Generate Random Point in a Circle - Documentation

1. Problem Statement

Design a class Solution that generates a random point uniformly inside a circle, including points on the boundary.

Class Signature:

```
class Solution:

    def __init__(self, radius: float, x_center: float, y_center: float):
        ...

    def randPoint(self) -> List[float]:
        ...
```

Input:

- A float radius, and two floats x_center, y_center representing the circle.
- Up to 30,000 calls to randPoint().

Output:

- Each call to randPoint() returns a list [x, y] that lies within the circle.

2. Intuition

To generate points uniformly inside a circle:

- Using a square and checking if the point lies within the circle is inefficient.
- Instead, sample angle and radius in polar coordinates, then convert to Cartesian.

3. Key Observations

- Random angle $\theta \in [0, 2\pi)$
- Radius must be scaled using sqrt(random) for uniform area distribution
- Convert polar to Cartesian:
    - x = x_center + r * cos(θ)
    - y = y_center + r * sin(θ)

4. Approach

1. Initialization:
    - Store radius, x_center, and y_center.
2. Generating Random Point:
    - Generate random angle $\in [0, 2\pi)$
    - Generate random r = sqrt(U) * radius, where $U \in [0, 1]$
    - Convert to Cartesian:
    - x = x_center + r * cos(angle)
    - y = y_center + r * sin(angle)

5. Edge Cases

- Very small or very large radius values (handled by float precision)
- Center at negative coordinates
- Points exactly on the boundary (allowed)
- High volume of calls (30,000+) → ensure performance

6. Complexity Analysis

□ Time Complexity:

- O(1) per call to randPoint() — Constant time to compute coordinates.

□ Space Complexity:

- O(1) — Only storing constants (radius, x_center, y_center)

7. **Alternative Approaches**

Method	Description	Pros	Cons
Rejection Sampling	Randomly pick in bounding square, reject if outside circle	Conceptually simple	Inefficient (~21% rejection rate)
Polar Coordinates ✅	Sample angle and radius using sqrt(U)	Efficient and uniform	Slightly more math involved

8. **Test Cases**

✅ Basic Test

```
obj = Solution(1.0, 0.0, 0.0)
print(obj.randPoint())  # Example: [0.1123, -0.7421]
```

✅ Non-zero Center

```
obj = Solution(2.0, 3.0, 4.0)
print(obj.randPoint())  # Example: [4.234, 5.982]
```

✅ Edge Case - Small Radius

```
obj = Solution(0.0001, 1.0, 1.0)
print(obj.randPoint())  # Should be very close to (1.0, 1.0)
```

9. **Final Thoughts**

- Using polar coordinates with a scaled radius is the most optimal and mathematically correct way to uniformly sample points in a circle.
- Ensures performance even with high number of calls.
- Covers all edge cases, including boundary conditions.