CSL 852, Computational Geometry: Practice Problems

Randomized incremental construction and random sampling

1. Provide all details of how you will implement the grid-based approach to determine all pairs that are closer than distance δ.
   In a model that supports floor function show how it can be implemented in linear time and linear space and otherwise it may take $O(n \log n)$ time.
   Hint: First try to come up with a scheme in 1 dimension and extend it to d dimension by a cartesian product construction.

2. Let $\mathcal{D}$ be a set of n disks of radius r.

   (a) Design a linear time algorithm to find out if there are any pair of intersecting disks (only the decision version), assuming floor and ceiling operations can be performed in $O(1)$ times. Describe the associated data structure in details.

   Solution: If two disks intersect, it implies that their centers are closer than $2r$. So the problem is to determine if the closest pair has distance less than $2r$. In a $2r \times 2r$ grid, each square can contain at most 4 points whose distances are less than $2r$. If the closest pair has distance less than $2r$ then they must be in the same square or in neighboring squares where a square has 8 neighbours. We map the centers to this grid (actually hash them using ceiling function) and check for this condition by inspecting $O(1)$ neighbouring grids. As there are at most $O(n)$ squares to inspect, the algorithm takes linear time.

   (b) If there there is no overlap, describe an algorithm to to report the closest pair of disks in $O(n \log n)$ time using randomized incremental construction.

   Solution: We add the points in a random order and maintain closest pair. The probability that a closest pair has to be updated is less than $2/i$ for the i-th step by backward analysis as the closest pair is defined by two points. We use the previous data structure (a grid of size of the closest pair) to check this and rebuild it if the closest pair has changed. The expected time for each step is $2/i \cdot i = O(1)$ if we use hashing or $O(\log n)$ if we store the squares in a dictionary. Thus the total expected running time is $O(n \log n)$.

   Hint: Use randomized incremental construction.