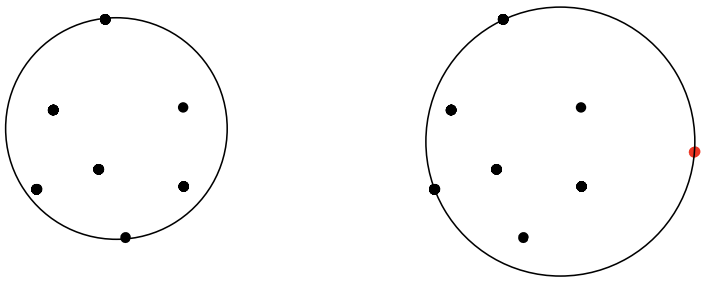
4. (20 pts) **Smallest Enclosing Circle**

In this problem we'll develop an algorithm for computing the *smallest enclosing circle* of a set P of $n \geq 2$ distinct points. This is the circle of minimum radius that contains the points of P. It is not hard to prove that the solution circle $C(P)$ is unique, and that $C(P)$ has two or more points of P on its boundary. In the case in which there are just two points of P on $C(P)$, the segment between those points is a diameter of the circle. If three points a, b, and c (and possibly others) are on the boundary of $C(P)$, then $C(P)$ is the unique circle through a, b, and c.



In the example on the left, only two points are on the smallest enclosing circle. On the right we've added a new point (colored red) to the set. The new point was outside of the previous smallest enclosing circle. The new solution has three points on the circle, including the new red point.

Here's a randomized incremental algorithm for computing the smallest enclosing circle of a list of n points:

SEC($[p_1, p_2, \ldots, p_n]$) = {
    Randomly permute the input points, so $[p_1, \ldots, p_n]$
    is a random permutation of the given points.

    Let C be the smallest circle enclosing p_1 and p_2.
    (This is just the circle for which p_1 and p_2 form a diameter.)

    for $i = 3$ to n do
        // at this point C is the smallest enclosing circle for $[p_1, \ldots, p_{i-1}]$
        if p_i is not in C then C = SEC1($[p_1, \ldots, p_i]$)
    done

    return C
}

Here we've made use of a function SEC1($[p_1, p_2, \ldots, p_i]$). This is an algorithm that computes the smallest enclosing circle of $[p_1, \ldots, p_i]$ *given the information that* p_i *is one of the points on the boundary of the smallest enclosing circle of* $[p_1, \ldots, p_i]$. For our purposes here, you need not worry about how to implement this function.

(continued on next page)

(a) Prove that in the context in which SEC1() is called, the smallest enclosing circle of $[p_1, p_2, \ldots, p_i]$ must have p_i on its boundary.

If p_i is not on the boundary of $C(\{p_1 \cdots p_i\})$ then $C(\{p_1 \cdots p_i\}) = C(\{p_1 \cdots p_{i-1}\})$. Since these are not equal, we know p_i must be on the boundary of $C(\{p_1 \cdots p_i\})$

(b) Assume we could implement SEC1($[p_1, p_2, \ldots, p_i]$) to run in expected $O(i)$ time[1]. Prove that the algorithm SEC($[p_1, p_2, \ldots, p_n]$) runs in expected $O(n)$ time. (Note that randomly permuting n points is $O(n)$ time, and testing if a given point is in a given circle is $O(1)$ time.)

Use backwards analysis. When we delete a random point, what's the probability that it causes the SEC to change? If there are four or more points on the circle boundary then this probability is 0. If there are three then the prob. is $\frac{3}{i}$. If there are two its $\frac{2}{i}$. So in all cases it's at most $\frac{3}{i}$. Therefore the expected cost of removing p_i is at most $\frac{3}{i} \cdot O(i) = O(1)$.

[1]To give a complete expected linear-time algorithm, SEC1() can analogously be expressed in terms SEC2() which is told *two* of the list of input points are on the circle boundary. This, finally, is easy to implement in $O(i)$ time.