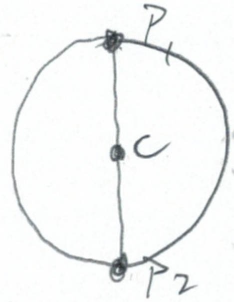
Smallest Enclosing Circle of $n \geq 2$ Points.

Ground Rules (Same as for closest Pair)

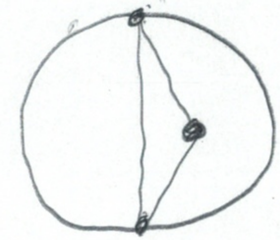
Simple Cases: $n = 2$

$$c = \frac{1}{2}(P_1 + P_2)$$



$$r = \frac{1}{2} |P_1 - P_2|$$

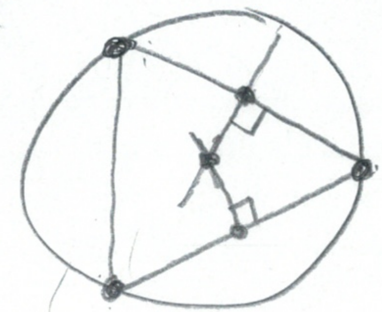
$n = 3$ case 1: Obtuse triangle



case 2: acute triangle

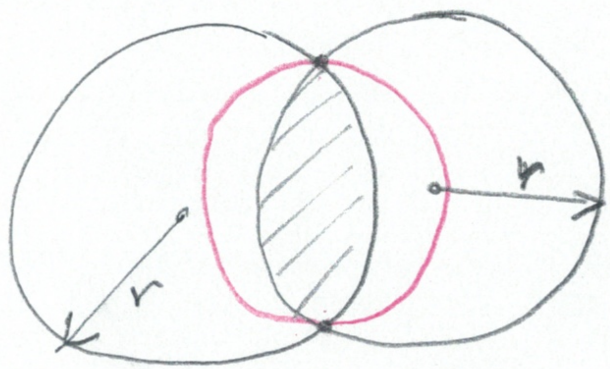
Compute Center

find radius.



Theorem 1: The $SEC(P_1 \cdots P_n)$ is unique

Proof: By contradiction. Suppose there are two distinct SECs.



there exists a smaller circle containing the intersection ∎

Theorem 2: The $SEC(P_1 \cdots P_n)$ is either the one whose diameter is some pair $(P_i P_j)$ or the cocircle of three points $(P_i P_j P_k)$ that form an acute triangle

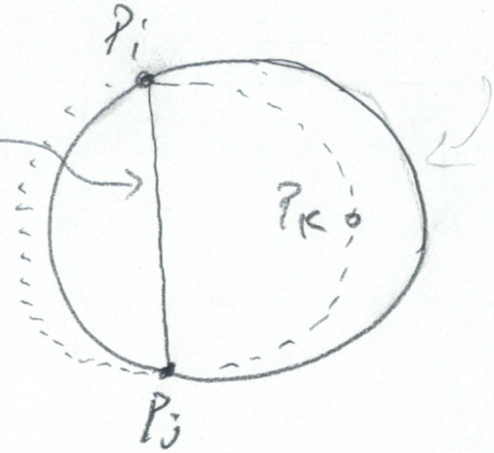
Proof: Consider an enclosing circle that touches no points of $P_1 \cdots P_n$

$\Rightarrow$ move it till it touches one P_i

$\Rightarrow$ now shrink it (keeping P_i) till it touches two $P_i P_j$

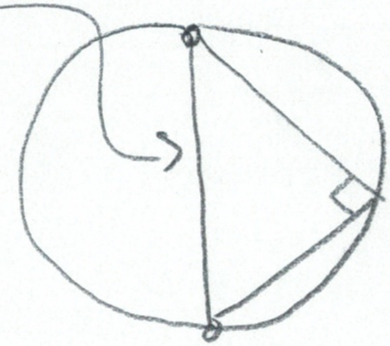
if $(P_i P_j)$ is a diameter we're done.

otherwise keep shrinking as shown on next page not a diameter



shrink the bigger half as shown until (1) $P_i P_j$ is a diameter or (2) we hit a point P_k the triangle $(P_i P_j P_k)$ is obtuse

Diameter



In either case, no smaller circle can contain the points

■

Corollary 3: The radius of the SEC($P_1 \cdots P_n$) is the maximum of $\frac{Diam(P_iP_j)}{2}$

and Radius($P_iP_jP_k$) where ($P_iP_jP_k$) is an acute triangle

These give an $O(n^3)$ algorithm to compute SEC($P_1, \cdots, P_n$):

Test all possibilities of Corollary 3.

The one with the largest radius must be the smallest encl. circle.

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

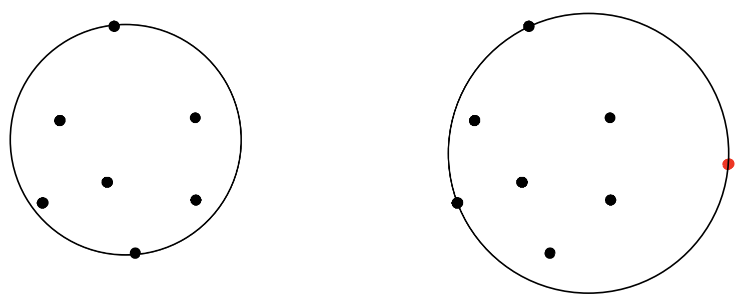
```
SEC([p_1, p_2, ..., p_n]) = {
    Randomly permute the input points, so [p_1, ..., p_n]
    is a random permutation of the given points.

    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, ..., p_{i-1}]
        if p_i is not in C then C <- SEC1([p_1, ..., p_{i-1}], p_i)
    done

    return C
}
```

The figure below shows what happens when a point p_i, shown in red, is added to the set on the left, and is not in the circle C. In this case the smallest enclosing circle for $[p_1, \ldots, p_i]$ must pass through the point p_i.



At this point in the algorithm, it calls $\text{SEC1}([p_1, p_2, \ldots, p_{i-1}], p_i)$, which computes the smallest enclosing circle of $[p_1, \ldots, p_{i-1}]$ *given the information that* p_i *is one of the points on the boundary of the smallest enclosing circle of* $[p_1, \ldots, p_i]$. The expected time of this call to SEC1 is $O(i)$ as shown on the next page.

We analyze the running time of SEC() using backward analysis. Imagine we have the circle on the right above, and we pick a random point to delete. The probability that the circle changes is at most $3/i$. In this case we have to call SEC1(), which has an expected running time of $O(i)$. In the other case the cost is $O(1)$. Thus the expected time to go once through the loop in SEC() is just $O(1)$. So the algorithm is expected $O(n)$.

```
SEC1([p_1, p_2, ..., p_n], q_1) = {
    We know that the point q is on the SEC containing p_1, ... p_n, q_1.

    Randomly permute the input points, so [p_1, ..., p_n]
    is a random permutation of the given points.

    Let C be the smallest circle enclosing p_1 and q_1.

    for i = 2 to n do
        // at this point C is the smallest enclosing circle for [p_1, ..., p_{i-1}, q_1]
        if p_i is not in C then C ← SEC2([p_1, ..., p_{i-1}], p_i, q_1)
    done

    return C
}
```

The argument that SEC1() runs in expected $O(n)$ time is very similar. Now the probability, in the backward analysis, that the deletion of a point causes a call to SEC2() is at most $2/i$. And again, assuming that SEC2() with i points is $O(i)$ time gives the proof that SEC1() is $O(n)$ time expected.

```
SEC2([p_1, p_2, ..., p_n], q_1, q_2) = {
    We know that the point q_1 and q_2 are on the SEC containing p_1, ... p_n, q_1, q_2.
    Let C be the smallest circle enclosing q_1 and q_2.

    for i = 1 to n do
        // at this point C is the smallest enclosing circle for [p_1, ..., p_{i-1}, q_1, q_2]
        if p_i is not in C then C ← Circle through points (p_i, q_1, q_2)
    done

    return C
}
```

Finally, it's very easy to see that each iteration of the loop in SEC2() is $O(1)$ time in the worst, case so the algorithm is $O(n)$.