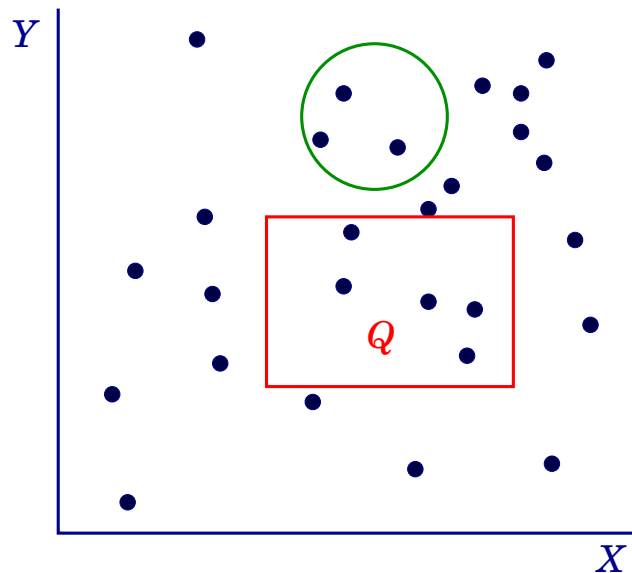
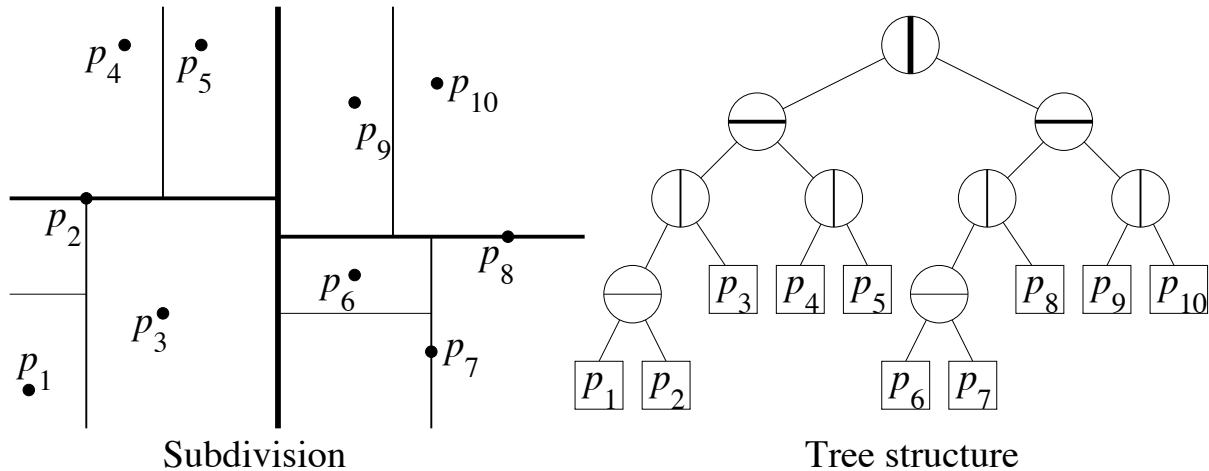


Multi-Dimensional Data



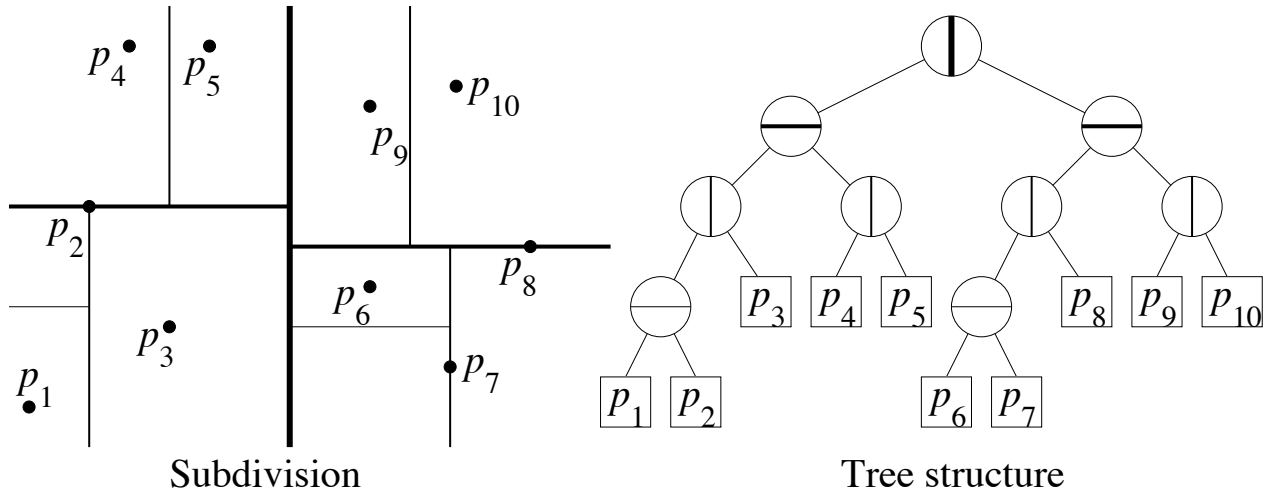
- Range searching in higher dimensions?
- kD -trees [Jon Bentley 1975]. Stands for **k -dimensional trees**.
- Simple, general, and arbitrary dimensional. Asymptotic search complexity not very good.
- Extends 1D tree, but alternates using x - y -coordinates to split. In k -dimensions, cycle through the dimensions.

kD -Trees



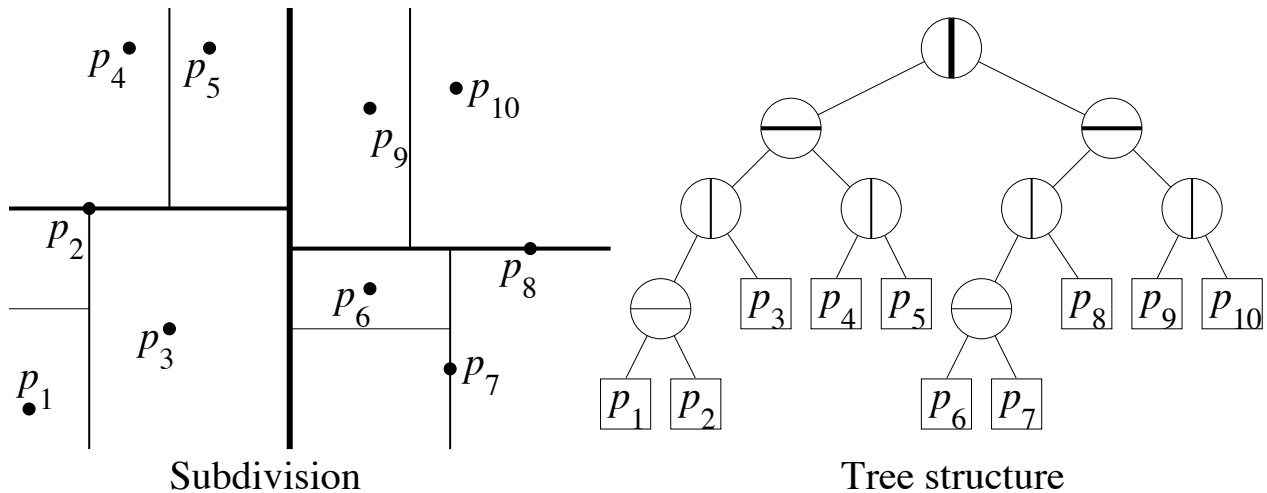
- **A binary tree. Each node has two values: split dimension, and split value.**
- **If split along x , at coordinate s , then left child has points with x -coordinate $\leq s$; right child has remaining points. Same for y .**
- **When $O(1)$ points remain, put them in a leaf node.**
- **Data points at leaves only; internal nodes for branching and splitting.**

Splitting



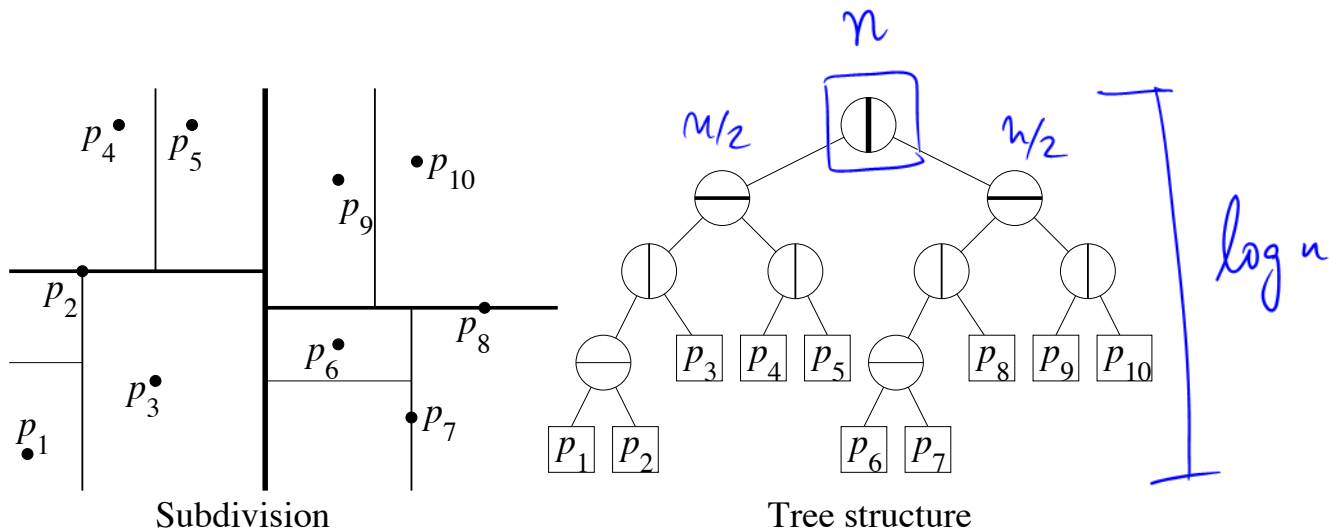
- To get balanced trees, use the **median** coordinate for splitting—median itself can be put in either half.
- With median splitting, the height of the tree guaranteed to be $O(\log n)$.
- Either cycle through the splitting dimensions, or make data-dependent choices. E.g. select dimension with max spread.

Space Partitioning View



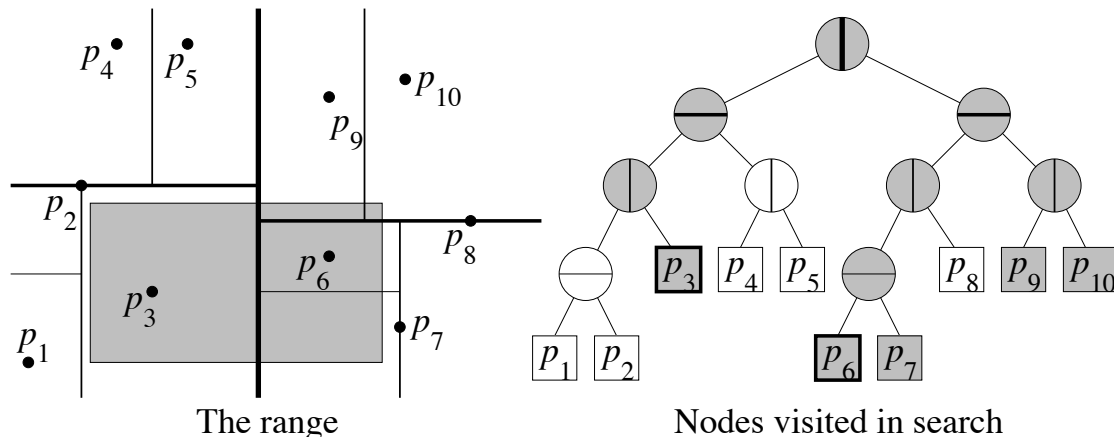
- kD -tree induces a space subdivision—each node introduces a x - or y -aligned cut.
- Points lying on two sides of the cut are passed to two children nodes.
- The subdivision consists of rectangular regions, called **cells** (possibly unbounded).
- Root corresponds to entire space; each child inherits one of the halfspaces, so on.
- Leaves correspond to the terminal cells.
- Special case of a general partition BSP.

Construction



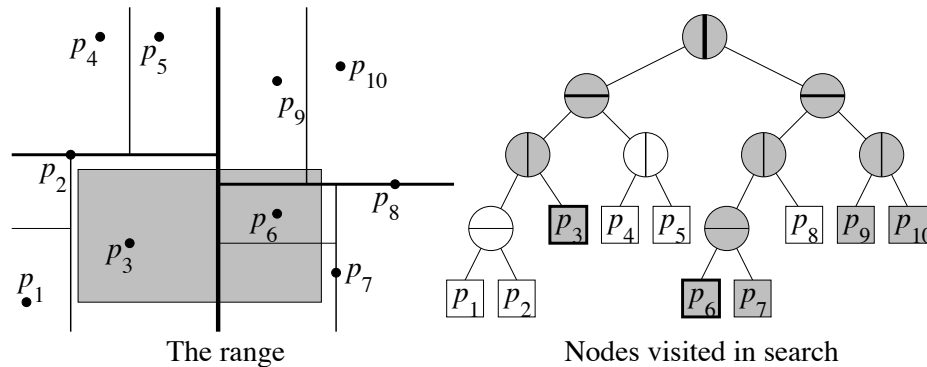
- Can be built in $O(n \log n)$ time recursively.
- Presort points by x and y -coordinates, and cross-link these two sorted lists.
- Find the x -median, say, by scanning the x list. Split the list into two. Use the cross-links to split the y -list in $O(n)$ time.
- Now two subproblems, each of size $n/2$, and with their own sorted lists. Recurse.
- Recurrence $T(n) = 2T(n/2) + n$, which solves to $T(n) = O(n \log n)$.

Searching kD -Trees



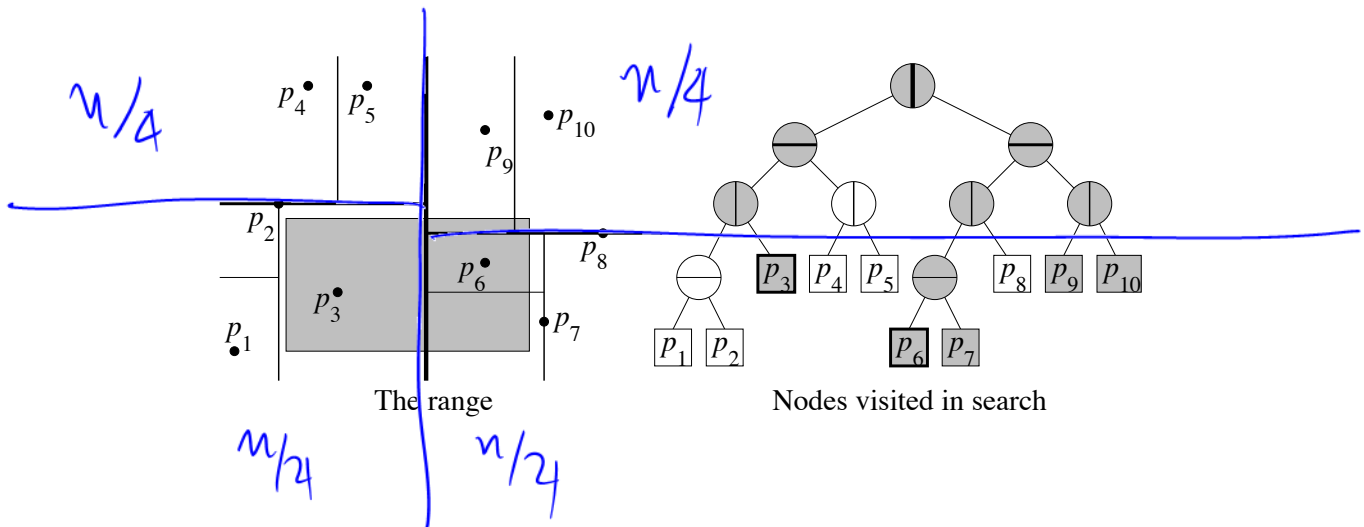
- Suppose query rectangle is R . Start at root node.
- Suppose current splitting line is vertical (analogous for horizontal). Let v, w be left and right children nodes.
- If v a leaf, report $cell(v) \cap R$;
if $cell(v) \subseteq R$, report all points of $cell(v)$;
if $cell(v) \cap R = \emptyset$, skip;
otherwise, search subtree of v recursively.
- Do the same for w .
- Procedure obviously correct. What is the time complexity?

Search Complexity



- When $cell(v) \subseteq R$, complexity is linear in output size.
- It suffices to bound the number of nodes v visited for which the boundaries of $cell(v)$ and R intersect.
- If $cell(v)$ outside R , we don't search it; if $cell(v)$ inside R , we enumerate all points in region of v ; a recursive call is made only if $cell(v)$ partially overlaps R ; the kD -tree height is $O(\log n)$.
- Let ℓ be the line defining one side of R .
- We prove a bound on the number of cells that intersect ℓ ; this is more than what is needed; multiply by 4 for total bound.

Search Complexity



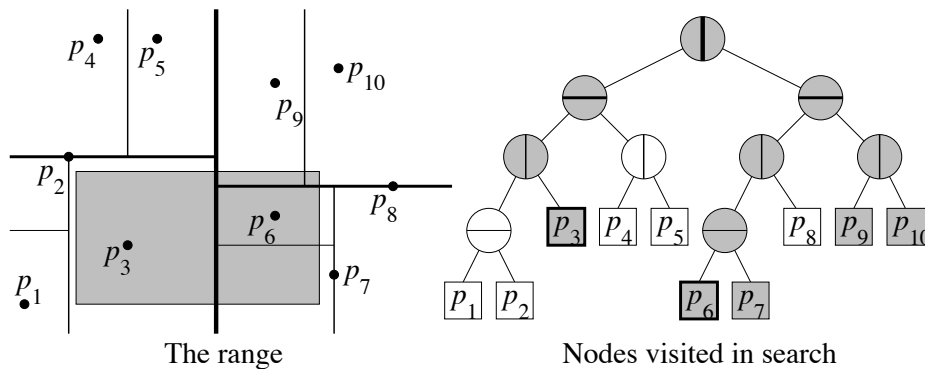
- How many cells can a line intersect?
- Since splitting dimensions alternate, the key idea is to consider two levels of the tree at a time.
- Suppose the first cut is vertical, and second horizontal. We have 4 cells, each with $n/4$ points.
- A line intersects exactly two cells; the others cells will be either outside or entirely inside R .

$$O(Q(n) + k)$$

- The recurrence is

$$Q(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2Q(n/4) + 2 & \text{otherwise.} \end{cases}$$

Search Complexity



- The recurrence $Q(n) = 2Q(n/4) + 2$ solves to

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

- kD -Tree is an $O(n)$ space data structure that solves 2D range query in worst-case time $O(\sqrt{n} + m)$, where m is the output size.

d -Dim Search Complexity

- What's the complexity in higher dimensions?
- Try 3D, and then generalize.
- The recurrence is

$$Q(n) = 2^{d-1}Q(n/2^d) + 1$$

- It solves to

$$Q(n) = O(n^{1-1/d})$$

- kD -Tree is an $O(dn)$ space data structure that solves d -dim range query in worst-case time $O(n^{1-1/d} + m)$, where m is the output size.