R-21.5

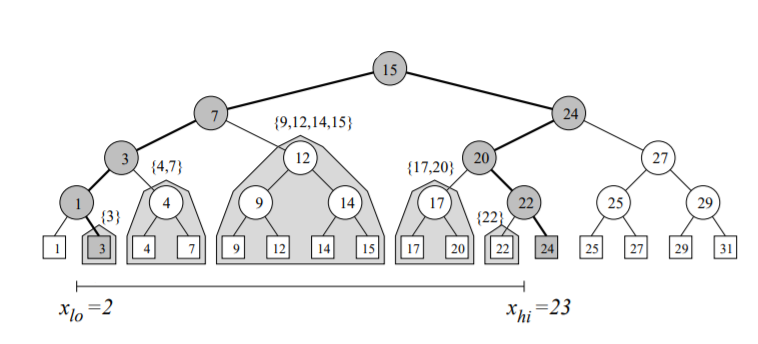
The worst-case distributions of n points will be a region-based k-d tree, which split on longest side or alternate same length side at exactly in half by a line perpendicular to this side. The region-based k-d tree’s depth is depending on the stop value we set.

Note that if we set the stop width to 0 or the smallest value closing to 0 this machine can hold and put all points there. Then this dividing would keep going until the width reached the smallest value the machine could hold. The depth would be the number of divides before reaching the stop width. In this approach, the number of dimensions does not matter. If k>=2, the line would just be an axis-aligned hyperplane.

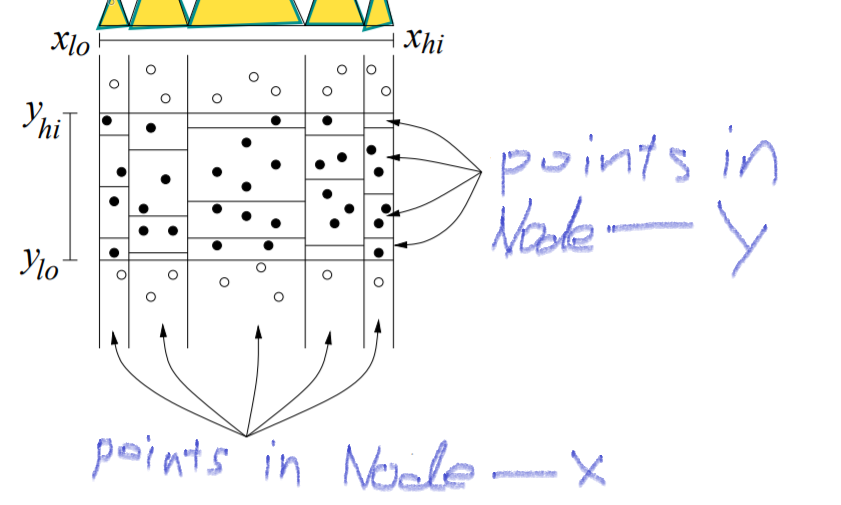
Instead of sort of cheating and setting the stop width to 0, we can also set the stop point in the plane to 1 with a region-based k-d tree. Then suppose the n points are distributed in a way that each divide would isolate exactly one point (like a spinning curve). Then the depth would be the number of points: n.

C-21.5

We need to build a 2D Range tree for points with the form (x,y) and query in (xlo,xhi,ylo,yhi). We first ignore the y-coordinates and build a 1D x-range tree on Set S as a balanced binary search tree T. In T, the node v contains an item with (xv, yv) coordinates and an element(v), and another auxiliary one-dimensional range tree T(v) that stores the same set of items as the subtree rooted at node v with y-coordinates as keys. For example, to search (2,23, ylo, yhi):



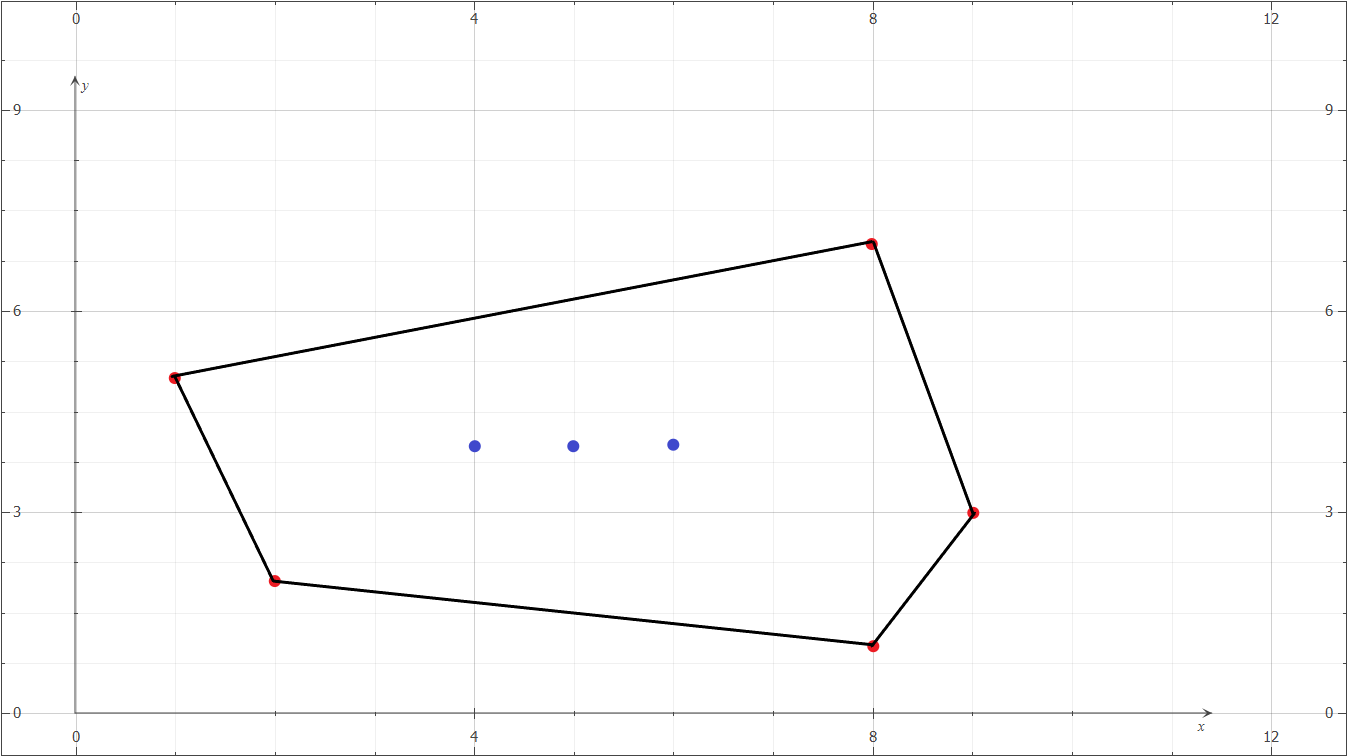
Then we can see that x-coordinates nodes {3,4,7,9,12,14,15,17,20,22} are included first. Then for each Node, we find the y-coordinates tree with the associated points like:



The search on this data structure is just double 1-D range search. Since for each 1-D range search, we got log n nodes and log n subsets which would be found in O(log n) time. Thus, for 2-D range search we need O((log n)^2) time. For 1-D range tree we need O(n) space, but for each node there is another subset to included, and sums up to O(n log n). (one subset is of size n, two of size n/2, etc.)

A-21.3

For a regular PST, we need to do the preliminary sorting of the items of S by x-coordinate with a sorting method like merge-sort for O(n log n) to build a dynamic priority search tree. But if we just create a static PST, then we don’t need to sort the data points, which reduce the time it takes to just the time of the bottom-up construction method for a heap, or O(n).

R-22.8

C-22.5

We need to check if when traversing the polygon, every turn must be all left turns or all right turns, else this is not a polygon. Then we need to check the self-intersecting by computing the sum of the internal angles of the polygon. It’s done by going around and summing of all internal angles together. If the sum is less than 180 degrees / (n-2), where n is the number of vertices. If all two checks are passed, then this is a convex polygon. The running time would be O(n + n) where each check takes O(n). The total time would be O(n).

A-22.2

We can check by draw a vertical line to the downside of each point and extend it to infinity. Since we assume that q is not on the boundary of P and that there is no vertex of P with the same x-coordinate as q, we can ignore the cases when the line is colinear with current edges of polygon or the line is through the vertical. We only need to count the number of times the line intersects with the polygon edges. If it’s an odd number, then the point is in the polygon, else lies outside.