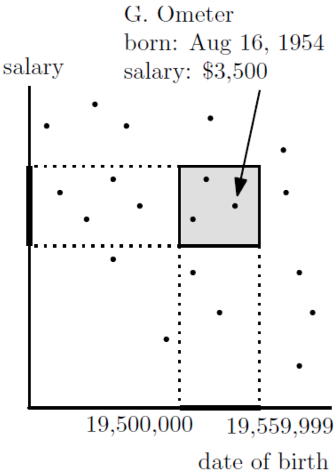
Range Searching

Database queries

A database query may ask for all employees with age between a_1 and a_2, and salary between s_1 and s_2.

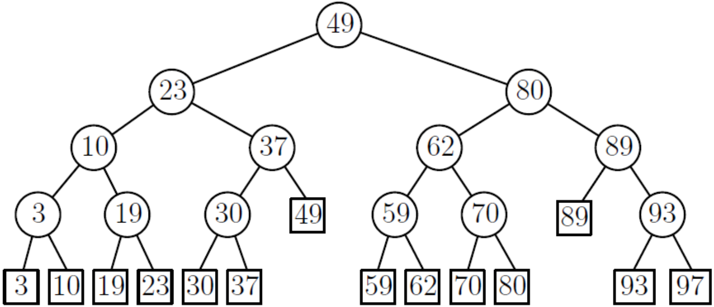


1D range query problem

- **1D range query problem**: Preprocess a set of n points on the real line such that the ones inside a 1D query range (interval) can be reported fast
- The points $p_1, \ldots, p_n$ are known beforehand, the query $[x, x']$ only later
- A solution to a query problem is a data structure description, a query algorithm, and a construction algorithm
- **Question**: What are the most important factors for the efficiency of a solution?

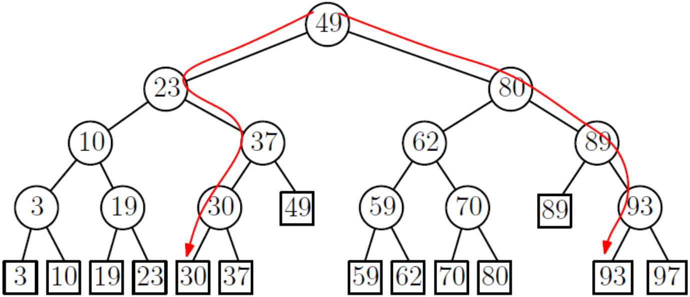
Balanced binary search trees

A balanced binary search tree with the points in the leaves



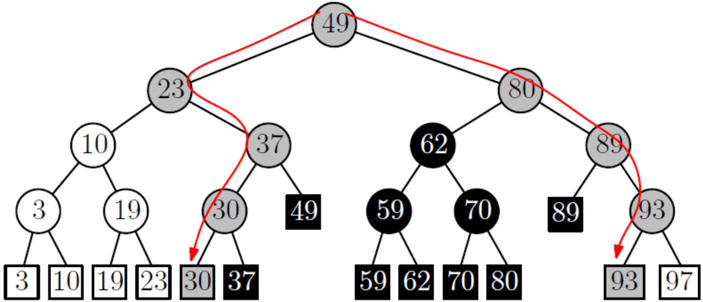
Balanced binary search trees

The search paths for 25 and for 90



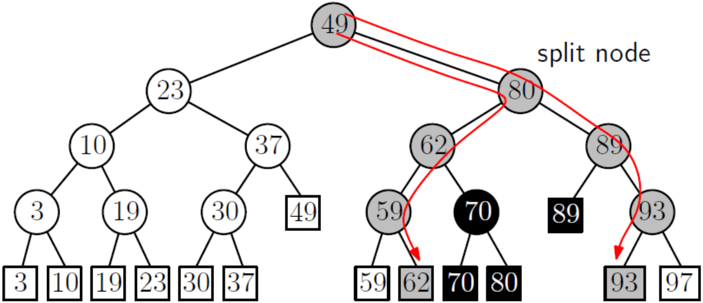
Example 1D range query

A 1-dimensional range query with $[25, 90]$.

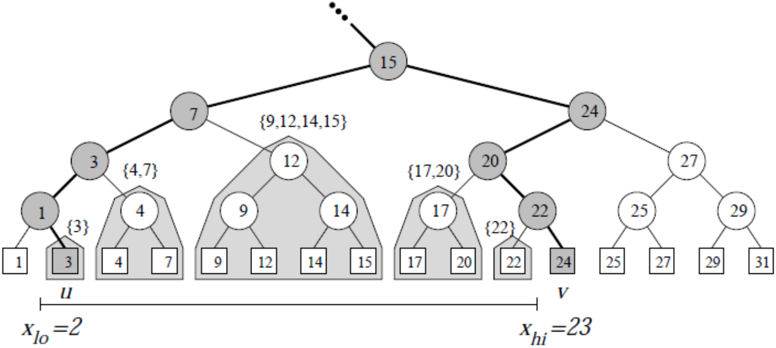


Example 1D range query

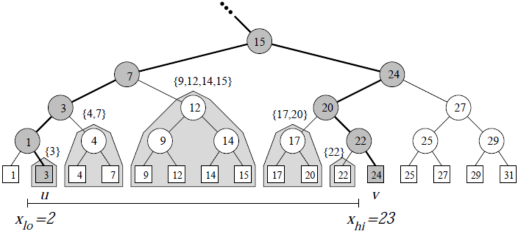
A 1-dimensional range query with $[61, 90]$.



Another Example



Ana;ysis



- Since search paths have $O(\log n)$ nodes, there are $O(\log n)$ canonical subsets, which are found in $O(\log n)$ time.
- To list the sets, traverse those subtrees in linear time, for additional $O(k)$ time.

Storage requirement and preprocessing

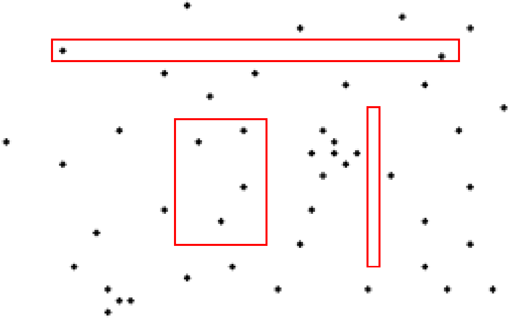
- A (balanced) binary search tree storing n points uses O(n) storage
- A balanced binary search tree storing n points can be built in O(n) time after sorting, so in $O(n \log n)$ time overall (or by repeated insertion in $O(n \log n)$ time)

Theorem 1

A set of n points on the real line can be preprocessed in $O(n \log n)$ time into a data structure of $O(n)$ size so that any 1D range query can be answered in $O(\log n + k)$ time, where k is the number of answers reported

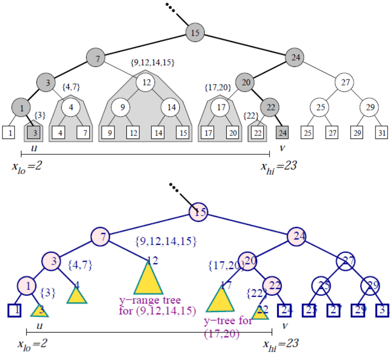
Range queries in 2D

- Kd-trees: Time: $O(\sqrt{n} + k)$; Space: $O(n)$
- range trees: Time: $O(\log^2 n + k)$; Space: $O(n \log n)$

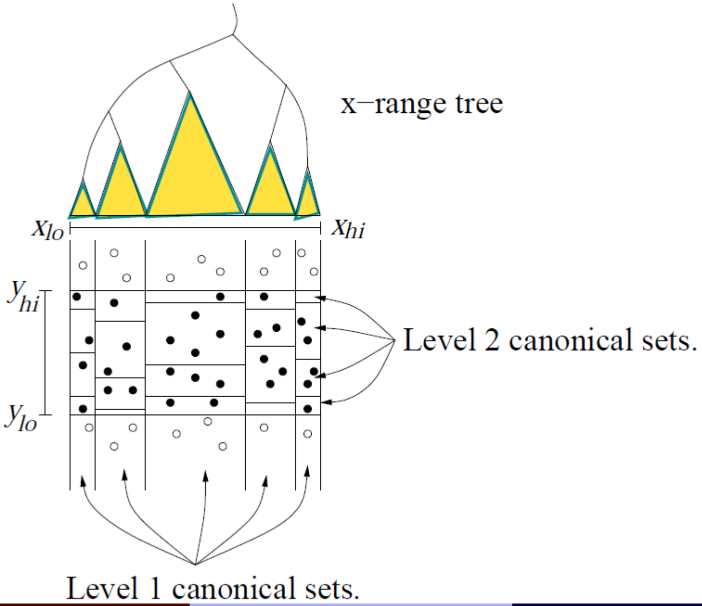


2D Range Tree

- The generic query is $R = [x_{lo}, x_{hi}] \times [y_{lo}, y_{hi}]$.
- We first ignore the y-coordinates, and build a 1D x-range tree.
- Key idea is to collect points of each canonical set, and build a y-range tree on them.
- We search each of the $O(\log n)$ canonical sets that include points for x-range $[x_{lo}, x_{hi}]$ using their y-range trees for range $[y_{lo}, y_{hi}]$.

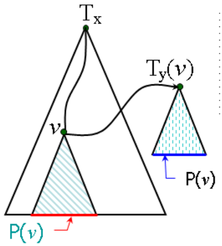


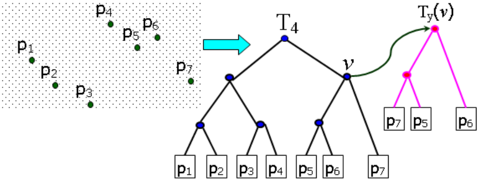
Range queries in 2D



2D Range Tree

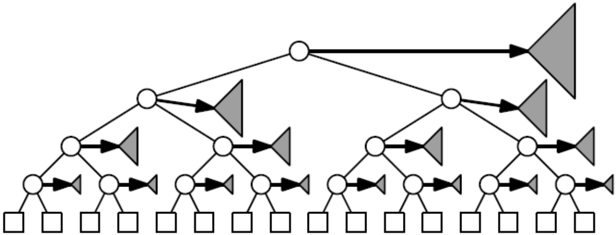
	x	y
p1	1	2.5
p2	2	1
p3	3	0
p4	4	4
p5	4.5	3
p6	5.5	3.5
p7	6.5	2





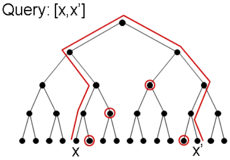
2D Range Tree

Every internal node stores a whole tree in an associated structure, on y-coordinate



Question: How much storage does this take?

- Time complexity for 2D is $O((\log n)^2 + k)$.



At most 2 × height of T = 2 log n

Each 1D query requires O(log n+k') time.

$\Rightarrow$

Query time = $O(\log^2 n + k)$

- Space complexity is $O(nlogn)$.
  1. At each level of the main tree associated structures store all the data points once (with constant overhead): O(n).
  2. There are O(log n) levels.
  3. So, overall space is $O(n \log n)$.

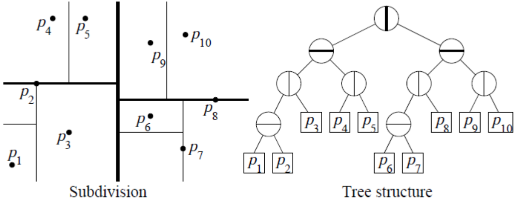
2D Range Tree

Theorem 2

A set of n points in the plane can be preprocessed in $O(n \log n)$ time into a data structure of $O(n \log n)$ size so that any 2D range query can be answered in $O(\log^2 n + k)$ time, where k is the number of answers reported.

In contrast, a *kd*-tree has $O(n)$ size and answers queries in $O(\sqrt{n} + k)$ time.

Kd-Tree



- 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.
- To get balanced trees, use the median coordinate for splitting-median itself can be put in either half.