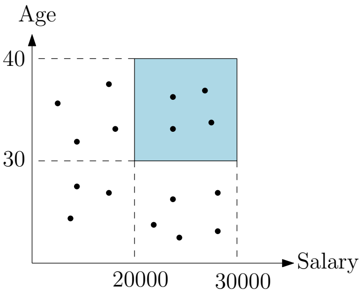
Range searching

Aritra Banik[1]

Assistant Professor
National Institute of Science Education and Research

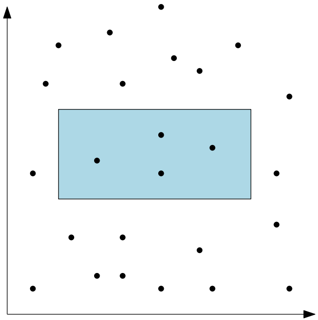
[1]Slide ideas borrowed from Marc van Kreveld and Subhash Suri

Range query



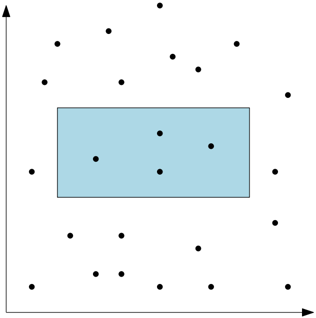
- A range query is a common database operation that retrieves all records where some value is between an upper and lower boundary.
- Range query: Asks for the objects whose coordinates lie in a specified query range (interval)

Range searching

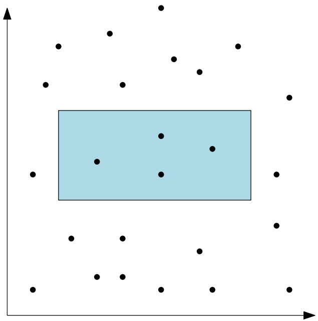


- Range Searching: Process a set of given data points efficiently such that given a range window set of points inside the range can e reported "QUICKLY".
- Time-Space tradeoff: the more we preprocess and store, the faster we can solve a query.
- A (search) data structure has a storage requirement, a query time, and a construction time (and an update time)

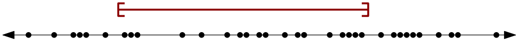
Range searching



- Construction time $O(1)$: query time??
- Objective is sub linear query time.

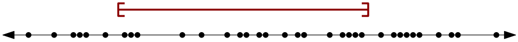


- Construction time $O(1)$: query time??
- Objective is sub linear query time.



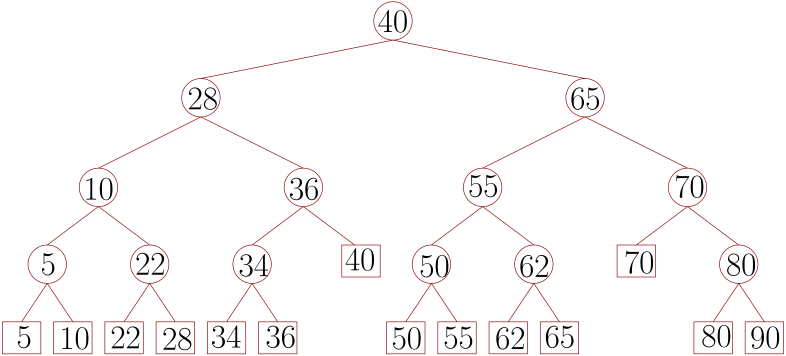
- 1D range query problem: Preprocess a set of n points on the real line such that the ones inside a 1D query range (interval) can be reported fast.
- The points $p_1 \ldots p_n$ are known beforehand, the query $[x, y]$ arises at run time.
- A solution to a query problem is a data structure description, a query algorithm, and a construction algorithm.

1D range query problem



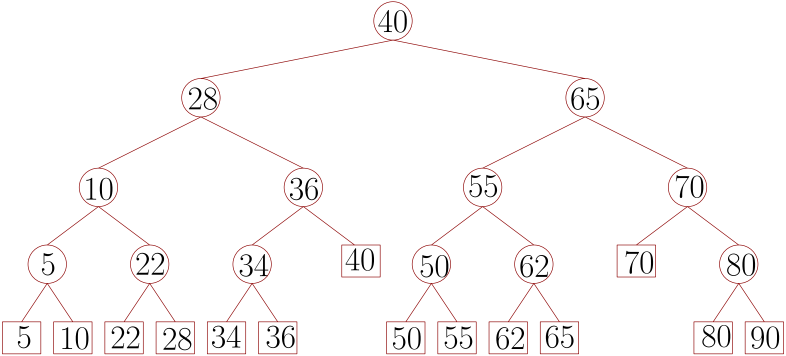
- 1D range query problem: Preprocess a set of n points on the real line such that the ones inside a 1D query range (interval) can be reported fast.
- The points $p_1 \ldots p_n$ are known beforehand, the query $[x, y]$ arises at run time.
- A solution to a query problem is a data structure description, a query algorithm, and a construction algorithm.

The Data Structure: Balanced binary search trees



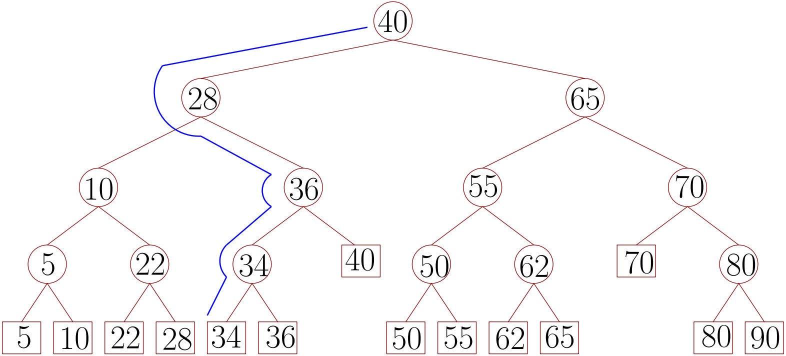
- Query [34, 80]
- Search path for 34.
- Search path for 80.

The Data Structure: Balanced binary search trees



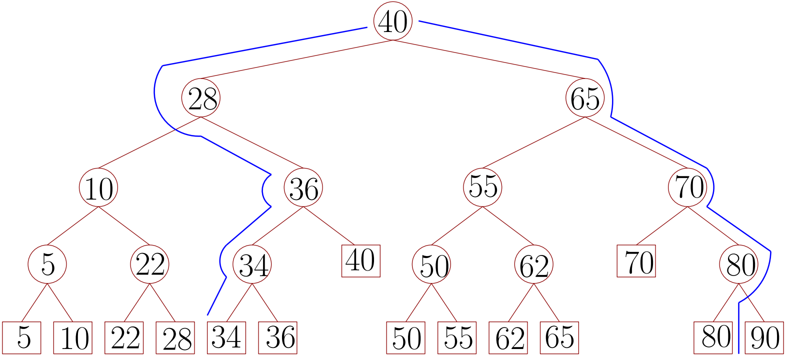
- Query $[34, 80]$
- Search path for 34.
- Search path for 80.

The Data Structure: Balanced binary search trees



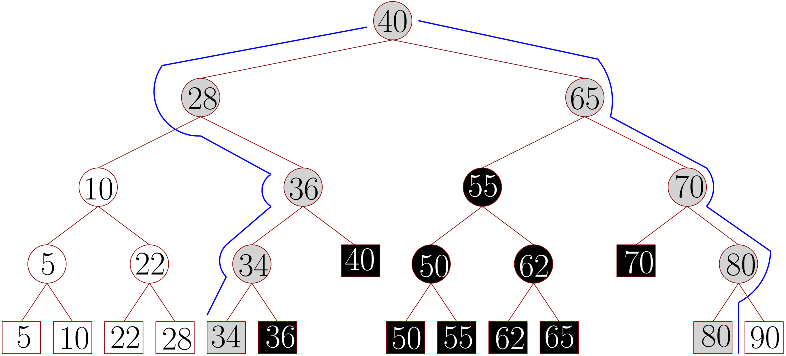
- Query $[34, 80]$
- Search path for 34.
- Search path for 80.

The Data Structure: Balanced binary search trees



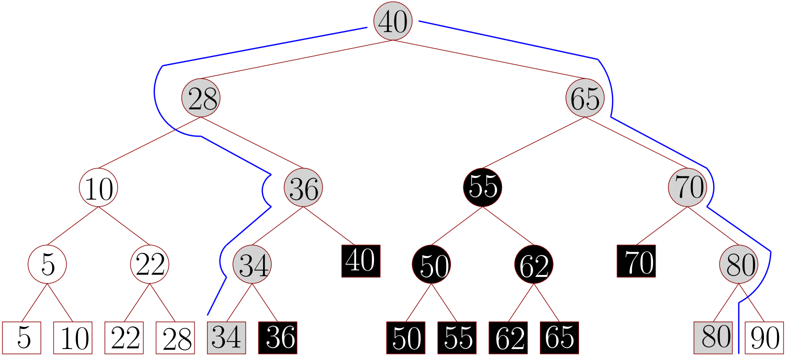
- Query $[34, 80]$
- Search path for 34.
- Search path for 80.

Three types of nodes for a given query:



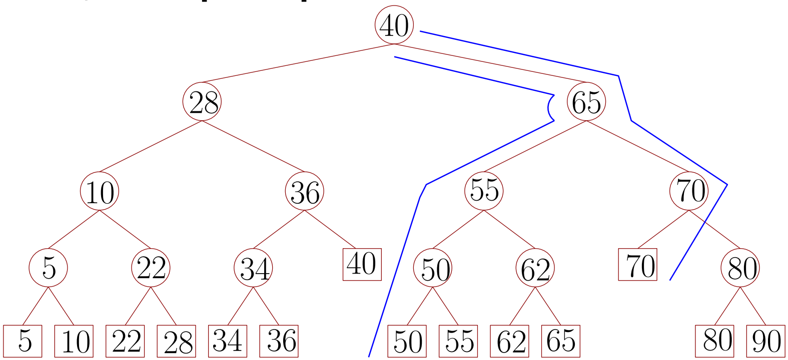
- **White nodes:** never visited by the query
- **Grey nodes:** visited by the query, unclear if they lead to output
- **Black nodes:** Visited by the query, whole subtree is output

Three types of nodes for a given query:



- **White nodes:** never visited by the query
- **Grey nodes:** visited by the query, unclear if they lead to output
- **Black nodes:** Visited by the query, whole subtree is output

Find a split node [50 – 70]

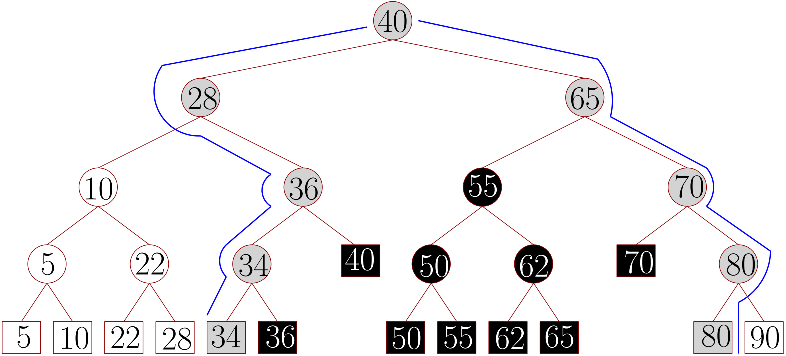


Algorithm

Algorithm 1 1DRangeQuery($T, [x : y]$)

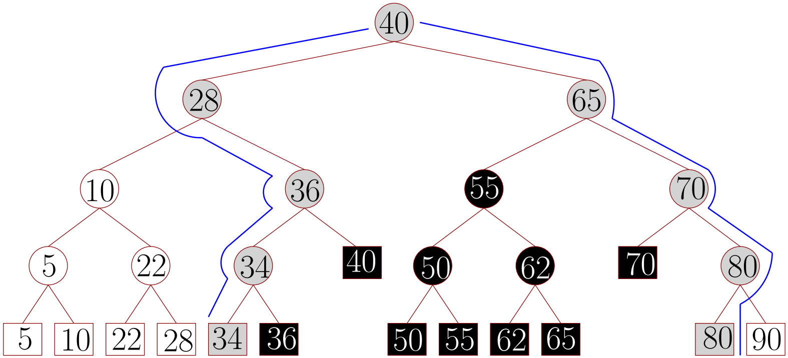
```
v_split ← FindSplitNode(T, x, y)
if v_split is a leaf then
    Check if the point in v_split must be reported.
else
    v ← lc(v_split)
    while v v is not a leaf do
        if x ≤ value(v) then
            ReportSubtree(rc(v))
            v ← lc(v)
        else
            v ← rc(v)
        end if
    end while
    v ← rc(v_split)
    Similarly, follow the path to y
end if
```

Runtime



- White nodes: never visited by the query; no time spent
- Grey nodes: visited by the query, unclear if they lead to output; time determines dependency on n
- Black nodes: visited by the query, whole subtree is output; time determines dependency on k, the output size

Runtime



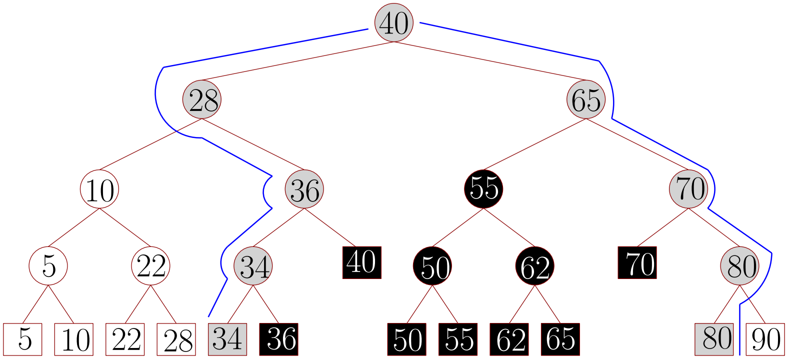
- Grey nodes: they occur on only two paths in the tree, and since the tree is balanced, its depth is $O(\log n)$
- Black nodes: Charged on output

The time spent at each node is $O(1) \Rightarrow O(\log n + k)$ query 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)

A 1-dimensional range tree for range counting queries

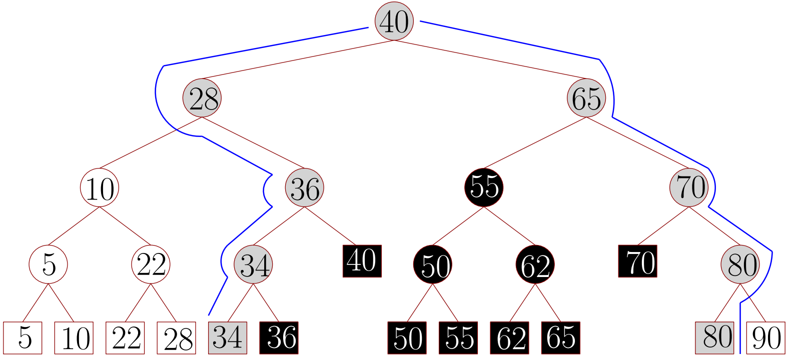


Theorem

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 range counting queries can be answered in $O(\log n)$ time

A 1-dimensional range tree for range counting queries

A 1-dimensional range tree for range counting queries

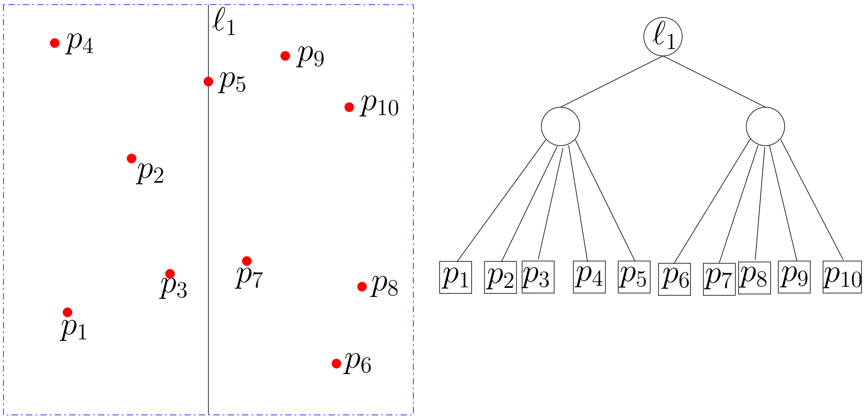


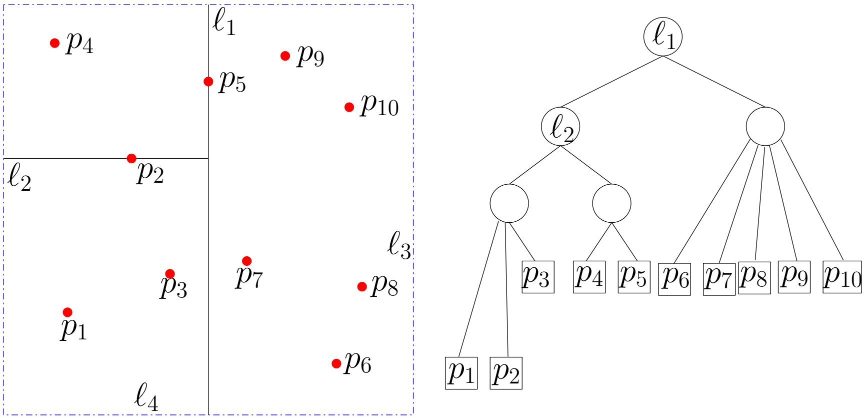
Theorem

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 range counting queries can be answered in* $O(\log n)$ *time*

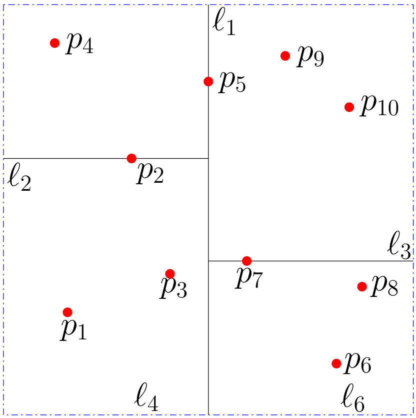
Kd-trees, the idea:

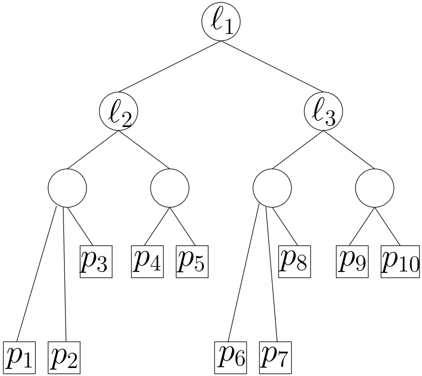
- Split the point set alternatingly by x-coordinate and by y-coordinate
- Split by x-coordinate: split by a vertical line that has half the points left and half right
- Split by y-coordinate: split by a horizontal line that has half the points below and half above

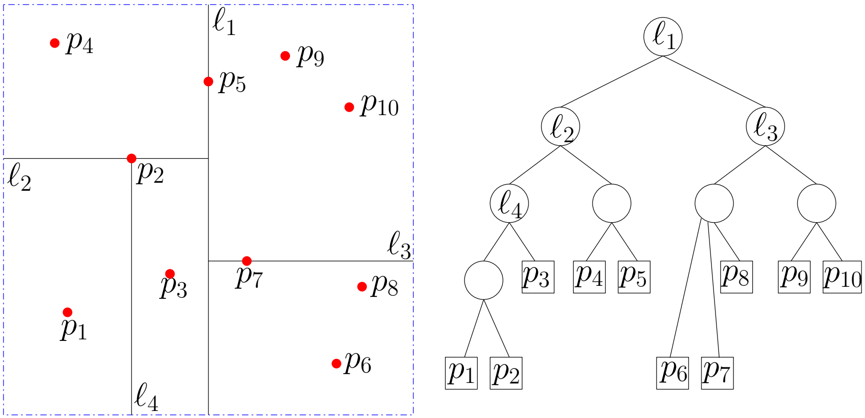

ℓ_1
p_4
p_9
p_5
p_{10}
p_2
p_7
p_3
p_8
p_1
p_6
ℓ_1
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10}


ℓ_1
p_4
p_9
p_5
p_{10}
ℓ_2
p_2
ℓ_3
p_7
p_3
p_8
p_1
p_6
ℓ_4
ℓ_1
ℓ_2
p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10}
p_1 p_2

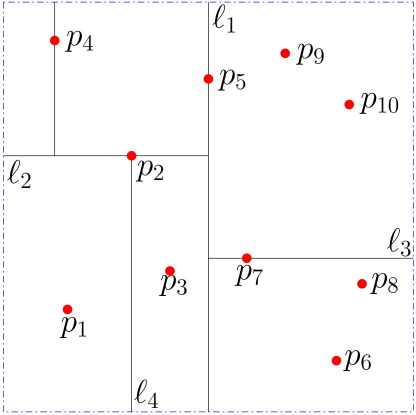
Kd-tree Construction

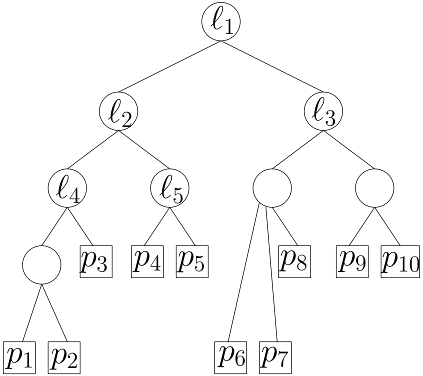


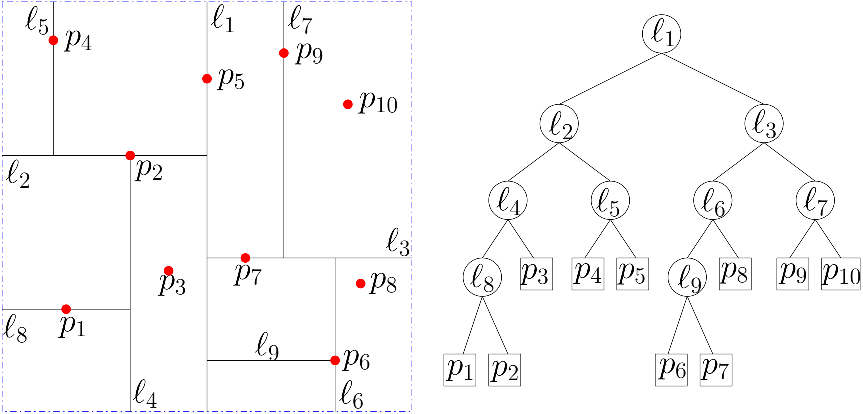



ℓ1
ℓ2
ℓ3
ℓ4
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10

Kd-tree Construction






ℓ5
p4
ℓ1
ℓ7
p9
p5
p10
ℓ2
p2
ℓ3
p7
p3
p8
ℓ8
p1
ℓ9
p6
ℓ4
ℓ6
ℓ1
ℓ2
ℓ3
ℓ4
ℓ5
ℓ6
ℓ7
ℓ8
p3
p4
p5
ℓ9
p8
p9
p10
p1
p2
p6
p7

Algorithm

Algorithm 2 BuildKdTree(P, $depth$)

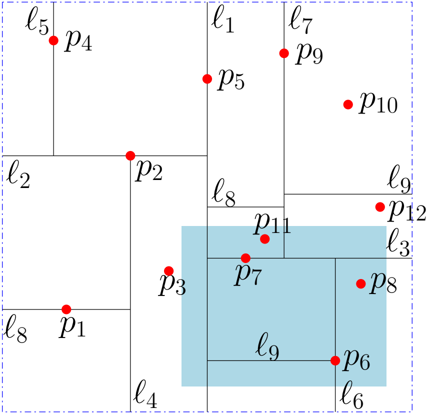
```
if P contains only one point then
    return a leaf storing this point
else if depth is even then
    Split P with a vertical line ℓ through the median x-coordinate
    into P_1 (left of ℓ) and P_2 (right of ℓ)
else
    Split P with a horizontal line ℓ through the median x-
    coordinate into P_1 (below ℓ) and P_2 (above ℓ)
end if
left ← BuildKdTree(P_1, depth + 1)
right ← BuildKdTree(P_2, depth + 1)
Create a node v storing ℓ, make left left the left child of v, and
make right right the right child of v.
return(v)
```

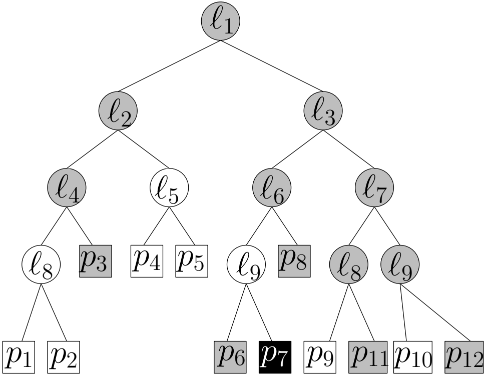
Complexity

- The median of a set of n values can be computed in $O(n)$ time
- Let $T(n)$ be the time needed to build a kd-tree on n points

$T(1) = O(1)$
$T(n) = 2T(n/2) + O(n)$
A kd-tree can be built in $O(n \log n)$ time

Kd-tree querying





Complexity Analysis

White, grey, and black nodes with respect to region(v):

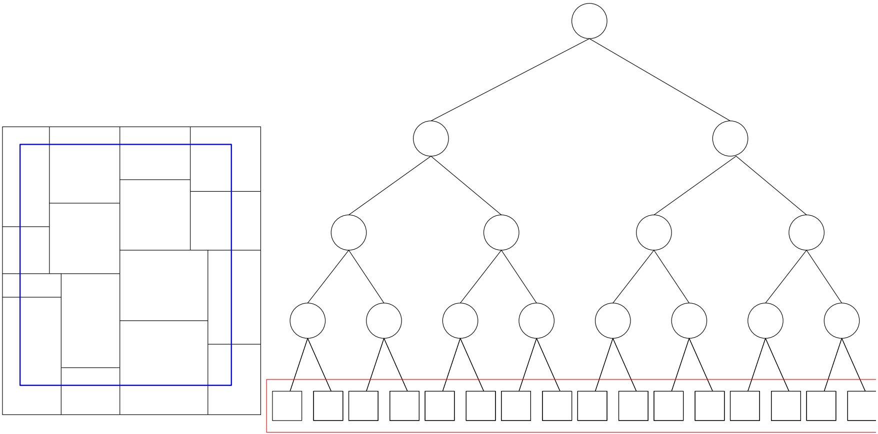
- White node v: R does not intersect region(v)
- Grey node v: R intersects region(v), but region(v) $\not\subseteq R$
- Black node v: region(v)$\subseteq R$

Complexity Analysis

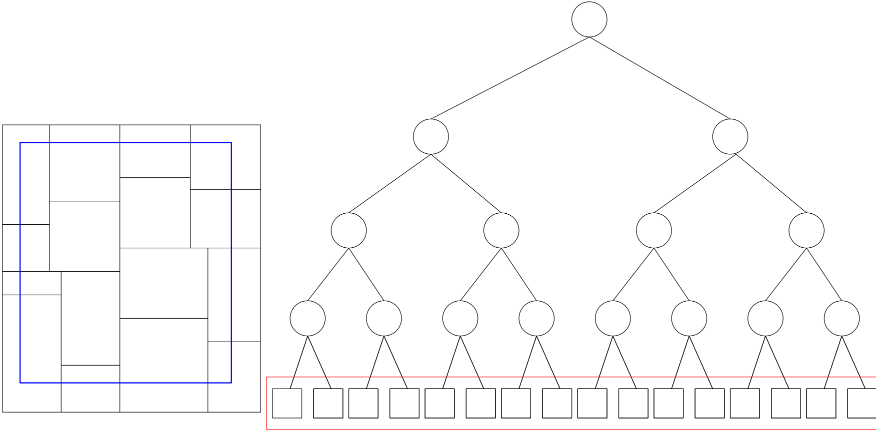
- White node v: R does not intersect region(v) Not visiting
- Grey node v: R intersects region(v), but region(v) $\not\subseteq R$
- Black node v: region(v)$\subseteq R$ Charged on the output size

Complexity Analysis

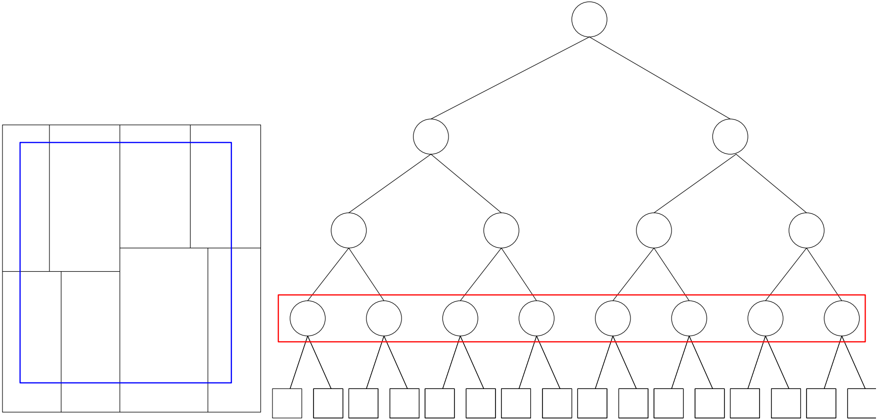
- White node v: R does not intersect region(v) Not visiting
- Grey node v: R intersects region(v), but region(v) $\not\subseteq R$
- Black node v: region(v)$\subseteq R$ Charged on the output size



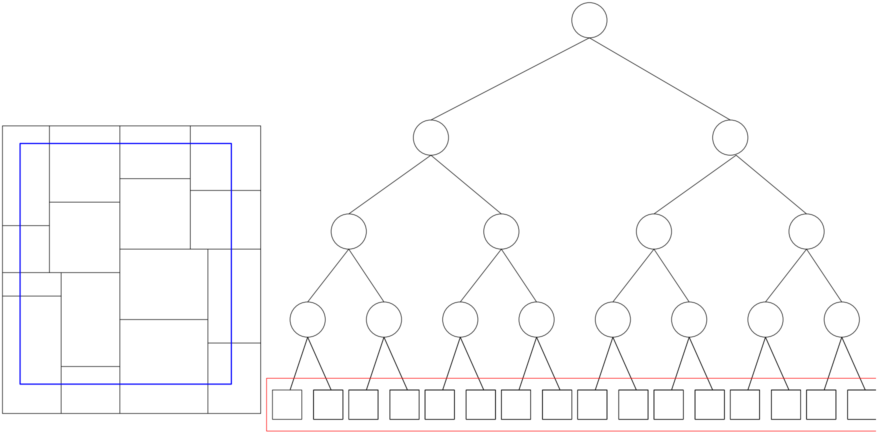
- How many grey nodes can be there among the leaf nodes.
- How many regions can be intersected by a axis parallel straight line.



- At max $O(\sqrt(n))$
- In the previous level $O(\sqrt(n/2))$

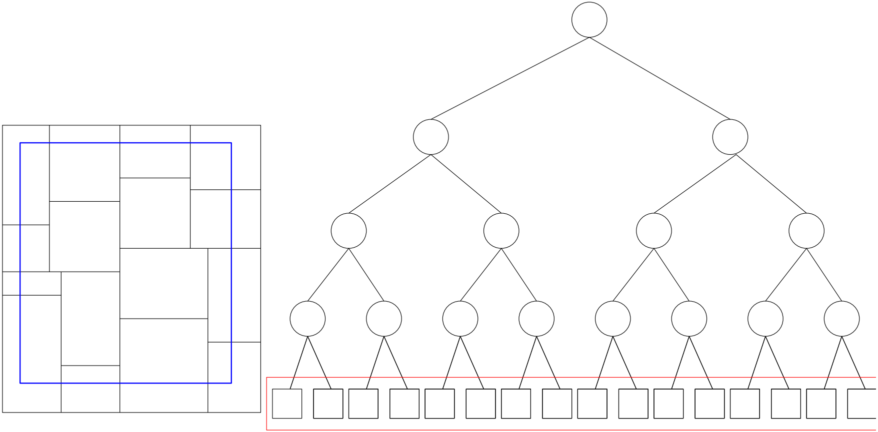


- At max $O(\sqrt(n))$
- In the previous level $O(\sqrt(n/2))$



- Total no of Gray cells are $\sqrt{n}(1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{4}} + \frac{1}{\sqrt{8}} \ldots)$
- $O(\sqrt(n))$

Complexity Analysis



- Total no of Gray cells are $\sqrt{n}(1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{4}} + \frac{1}{\sqrt{8}} \ldots)$
- $O(\sqrt{(n)})$

Higher dimensions

- A 3-dimensional kd-tree alternates splits on x, y, and z coordinate
- A 3D range query is performed with a box

Theorem

A set of n points in d-space can be preprocessed in $O(n \log n)$ time into a data structure of $O(n)$ size so that any d-dimensional range query can be answered in $O(n^{1-1/d} + k)$ time, where k is the number of answers reported.

Higher dimensions

- A 3-dimensional kd-tree alternates splits on x, y, and z coordinate
- A 3D range query is performed with a box

Theorem

A set of n points in d-space can be preprocessed in $O(n \log n)$ time into a data structure of $O(n)$ size so that any d-dimensional range query can be answered in $O(n^{1-1/d} + k)$ time, where k is the number of answers reported.