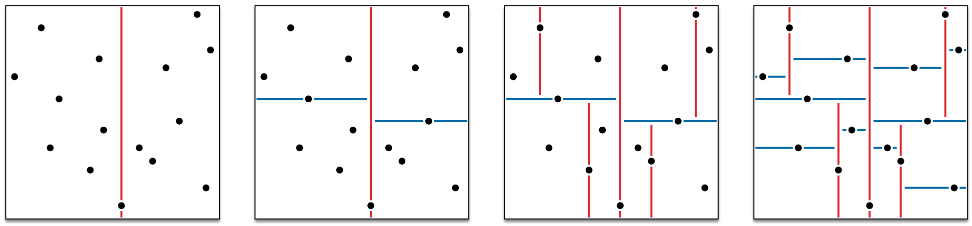
1. Suppose we are given a set P of n points in the plane. A *kd-tree*[1] for P recursively subdivides the points as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point in the interior of the box (*not* on its boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



Building a kd-tree for 15 points.

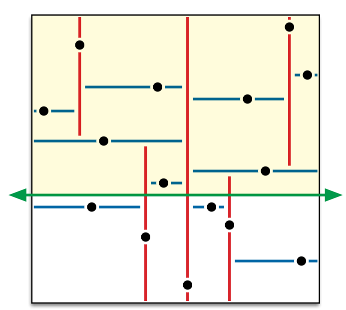
Formally, a kd-tree is a perfectly balanced binary tree in which each node v stores the following information:

- $v.x$ and $v.y$: The coordinates of the point defining the cut at v
- $v.dir \in \{vertical, horizontal\}$: The direction of the cut at v.
- $v.left$ and $v.right$: The children of v if $v.dir = vertical$
- $v.up$ and $v.down$: The children of v if $v.dir = horizontal$
- $v.size$: the number of nodes in the subtree rooted at v.

Describe and analyze an algorithm that answers the following query in $O(\sqrt{n})$ time, assuming the points P are stored in a kd-tree.

> CountAbove(b): Return the number of points in P that lie above the horizontal line $y = b$.

To avoid some boundary cases, assume that $n = 2^k - 1$ for some integer k, that all points in P have distinct x- and y–coordinates, and that no point in P lies directly on the line $y = b$. *[Hint: How many boxes does the query line intersect?]*



There are 9 points above the green line.

[1]The name "kd-tree" was originally an abbreviation for "k-dimensional tree", which suggests that I really should call this example a "2d-tree". Over time this meaning has been mostly forgotten, so most modern users would refer to this data structure as a "two-dimensional kd-tree". See also: Sahara Desert, Mississippi River, Lake Tahoe, La Brea Tar Pits, and DC Comics. Also, who in their right mind uses the letter k to stand for *dimension*?

The remaining problems are for you play with on your own.
Discussion in office hours or on Discord is welcome, but don't submit solutions!

---

2. Suppose we are given a set P of $n = 2^k - 1$ points in the plane with distinct coordinates, stored in a kd-tree. Describe how to answer each of the following queries in $O(\sqrt{n})$ time. If necessary specify any additional information that must be stored at each node in the kd-tree (like $v.size$ for question 1).

   (a) $\textsc{LowestAbove}(b)$: Return the lowest point $(x, y) \in P$ such that $y > b$.

   (b) $\textsc{LeftmostBelow}(t)$: Return the leftmost point $(x, y) \in P$ such that $y < t$.

   (c) $\textsc{LineRight}(k)$: Return a real number a such that there are exactly k points $(x, y) \in P$ where $x < a$.

   (d) $\textsc{CenterLeft}(r)$: Return the *center of mass* or *average* of all points $(x, y) \in P$ such that $x < r$. (The center of mass of k points $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ is the point $(\sum_{i=1}^k \frac{x_i}{k}, \sum_{i=1}^k \frac{y_i}{k})$.) *[Hint: For each node v, separately maintain the number of points in v's subtree, the sum of their x-coordinates, and the sum of their y-coordinates.]*

   (e) $\textsc{AboveRight}(\ell, b)$: Return the number of points $(x, y) \in P$ such that $x > \ell$ and $y > b$.

   (f) $\textsc{BoxCount}(\ell, r, b, t)$: Return the number of points $(x, y) \in P$ such that $\ell < x < r$ and $b < y < t$.

   (g) $\textsc{BoxFar}(\ell, r, b, t)$: Return the farthest point $(x, y) \in P$ from the origin (that is, maximizing the function $x^2 + y^2$) such that $\ell < x < r$ and $b < y < t$.

   (h) L_1-$\textsc{Neighbor}(a, b)$: Find the largest diamond (square rotated 45°) centered at (a, b) with no point in P in is interior, and return a point in P that lies on the boundary of that diamond.

   *(i) L_∞-$\textsc{Neighbor}(a, b)$: Find the smallest axis-aligned square $\square$ centered at (a, b) with no point in P in is interior, and return a point in P that lies on the boundary of $\square$. (This one might require $O(\sqrt{n} \log n)$ time.)

*3. There are several ways to add support for insertions and deletions in kd-trees.

(a) Show that using the Bentley-Saxe logarithmic method (described in Homework 3 problem 4) to support insertions, and using tombstones and global rebuilding to support deletion, we get the following amortized time bounds:

- Insert: $O(\log^2 n)$ amortized time
- Delete: $O(\log n)$ amortized time
- Any of the queries for problem 1 or 2: $O(\sqrt{n})$ worst-case time.

(b) Suppose we allow the kd-tree to use *approximate* medians to define cuts, so if a node has size m, its children each have size at most αn for some constant $\alpha > 1/2$. Show that if we support insertions using a local-rebuilding strategy similar to scapegoat trees, and we implement deletion using tombstones, we can achieve the following time bounds:

- Insert: $O(\log n)$ amortized time
- Delete: $O(\log n)$ amortized time
- Any of the queries for problem 1 or 2: $O(n^\beta)$ worst-case time, where $\beta > 1/2$ is a constant that depends on α.