Tom St. John
Gowri Somanath
Feng Li

Problem 1

Suppose the line direction is from point p to q,

a. The determinant D is just the magnitude of the cross product of vector $\overrightarrow{pq}$ and $\overrightarrow{pr}$. And this cross product magnitude is equal to $|\overrightarrow{pq}||\overrightarrow{pr}| \sin\theta$, θ is the angle between these two vectors. So if the r lies left of the line, $\sin\theta > 0$, then D > 0; if the r lies on the line, D = 0; and if r lies on the right of the line, D < 0.
b. From (a) we know D = $|\overrightarrow{pq}||\overrightarrow{pr}| \sin\theta$, that is the unsigned area of the parallelogram having $\overrightarrow{pq}$ and $\overrightarrow{pr}$ as sides. So |D| is twice the surface of the triangle determined by p, q and r.
c. This method is very robust and only needs to compute the sign of the determinant. And it has little numerical problems.

Problem 2

Solution:

In order to use plane sweep algorithm to solve this problem, we need to change the representation of circle a little bit so that circle intersection computation can imitate the line segment intersection computation. Hence, we split each circle into two half circles, i.e. left half and right half. Then we use the same procedure of computing line segment intersection to compute circle intersection.

The algorithm is as follows:

Input: A set **S** of n circles

Output: The intersection points among the circles in **S**

```
Initialize:   a priority queue Q to contain all the upper points (center_x, center_y + r) and
                  lower points (center_x, center_y - r) of n circles
              An empty structure T
While Q is not empty do
    Let p is next event point, and delete p from Q
    If p is the upper point (center_x, center_y + r) of circle c (center_x, center_y)
        Add c to T
        Test c for intersections with its neighbors in T
            For each intersection that is found
                Output that intersection and add the intersection to Q
    If p is the lower point (center_x, center_y – r) of circle c (center_x, center_y)
        Delete c from T
        Test the prior neighbors of c for intersections
            For each intersection that is found
                Output that intersection and add the intersection to Q
    If p is the intersection point of half circle c1 and c2
        If the intersection is a tangent intersection
            continue;
        Exchange the place of c1 and c2
        Test c1 and c2 for intersections with their neighbors in T
        For each intersection that is found
                Output that intersection and add the intersection to Q
```

Notes:

1. Before computing the intersection points among circles, we first sort the upper points (center_x, center_y + radius) and lower points (center_x, center_y – radius) of all the circles along the Y direction, and save the result into a priority queue **Q**.
2. For the structure **T**, the half circle is represented as (center point, radius, left/right). Based on this representation, we can avoid the intersection computation between two half circles of the same circle.
3. When computing the intersection, we also have to check if this intersection point is a tangent point or not. Because if the intersection is a tangent intersection, there is no need to exchange the place of the two half circles. Since we have represent the half circle as (center point, radius, left/right), it is very trivial to check the tangency problem, including inner tangency and outer tangency.

Problem 3

Part 3.1

Since P[k-1] is known to be within the convex hull, the edge P[k]P[k-1] must cross over one of the edge segments d[bot]d[bot+1] or d[top-1]d[top] if P[k] is outside of the convex hull. Likewise, if the edge P[k]P[k-1] does not cross over either of the previously mentioned edge segments, P[k] must be within the convex hull.

Part 3.2

In order to test whether P[k]P[k-1] crosses the edge segments d[bot]d[bot+1] or d[top-1]d[top], you check whether P[k] is to the left of the two edge segments. If P[k] is to the left of both edge segments, then P[k]P[k-1] does not cross over either of the edge segments. If P[k] is to the right of either edge segment, then P[k]P[k-1] crosses over one of the edge segments.

Part 3.3

(1) Check if P[k] is to the left of both d[bot]d[bot+1] and d[top-1]d[top] edge segments
(2) If P[k] is to the right of either segment, remove the corresponding end vertex from the deque
(3) Repeat Step 2 until P[k] is to the left of both edge segments
(4) Add P[k] to both ends of the deque (d[top]=P[k], d[bot]=P[k])

Part 3.4

At most, there are: two insertions into the deque for each vertex (2 corresponding tests)
One removal from deque for each vertex (1 corresponding tests)

This results in 3*n insert/delete operations and 3*n comparisons, so running time is O(n).

Part 3.5

This algorithm computes the convex hull as it encounters the points in the set, rather than first locating the extreme points. Because of this, the ordering of the vertices is not important, and there is no need to sort them, resulting in a run time of O(n) instead of O(n*log n)