1 Motivation

We assume that you have used the 'search nearby' feature of Google maps to search for restaurants close to you. (Use it and party with your friends over the upcoming weekend.) How should Google store the locations of restaurants so that it can answer such queries fast? A naïve idea is use a list and, given a query, scan the list and return the subset of nearby restaurants. This requires time $O(n)$ on a list of n locations. How can we speed this up in practice? Note that the list of restaurants remains fairly unchanged over time, while 'search nearby' queries are much more frequent. Therefore, it makes sense to pre-process the list of restaurants and create an appropriate data structure that enables processing 'search nearby' queries much faster than a brute-force search (assuming the number of "nearby" restaurants is much smaller than the total number of restaurants, which is typically the case).

2 Problem Statement

The ℓ_∞-distance between two points – $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, denoted by $\|p_1 - p_2\|_\infty$, is defined as $\max(|x_1 - x_2|, |y_1 - y_2|)$. Given a list S of n data points in $\mathbb{R}^2$, a query point $q \in \mathbb{R}^2$, and a non-negative number d, we wish to find the set $\{p \in S \mid \|p - q\|_\infty \leq d\}$, the set of all points in S that are at most an ℓ_∞-distance d away from the point q. You are required to design and program an appropriate data structure, which we call `PointDatabase`, to store a set of data points that admits the following.

1. An algorithm which, given a list S of data points, outputs an object $R(S)$ of `PointDatabase` which is a "representation" of S.
2. An algorithm which, given the object $R(S)$ of `PointDatabase`, a query point q, and a distance d, outputs the set $\{p \in S \mid \|p - q\|_\infty \leq d\}$.

More specifically, your task is to implement a Python class `PointDatabase` with the following methods.

- `__init__(self, pointlist)`: A constructor which, given a list `pointlist` of pairs of numbers, creates an object of `PointDatabase`. This method must run in time $O(n \log n)$, where n is the number of points in `pointlist`.
- `searchNearby(self, q, d)`: An accessor method which, given a point `q` and distance `d`, returns the list of all points, in the set that `self` represents, that are at ℓ_∞-distance at most `d` from `q` (arranged in an arbitrary order). This method must run in time $O(m + \log^2 n)$, where n is the cardinality of the set of points that `self` represents, and m is the number of points returned.

For simplicity, you may assume the following.

1. The coordinates of all points involved are integers (thus, avoiding floats and the resulting errors).
2. In the list `pointlist` from which a `PointDatabase` object is constructed, no two data points have the same x-coordinate, and no two data points have the same y-coordinate.
3. No data point is at ℓ_∞-distance exactly d from the query point q.

3 Submission Specifications

Submit a single file named `a3.py`. Your submitted file must contain an implementation of the `PointDatabase` class with the two methods `__init__(self, pointlist)` and `searchNearby(self, q, d)` mentioned above. The signatures of these two methods should match exactly with the signatures given. Of course, you are allowed to implement other methods in the class, as needed.

4 Test-case Specification

Assuming the auto-grader finds the file `a3.py` and the required class and methods implemented, the auto-grader will run several test-cases. Each test-case will involve constructing one object of `PointDatabase` from a list of points, followed by running multiple `searchNearby` queries on it. If the constructor call in a test-case fails or times out, the subsequent queries will not be run and you will lose all marks for that test-case. Here is an example test-case.

```
>>> pointDbObject = PointDatabase([(1,6), (2,4), (3,7), (4,9), (5,1), (6,3), (7,8), (8,10),
 (9,2), (10,5)])
>>> pointDbObject.searchNearby((5,5), 1)
[]
>>> pointDbObject.searchNearby((4,8), 2)
[(3,7), (4,9)]
>>> pointDbObject.searchNearby((10,2), 1.5)
[(9,2)]
```

The following image is a visual aid for understanding the above test-case.