1. (25 pts) Random Incremental Voronoi Diagram

A Voronoi diagram of a region $R$ of the plane with respect to point set $S$, referred to as sites, is a partitioning of $R$ into regions based on distance to sites in $S$. Specifically, it is a partitioning of the points of $R$ into regions called cells, where each cell is the set of points closest to a site in $P$. See Figure 1 for an illustration.

![Voronoi Diagram](image)

Figure 1: A voronoi diagram of the unit square. Sites are marked in black and their regions are marked in color.

Suppose we maintain a Voronoi diagram of the unit square, and add $n$ sites in some order. Whenever a site is added we color its (new) cell. Thus, when the first site is added we color the entire unit square, when the second site is added we color its cell in the Voronoi diagram of the unit square with only the first two sites, and so on. The amount of paint used on arrival of a site is equal to the area of the new cell of the arrived. Here we will bound the expected amount of paint used.

(a) Give a set of $n$ distinct sites $S$ and arrival order where $\Omega(n)$ paint is used.
(b) Show that for any set of $n$ sites $S$ which are added in random order $O(\log n)$ paint is used.
2. (25 pts) **Rectangular Tiling**

Consider a set \( R \) of \( m \) disjoint axis-aligned rectangles. These rectangles are contained inside of the \( n \times n \) square \( S \) whose lower left corner is at \((0,0)\) and upper right corner is at \((n,n)\).

Describe an algorithm that takes \( R \) and \( S \) as input and computes another set \( R' \) of disjoint axis-aligned rectangles such that (1) all the rectangles of \( R \) and \( R' \) are disjoint (except possibly along their boundaries), (2) the union of \( R \) and \( R' \) is \( S \), and (3) \(|R'| \leq 3m + 1\). The running time of your algorithm should be \( O(m \log m) \). Prove that your algorithm has the desired running time and produces the required output.

3. (25 pts) **Red-Green Closest Pair**

Suppose we have two sets of points in the unit interval, \( n \) red points \( R \) and \( m \) green points \( G \). The goal of this problem is to find the closest pair of points with one point from \( R \) and the other one from \( G \). The algorithm should run in expected linear time, \( O(n + m) \). We shall develop a hashing based algorithm similar to the 2D closed pair algorithm presented in class.

(a) Suppose we are given the closest bichromatic distance \( \alpha \). Give a linear expected time algorithm to find a pair of points realizing this distance.
   
   Hint: Try looking for a pair such that one point is in one bucket and the other is in a neighboring one.

(b) Similar to the closest pair algorithm from class define a function \( \text{Lookup}(\text{Grid}, p) \) and give an efficient algorithm for it.

(c) Give your complete algorithm with proof of correctness and run time analysis.
4. (25 pts) **Pickup Sticks, a Programming Problem**

The input consists of \( n \) non-intersecting segments in the plane. The goal is to compute a valid order in which the segments can be removed. A removal order is valid if when it’s time for a segment to be removed it can be moved purely in the \( y \) direction and never touch any of the remaining segments (which have not yet been removed) along the way. A valid ordering always exists. The time limit is 10 seconds (5 seconds for C and C++).

There are three valid orderings for the segments shown above. They are \([1, 4, 0, 2, 3]\), \([1, 4, 2, 0, 3]\), and \([1, 4, 2, 3, 0]\).

**Input:**

The first line contains the number \( n \leq 150,000 \). Each of the next \( n \) lines describes one of the segments (numbered from 0 to \( n - 1 \) respectively) and consists of four space-separated numbers: \( x_1 \ y_1 \ x_2 \ y_2 \) with \( x_1 < x_2 \). All of these coordinates will be integers in the range \([0, 10^8]\). The leftmost end of this segment is point \((x_1, y_1)\), and the rightmost endpoint will be \((x_2, y_2)\), and there are no vertical segments.

**Output:**

Output a permutation of 0 to \( n - 1 \), with one number on each line, which is an ordering in which the segments can be removed (by moving them up) without ever touching (even at a single point) any other segment. There may be more than one correct answer. Output any of them.

**Example:**

Below on the left is an input that corresponds to the example above. On the right is one possible correct output.

```
5
8 2 10 3
2 9 4 8
3 5 5 3
1 2 5 2
2 6 10 4
```