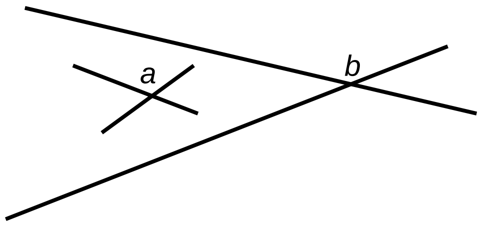
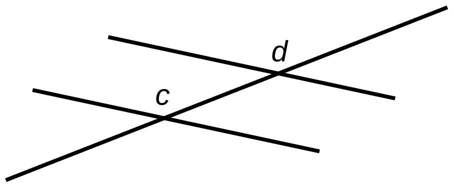
Lecture 3 exercise solutions

CLRS3 33.2-3.

The answer is no to both questions. On the left, there is an example where the leftmost intersection, *a*, will be discovered second, after *b*. On the right, there is an example, where intersection *d* will not be reported at all.

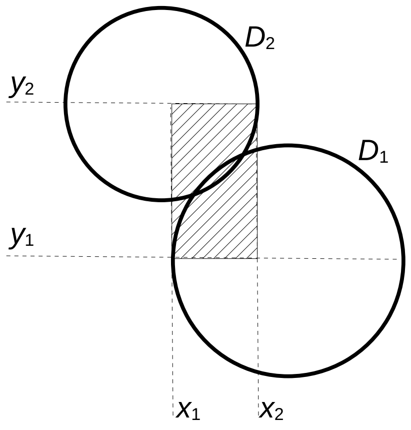




CLRS3 33.2-6.

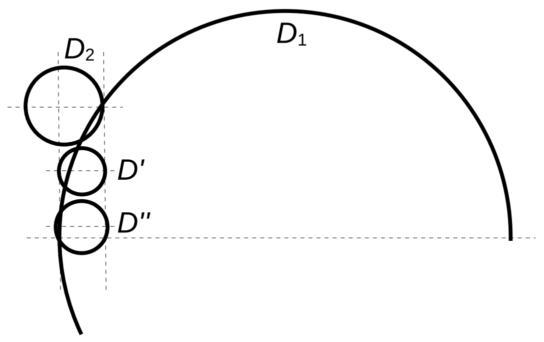
It turns out the ANY-SEGMENTS-INTERSECT from the textbook works also for the disks if the following details are taken care of (we assume that a disk is represented by (x, y, r) – the coordinates of the center and the radius)

1. The two "endpoints" of a disk are its leftmost point $(x - r, y)$ and its rightmost point $(x + r, y)$.
2. The total order of the sweep-line status set T (maintained by a red-black tree) is based on the y coordinate of the centers of disks.
3. Intersection check is simple. Two disks intersect if the distance between their centers is smaller than the sum of their radii. Note that we do not need the absolute value of the distance (which involves taking the square root). Instead, we can compare the square of the distance with the square of the sum of their radii.



Why is this algorithm correct? First, it is easy to see that the algorithm checks all pairs of disks such that they are neighbors in the vertical order of T at some position of the sweep-line. Thus, to prove the correctness of the algorithm we have to show that, if there are some intersecting pairs of disks, then the disks of at least one of such pairs will become neighbors in T at some sweep-line position. Consider any pair of intersecting disks D_1 and D_2 and let $[x_1, x_2]$ be the intersection of their x-intervals (see the figure). That is, $[x_1, x_2]$ is the interval during which both of the disks are in T. For them *not* to be neighbors in T during all of the interval $[x_1, x_2]$ (and thus avoid detection), there has to be another disk D, such that (i) the y coordinate of D's center is in $[y_1, y_2]$, where y_1 and y_2 are the y coordinates of the centers of D_1 and D_2 *and* (ii) the x-interval of D *contains* $[x_1, x_2]$. It is easy to prove that D will intersect with at least one of the two disks (to see this, it is enough to consider a horizontal centerline of D – it can not cross the hatched area in the figure without intersecting one of the disks, otherwise the two disks would not intersect each other). If there are several such Ds, either the one with the highest center will intersect D_2 (and be its neighbor in T) or the one with the lowest center will intersect D_1 (and be its neighbor in T). In both cases, the intersection will be detected by the algorithm. In the figure below, there are two such

Ds in-between D_1 and D_2 in *T*. The intersection between D_1 and D'' will be detected.



CLRS3 33.3-5

The idea is not to repeat the expensive firsts phase of Graham's scan – sorting according to polar angles with respect to the anchor point. Starting from the third point, we don't care about the points inside the hull and, if we maintain the points of the convex hull in the counterclockwise order starting from the bottommost point, this order is also the order according to the polar angles with respect to the bottommost point. When a new point comes we can in $\Theta(n)$ time insert it into the right place in this sorted order using cross-product comparisons. Then, again in $\Theta(n)$ time, redo the rotational sweep to build the convex hull.

The special case is when the point lower than the current bottommost point arrives. Then we change our anchor point to the topmost point – going counterclockwise along the convex hull, we can easily find the topmost point. Then, continuing counterclockwise along the hull and going all the way around the hull until we are back to the topmost point, we collect all the points of the hull in the order of the polar angle with respect to the topmost point. We insert a new point in this order and redo the rotational sweep to build the convex hull. We live with the new anchor point (the topmost point) until a point above it arrives, at which point we change the anchor point again to the bottommost point and so on.

Adding each point in this way costs $\Theta(n)$, thus the total running time is $\Theta(n^2)$.

CLRS3 33.1-3.

We can not directly reuse the first phase of the Graham's scan because p_0 is not guaranteed to be the lowest point. Thus, before comparing any two points using cross-products (as in the sorting of the Graham's scan), one has to check if both of them are above p_0, or both of them are bellow p_0, or one is above and the other is bellow. In the first and the second case their polar angles can be compared using cross-products. In the third case, the point above p_0 has a smaller polar angle than the point below p_0. With this comparison procedure, any efficient sorting algorithm can then be used.

CLRS3 33.1-4.

For each point:

1. Use this point as an origin point and sort the remaining points according to their polar angle using the solution to 33.1-3.
2. Go through the sorted order and, for each pair of neighbor points, check (using the cross product) if they, together with the origin poin,t form a colinear triple.

If there is a colinear triple in the dataset it will be definitely caught by this algorithm when, examining the lowest of the three points as an origin point, the other two points will have the same polar angle and will become neighbors in the sorted order (unless they belong to a larger set of colinear points, in which case some other two points from this set may become neighbors instead and be returned as forming a triple with the origin point).

In each iteration of the loop, step 1, the sorting, dominates ($\Theta(n \lg n)$), as step 2 takes $\Theta(n)$ time. The algorithm does n iterations, thus the total running time is $\Theta(n^2 \lg \text{n})$.