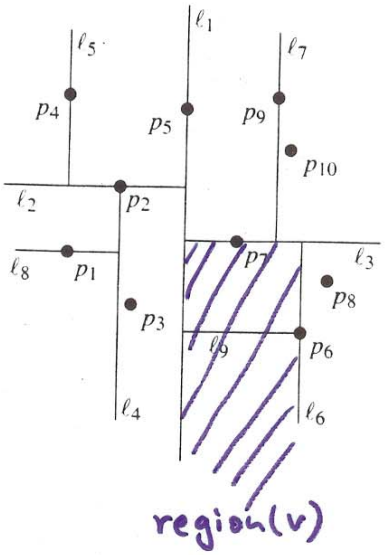
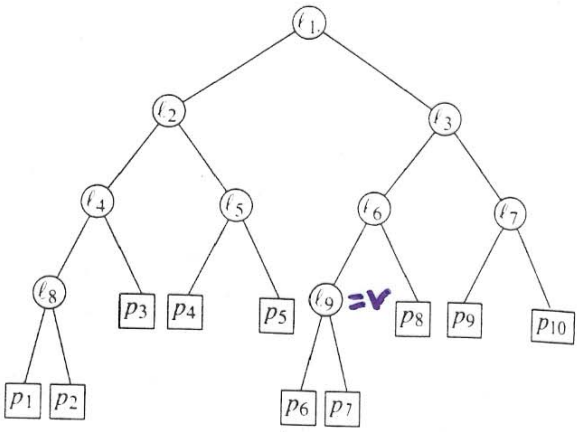
KD-Trees for 2-D Range Searching

Given: A set $P := \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$

Task: Process P into a data structure that allows fast 2D range queries: Report all points in P that lie in the query rectangle $[x, x'] \times [y, y']$

⇒ Recursively split P into two sets of same size, alternatingly along a vertical or horizontal line





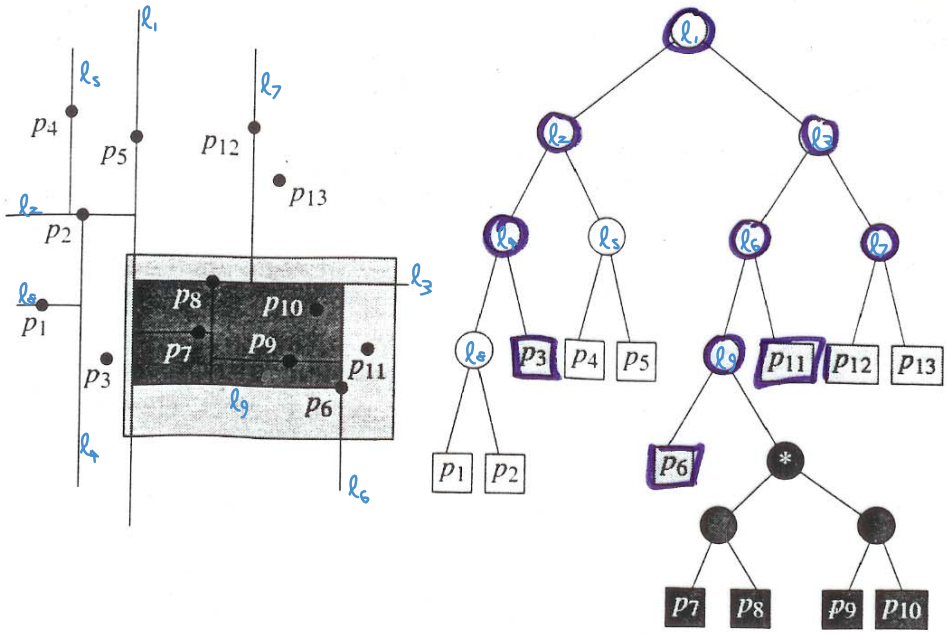
- Sort P by x- and by y-coordinate in advance
- Use these two sorted lists to find median
- Pass sorted lists into recursive calls

⇒ $T(n) = \begin{cases} O(1) & , n = 1 \\ O(n) + 2T(\lceil n/2 \rceil), & n > 1 \end{cases}$

$= O(n \log n)$ time

Algorithm BUILDKDTREE(P, *depth*)
Input. A set of points P and the current depth *depth*.
Output. The root of a kd-tree storing P.

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** *depth* is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let P_2 be the set of points to the right of ℓ.
5. **else** Split P into two subsets with a horizontal line ℓ through the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE(P_1, *depth* + 1)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE(P_2, *depth* + 1)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v



lc(v) = left_child(v)
rc(v) = right_child(v)

Algorithm SEARCHKDTREE(v, R)
Input. The root of (a subtree of) a kd-tree, and a range R.
Output. All points at leaves below v that lie in the range.

```
1.  if v is a leaf
2.     then Report the point stored at v if it lies in R.
3.     else if region(lc(v)) is fully contained in R
4.             then REPORTSUBTREE(lc(v))
5.             else if region(lc(v)) intersects R
6.                     then SEARCHKDTREE(lc(v), R)
7.          if region(rc(v)) is fully contained in R
8.             then REPORTSUBTREE(rc(v))
9.             else if region(rc(v)) intersects R
10.                    then SEARCHKDTREE(rc(v), R)
```

// region(lc(v)) = region(v) ∩ $\ell(v)^{left}$ ⇝ compute on the fly

Theorem: A kd-tree for a set P of n points can be constructed in $O(n \log n)$ time and uses $O(n)$ space. A rectangular range query can be answered in $O(\sqrt{n} + k)$ time; k = # reported points.
(Generalization to d dimensions: Also $O(n)$ storage, $O(n \log n)$ construction time, but $O(n^{1-1/d} + k)$ query time.)

Proof Sketch:

- Sum of # of visited vertices in ReportSubtree = $O(k)$
- # visited vertices that are not in one of the reported subtrees = $O($# regions(v) intersected by a line$)$

⇝ $Q(n)$:= # intersected regions in kd-tree of n points whose root contains vertical splitting line

⇝ $Q(n) = 2 + 2Q(\frac{n}{4})$, $n > 1$ ⇒ $Q(n) = O(\sqrt{n})$