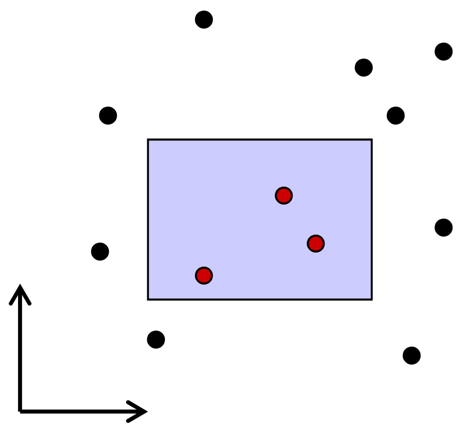
CMPS 3130/6130 Computational Geometry
Spring 2015



Orthogonal Range Searching II

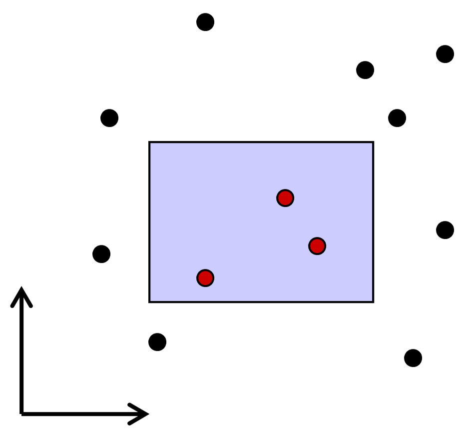
Carola Wenk

Orthogonal range searching

Input: A set P of n points in d dimensions

Task: Process P into a data structure that allows fast orthogonal range queries. Given an axis-aligned ***box*** (in 2D, a rectangle)

- Report on the points inside the box:
  - Are there any points?
  - How many are there?
  - List the points.

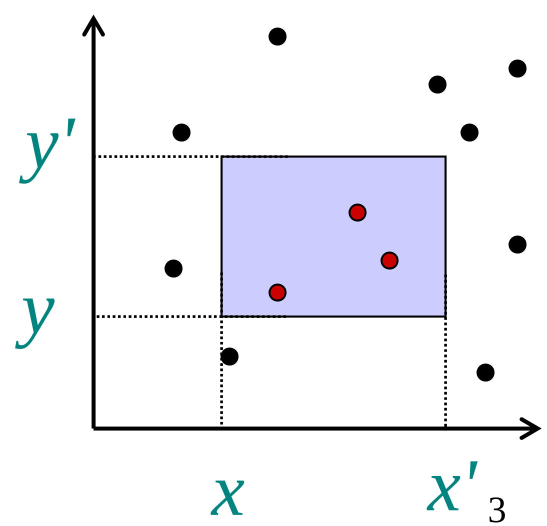


Orthogonal range searching: KD-trees

Let us start in 2D:

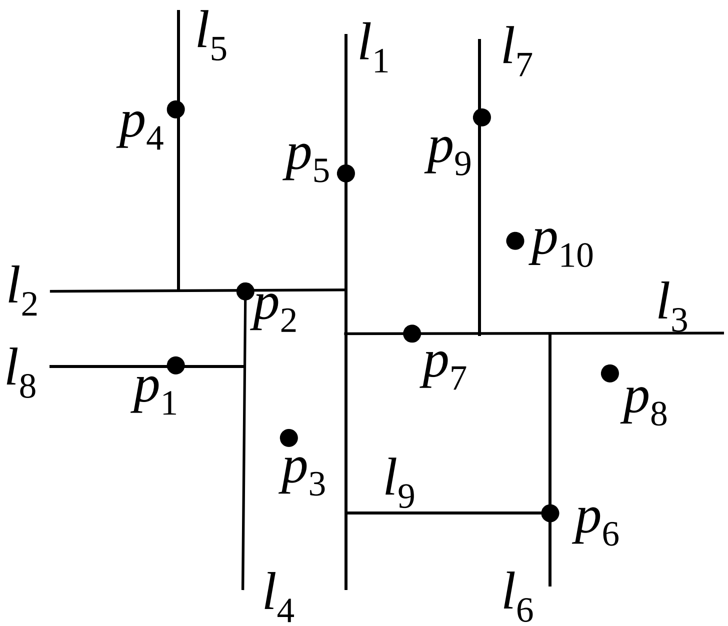
Input: A set *P* of *n* points in 2 dimensions

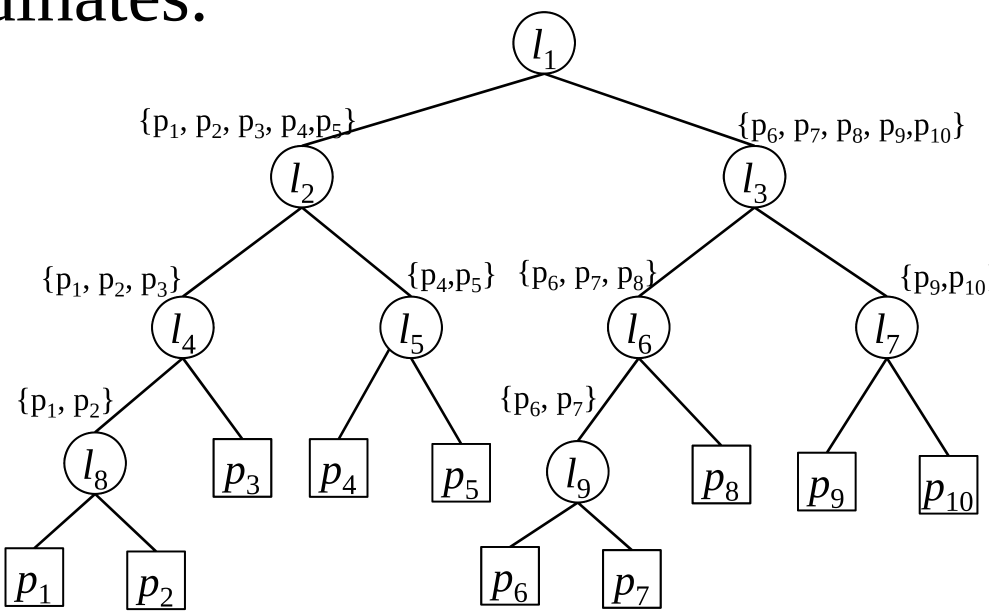
Task: Process *P* into a data structure that allows fast 2D orthogonal range queries: Report all points in *P* that lie in the query rectangle $[x,x'] \times [y,y']$



KD trees

Idea: Recursively split P into two sets of the same size, alternatingly along a vertical or horizontal line through the median in x- or y-coordinates.

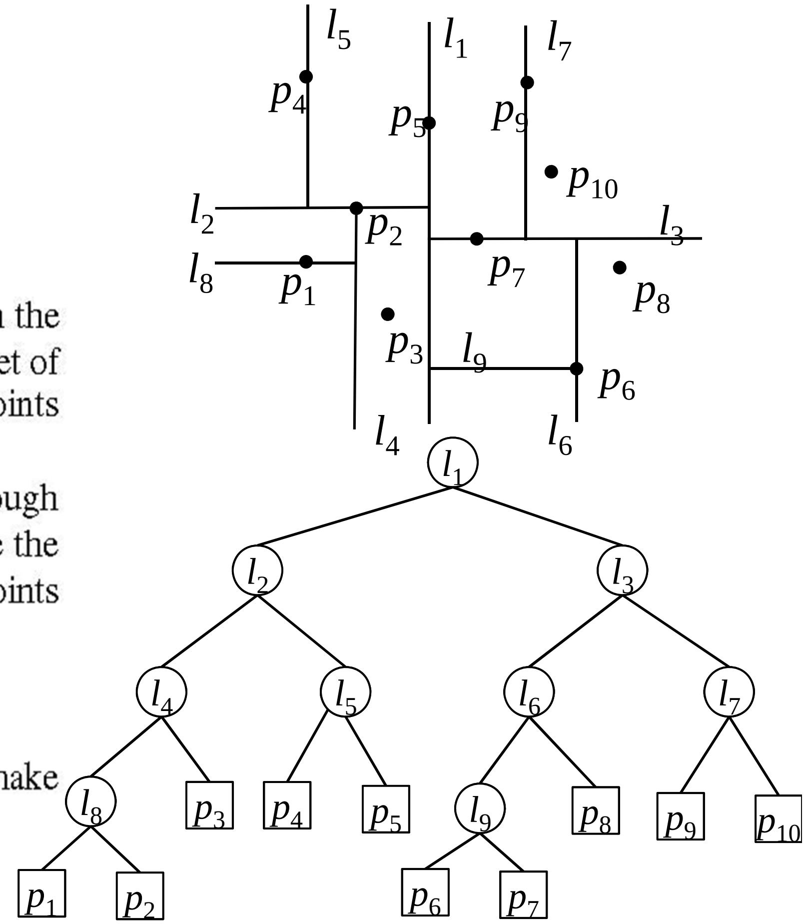




BuildKDTree

Idea: Recursively split P into two sets of the same size, alternatingly along a vertical or horizontal line through the median in x- or y-coordinates.

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. $\nu_{\text{left}} \leftarrow$ BUILDKDTREE(P_1, $depth+1$)
7. $\nu_{\text{right}} \leftarrow$ BUILDKDTREE(P_2, $depth+1$)
8. Create a node ν storing ℓ, make ν_{left} the left child of ν, and make ν_{right} the right child of ν.
9. **return** ν



BuildKDTree Analysis

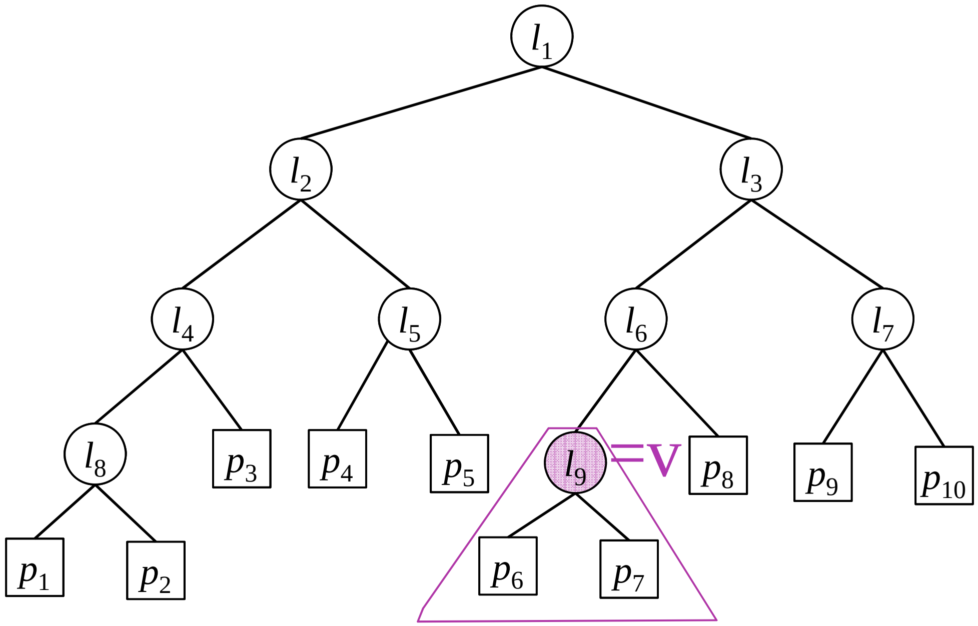
- Sort P separately by x- and y-coordinate in advance
- Use these two sorted lists to find the median
- Pass sorted lists into the recursive calls
- Runtime:

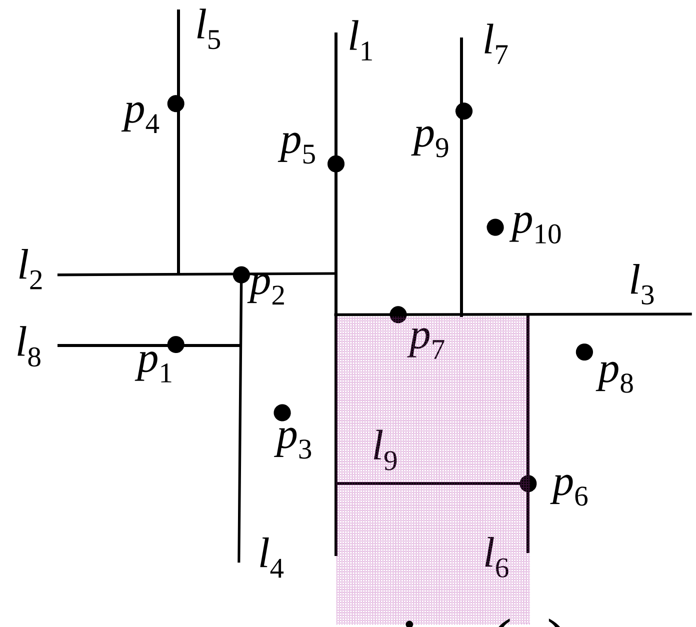
$$T(n) = \begin{cases} O(1) & , n = 1 \\ O(n) + 2T\left(\frac{n}{2}\right) & , n > 1 \end{cases}$$

$$= O(n \log n)$$

- Storage: O(n), because it is a binary tree on n leaves

Regions





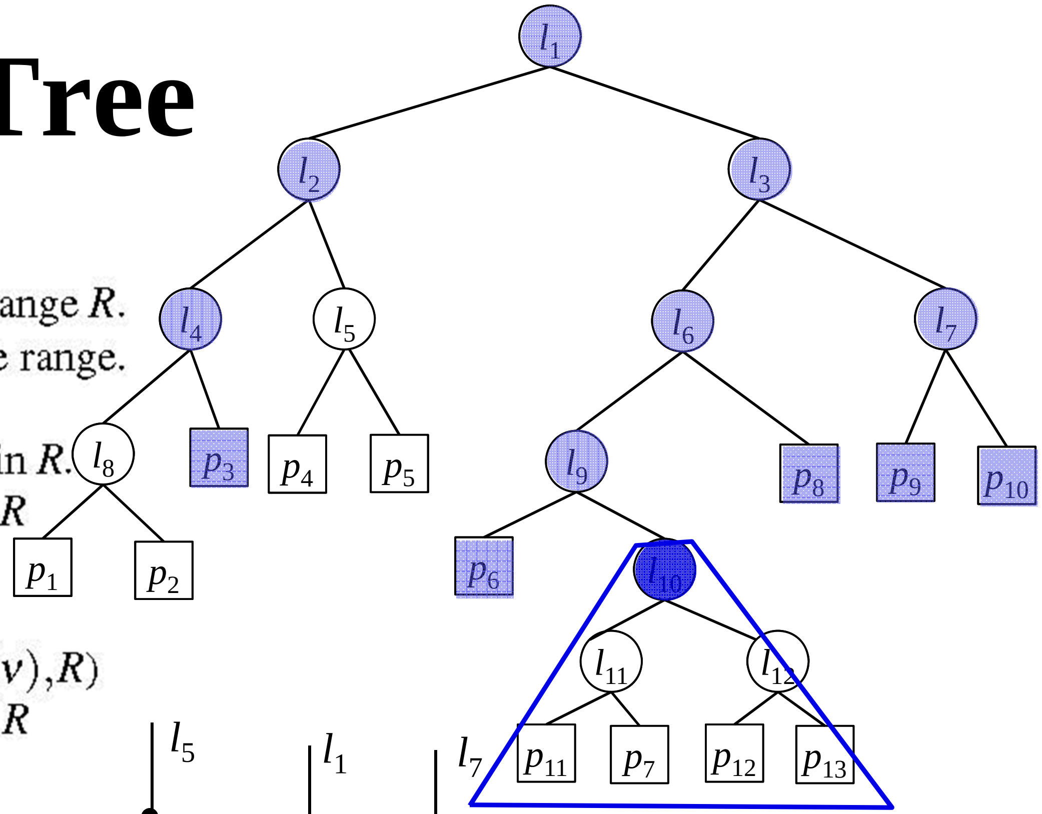
- lc(v)=left_child(v)
- region(lc(v)) = region(v)$\cap$l(v)$^{\text{left}}$

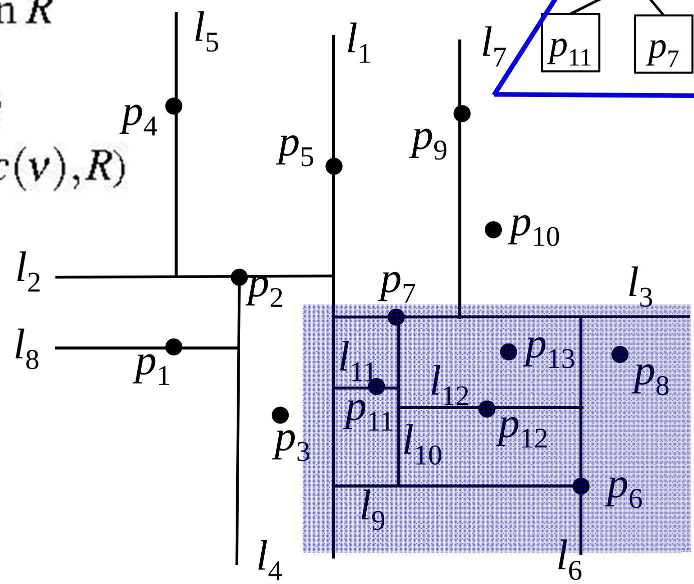
$\Rightarrow$ Can be computed on the fly in constant time

SearchKDTree

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.

```
if v is a leaf
   then Report the point stored at v if it lies in R.
   else if region(lc(v)) is fully contained in R
          then REPORTSUBTREE(lc(v))
          else if region(lc(v)) intersects R
                 then SEARCHKDTREE(lc(v),R)
        if region(rc(v)) is fully contained in R
          then REPORTSUBTREE(rc(v))
          else if region(rc(v)) intersects R
                 then SEARCHKDTREE(rc(v),R)
```





How many nodes does a search touch?

SearchKDTree Analysis

Theorem: A kd-tree for a set of n points in the plane 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, where k = # reported points.
(Generalization to d dimensions: Also $O(n \log n)$ construction time and $O(n)$ space, but $O(n^{1-\frac{1}{d}} + k)$ query time.)

SearchKDTree Analysis

Proof Sketch:

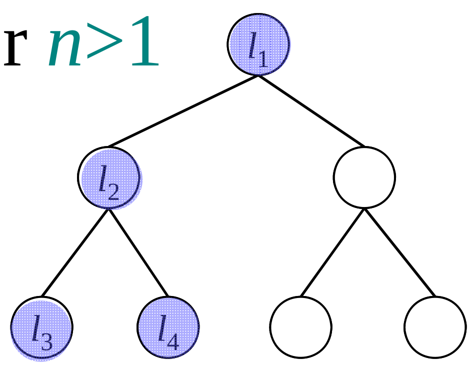
- Sum of # visited vertices in ReportSubtree is $O(k)$
- # visited vertices that are not in one of the reported subtrees = O(# regions(v) intersected by a query line)

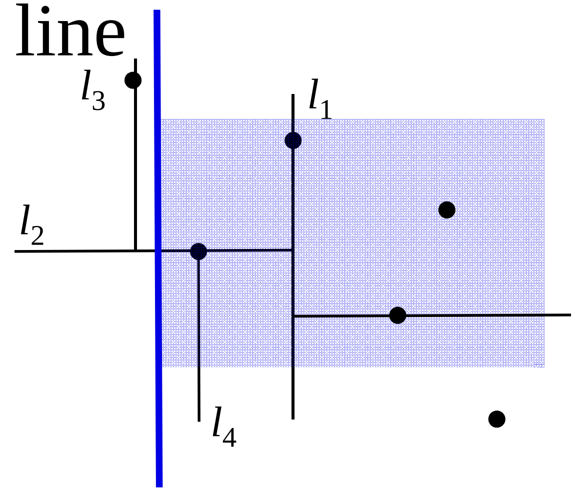
$\Rightarrow$ Consider intersections with a vertical line only.

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

$\Rightarrow$ $Q(n) = 2 + 2Q(n/4)$, for $n>1$

$\Rightarrow$ $Q(n) = O(\sqrt{n})$





Summary Orthogonal Range Searching

Range trees

Query time: $O(k + \log^{d-1} n)$ to report k points

(uses fractional cascading in the last dimension)

Space: $O(n \log^{d-1} n)$

Preprocessing time: $O(n \log^{d-1} n)$

KD-trees

Query time: $O(n^{1-\frac{1}{d}} + k)$ to report k points

Space: $O(n)$

Preprocessing time: $O(n \log n)$