Finding collinear points

The problem: Given a set of n points in the plane, determine if there exist three points that are collinear.

We'll assume that we can check whether any three given points are collinear in $O(1)$ time. We'll come back with details on how to do this next week.

Brute force

Algorithm 1 (brute force)

- for all distinct triplets of points p_i, p_j, p_k: if collinear return true
- (if you get here) return false

Questions:

- Argue that the algorithm is correct (can it miss any triplets?).
- What is the (worst-case) running time?
- How much space does it use?

Via sorting

Algorithm 2

- initialize array L = empty
- for all distinct pairs of points p_i, p_j
  - compute their line equation (slope, intercept) and add it to an array L
- sort array L by (slope, intercept)
- traverse L and if you find any 3 consecutive identical (s,i) $\rightarrow$ collinear

Questions:

- Argue that the algorithm is correct.

- What is the (worst-case) running time?

- How much space does it use?

With a binary search tree

Algorithm 3

- initialize BBST = empty
- for all distinct pairs of points p_i, p_j
  - compute their line equation (s, i)
  - insert (s,i) in BBST; if when inserting you find that (s,i) is already in the tree, you got three collinear points and return true
- (if you ever get here) return false

Questions:

- Argue that the algorithm is correct.
- What is the (worst-case) running time?
- How much space does it use?
- How does it compare to Algorithm 2?

With hashing

Algorithm 4

- initialize HashTable = empty
- for all distinct pairs of points p_i, p_j
  - compute their line equation (s, i)
  - insert (s,i) in HashTable; if when inserting you find that (s,i) is already in the HT, you got three collinear points and return true
- (if you ever get here) return false

Questions:

- Argue that the algorithm is correc.
- What is the (worst-case) running time?
- How much space does it use?
- Hoes does it compare to Algorithm 3?
- Under what assumption on the input is Algorithm 4 faster than Algorithm 3?

A different way to sort

Algorithm 5

- for every point p_i
  - set array L = empty
  - for every point p_j (with $p_j! = p_i$)
    - compute slope of p_j wrt to p_i and add it to array L
  - sort L
  - traverse L and if you find two consecutive points that have same slope, they are collinear with p_i so return true
- (if you get here) return false

Questions:

- Argue that the algorithm is correct (can it miss any triplets?).
- What is the (worst-case) running time?
- How much space does it use?