CSCI 432 Handout: Welzl's Smallest Enclosing Circle

Name: _____

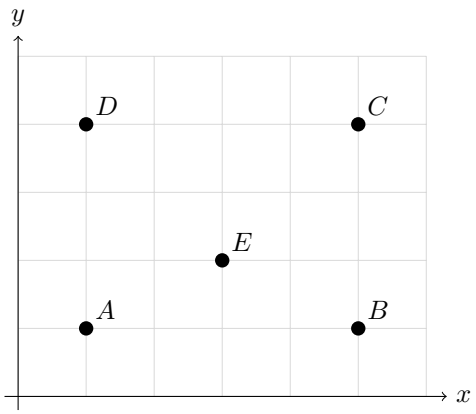
25 April 2026

Motivation

Given a set of points in the plane, the **smallest enclosing circle** (SEC) is the circle of minimum radius that contains every point. Think of it as drawing the tightest possible fence around a group of landmarks on a map.

A key structural fact makes this tractable: the SEC of any finite point set is **unique**, and is always "pinned" by at most **3** points sitting exactly on its boundary. These boundary points are called the *basis*. Welzl's algorithm exploits this to find the SEC in expected $O(n)$ time.

1. Below are five points. Sketch a circle that you think is the smallest one enclosing all five. Then circle which of the five points lie *on the boundary* of your circle (the basis). Does your basis have 1, 2, or 3 points?



Answer

2. A circle in the plane is uniquely determined by at most 3 points on its boundary. Fill in the table below.

Basis size $\lvert R\rvert$	**What circle does it determine?**	**How do you compute the centre?**
1		
2		
3		

3. **Key Lemma.** Suppose we already know the SEC of $P \setminus \{p\}$ (everyone except p). If p is *not* inside that circle, what must be true about p in the SEC of the full set P? Why does that make sense geometrically?

   Answer

The Algorithm

The algorithm is recursive. We pass two sets to each call:

- P — the points not yet processed.
- R — the points *forced* to be on the boundary ($|R| \leq 3$ always).

The goal of BoundaryMiniDisk(P, R) is to return the SEC of $P \cup R$ with every point of R on its boundary. The initial call is MiniDisk(P).

Algorithm MiniDisk(P)

Input: Point set P
Output: Smallest enclosing disk D of P

```
0: if P = ∅ then
0:    D ← ∅
0: else
0:    p ← Rand(P)
0:    D ← MiniDisk(P − {p})
0:    if p ∉ D then
0:       D ← BoundaryMiniDisk(P − {p}, p)
0:    end if
0: end if
0: return D =0
```

Algorithm BoundaryMiniDisk(P, R)

Input: Point sets P and R
Output: Smallest enclosing disk D of P with the points in R on its boundary

```
0: if P = ∅ or |R| = 3 then
0:    D ← BoundaryMiniDisk(∅, R)
0: else
0:    p ← Rand(P)
0:    D ← BoundaryMiniDisk(P − {p}, R)
0:    if D is defined and p ∉ D then
0:       D ← BoundaryMiniDisk(P − {p}, R ∪ {p})
0:    end if
0: end if
0: return D =0
```

1. In plain English, describe what each of the two recursive calls after the initial if statement in BoundaryMiniDisk(P, R) is "trying to do." When does the second call happen, and what does it mean when it does?

   Answer

2. Why is it safe to stop when $|R| = 3$ even if P is still non-empty?

   Answer

3. TRIVIAL(R) builds the circle directly from the boundary points in R. Describe what it returns for each value of $|R|$. (You filled in the geometry in Q2 above)

   Answer

4. **Trace the algorithm** on the five points from the first page, using the set $[A, B, C, D, E]$. For each call, write down P, R, the chosen point p, and whether p ends up inside the current circle. Continue until you reach a base case, then unwind back to the top.

   Hint: Start by calling MINIDISK($\{A, B, C, D, E\}$) *and peel off one point at a time.*

   Answer

Call #	P	R	**pick** p	p **inside circle? Action taken**
1				
2				
3				
4				
5				
6				
(unwind)				
(unwind)				
(unwind)				

Final circle: center _____, radius _____, basis _____

Complexity

1. The algorithm makes at most *two* recursive calls at each level. Suppose (worst case) the second recursive call always fires. Write the recurrence for $W(n)$ and solve it. What does this tell you about the importance of randomization?

   Answer

2. **Backwards analysis.** Imagine the random permutation has already been decided and we reveal it from *last* to *first*. Point p_i (the i-th in the permutation) triggers the second recursive call only if it is a basis point of $\text{SEC}(\{p_1, \ldots, p_i\})$.

   (a) How many basis points does $\text{SEC}(\{p_1, \ldots, p_i\})$ have at most?
   Answer

   (b) Since the permutation is random, what is the probability that p_i is one of those basis points?
   Answer

3. Using your answer above, write the recurrence for $T(n)$, the *expected* number of operations.

$$T(n) \;=\; T(n-1) \;+\; O(1) \;+\; \underbrace{\qquad\qquad}_{\text{expected cost of 2nd call}}$$

4. Give the space complexity of the algorithm and briefly justify your answer.

   Answer