Segments-Intersect executes 2n times for n segments. Each iteration does either two intersection tests (left endpoint) or one such test (right endpoint). So 3n tests in all, ~~thus~~ at most 3n intersections can be reported. But there can be up to $(n^2-n)/2$ intersections of n segments. So some must missed.

Particular example:



B always remains below A on the sweep line, so A and C never "see" each other to get tested for intersection.

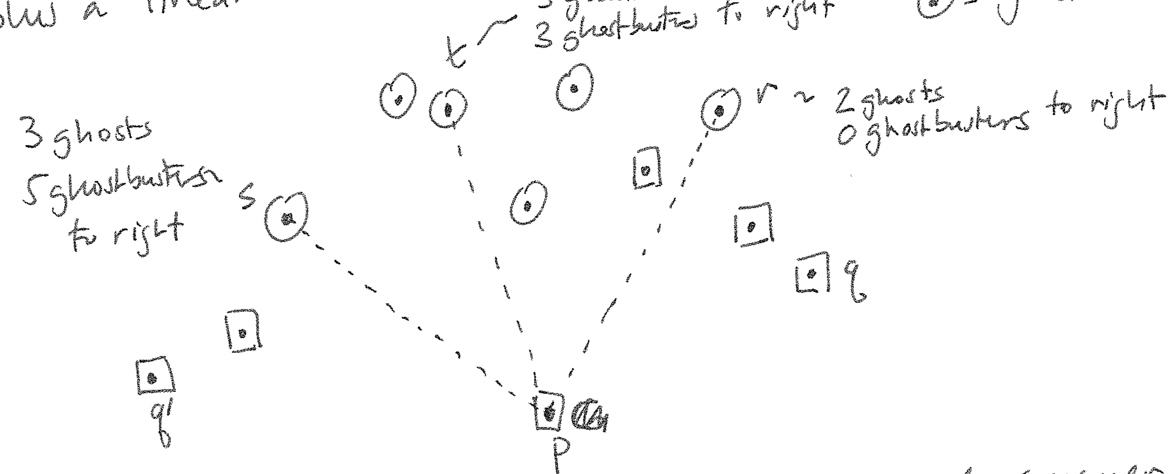
33-3

a) Start with a point p on the convex hull (such as min y point). Assume p is a ghost. (The ghostbuster case works the same.) Sort the other points by increasing polar angle with p. Let the first point in this order be q.

If q is a ghostbuster, we have the property we want: There are n−1 ghosts and n−1 ghostbusters to the left of the p−q line. (Same works if last point in order is a GB.)

If q is a ghost, too, then consider the line from p to r, where r is the first ghostbuster in order. There are 0 ghostbusters and $c>0$ ghosts to the right of the p-r line (since at least q is included). So more ghosts than ghostbusters to the right. The line from p to s, where s is the last ghostbuster in polar-angle order, has n-1 ghostbusters ~~to the right~~ and n-1-d ghosts to the right (since the last point in order is a ghost). So more ghostbusters than ghosts. So there is some point t in between r and s where there are the same # of ghosts and ghostbusters to the right. This process is $O(n \lg n)$, since it is a polar-angle sort, plus a linear scan.

$\boxed{\cdot}$ = ghost
$\bigcirc$ = ghostbusters



b) Use the previous result for divide-and-conquer. Find the partitioning line, and have the ghost shoot at the ghostbuster. Then solve left and right recursively. In the worst case, we have to repeat the process n times, because all the points land on one side of the line. So $O(n \cdot n \lg n) = O(n^2 \lg n)$ worst case complexity.

5A. RFFT(0, 1, 2, 3)

- $y[0] = \text{RFFT}(0, 2)$
  - $y[0] = \text{RFFT}(0) = 0$
  - $y[1] = \text{RFFT}(2) = 2$

  return $(0 + 2(1),\ 0 - 2(1)) = (2, -2)$
- $y[1] = \text{RFFT}(1, 3)$
  - $y[0] = \text{RFFT}(1) = 1$
  - $y[1] = \text{RFFT}(3) = 3$

  return $(1 + 3(1)\cancel{3},\ 1 - 3(1)) = (4, -2)$

return $(2+4,\ -2+{-2i},\ 2-4,\ -2-{-2i}) = (6, -2-2i, -2, -2+2i)$

5B. We can merge the lists for different letters in a pattern. We need to offset the positions in the list for a letter by the position of the letter in the pattern.

Suppose the pattern is 'test'

So we merge

t-list (no offset) = 1, 11, 14, 16, 19
e-list (-1 offset) = 11, 16
s-list (-2 offset) = 2, 5, 11
t-list (-3 offset) = 8, 11, 13, 16

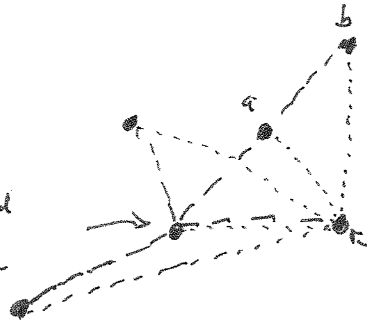
intersect-merge = 11

pattern appears at position 11.

5C. For each point, sort the other points by polar angle around that point. Go through the sorted list, and use X-product to see if the point plus any two consecutive points on the ~~so~~ list are in a row.

$O(n \lg n)$ for the polar-angle sorting. To run through n points as the sorting origin is $O(n^2 \lg n)$.

SC example:



try 2nd points a, b line up.

try first - no 2 points line up with it

5D. The Idea is to determine the Hamitonian ~~cy~~ path by deleting edges ~~fro~~ from G as long as they don't destroy the Hamitonian path. (More accurately, we remove an edge if there is some Ham. path that doesn't use it.)

Let ~~HP~~ HP be ~~a~~a polynomial-time algorithm that decides Ham-Path.

1. If HP(<G, v, w>) = false return("no cycle")
2. For each edge e in G
   - let G' be e with ~~G~~ e removed
   - if HP(<G', v, w>) = true, then G ⟸ G'
3. The edges in G form a path from v to w. The order of the nodes can be found, for example, with DFS.