Q 1.1: What kind of coordinate system can we use to denote the cells in a pointy top hex grid?

A: Maybe we can use like what we do in chess (e.g B3 C6 F4)

Q 1.2: If there are alternatives, which one provides the easiest method to compute distances, or perform intersections on ranges as depicted above?

A: It may also make sense to calculate the area of ​​a circle, but as I mentioned in the example, specifying it with numbers, letters or symbols in the x and y directions can make it simpler and more understandable.

Q 2: Which data structure is better suited to store the entire map?

A: I search each data structure from linear to non-linear data structure. But I think Axial Coordinates is much better for this types of map

Q 3.1: Which data structure is better suited to store a region defined by the sensor reading?

A: In class we see tree method (R-trees) for this stuation we can apply same procedure.

Q 3.2: Does it matter if the region is a circle or a ring?

A: Yes, the shape of the region (circle or ring) can influence the choice of data structure for storing and querying sensor readings.

Q 4: Implement with Java the coordinate system, the map, and finding the intersection. Your program should get inputs as   
- Number of cells in map, or an indicator of its dimensions (ie. rows, columns, etc)

- Number of cells with radar responses

- Coordinates of cells with radar responses (repeats until the number indicated is satisfied) Then your program should output

- The number of cells in the intersection,

- Their coordinates.

**import java.util.\*;  
  
class Hex {  
 int q, r; // Axial coordinates  
  
 Hex(int q, int r) {  
 this.q = q;  
 this.r = r;  
 }  
  
 // Calculate distance between two hexes in axial coordinates  
 int distanceTo(Hex other) {  
 return (Math.*abs*(q - other.q) + Math.*abs*(q + r - other.q - other.r) + Math.*abs*(r - other.r)) / 2;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) return true;  
 if (obj == null || getClass() != obj.getClass()) return false;  
 Hex hex = (Hex) obj;  
 return q == hex.q && r == hex.r;  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(q, r);  
 }  
}  
  
class RTreeNode {  
 List<Hex> hexCells;  
  
 RTreeNode() {  
 hexCells = new ArrayList<>();  
 }  
  
 void insert(Hex hex) {  
 hexCells.add(hex);  
 }  
  
 List<Hex> queryRange(Hex center, int radius) {  
 List<Hex> result = new ArrayList<>();  
 for (Hex hex : hexCells) {  
 int distance = center.distanceTo(hex);  
 if (distance <= radius && distance > radius - 1) {  
 result.add(hex);  
 }  
 }  
 return result;  
 }  
}  
  
public class HexSensorSystem {  
  
 public static void main(String[] args) {  
 RTreeNode rTree = new RTreeNode();  
  
 // Populate the R-tree with hex cells  
 for (int q = -5; q <= 5; q++) {  
 for (int r = -5; r <= 5; r++) {  
 if (Math.*abs*(q + r) <= 5) {  
 rTree.insert(new Hex(q, r));  
 }  
 }  
 }  
  
 // Simulate a detection at (0, 0)  
 Hex sensorLocation = new Hex(2, 3);  
 int detectionRadius = 2;  
  
 // Find hex cells within a ring of distance 3  
 List<Hex> detectedHexes = rTree.queryRange(sensorLocation, detectionRadius);  
  
 System.*out*.println("Detected hex cells at distance " + detectionRadius + ":");  
 for (Hex hex : detectedHexes) {  
 System.*out*.println("Hex(" + hex.q + ", " + hex.r + ")");  
 }  
 }  
}**

Q 5: Your report should also include a single test case

- A sketch (hand made and photographed is acceptable) which marks the towers and the regions, and the intersection.

- Screen shot of your program working, taking the inputs

- Screen shot of your program working, showing the outputs