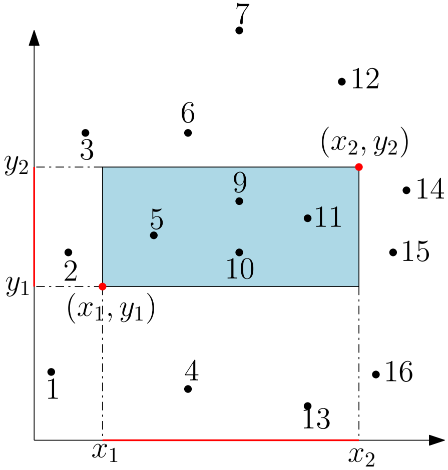
Range Trees

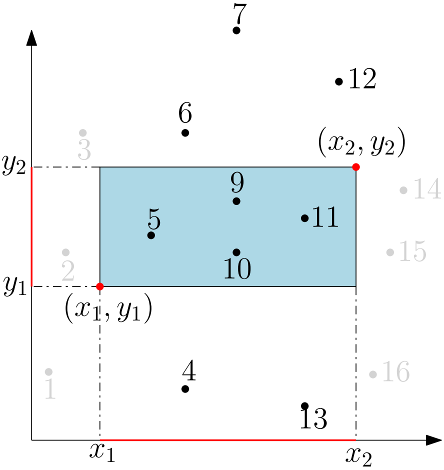
Aritra Banik*
National Institute of Science Education and Research

Summer School on Graph Theory and Graph Algorithm at NIT Calicut

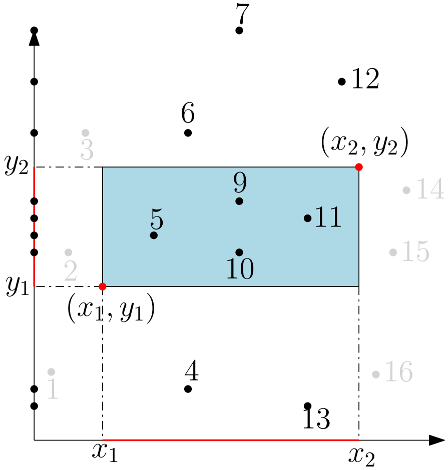
*Slide ideas borrowed from Marc van Kreveld and Subhash Suri


7
12
6
3
(x_2, y_2)
y_2
9
14
5
11
2
15
10
y_1
(x_1, y_1)
4
16
1
13
x_1
x_2

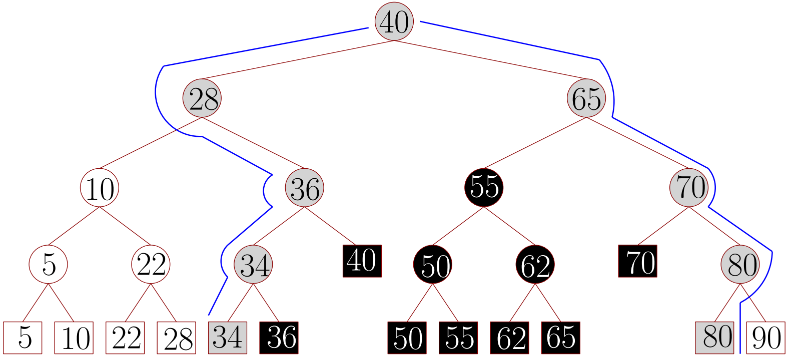
2-dimensional Range Searching



2-dimensional Range Searching

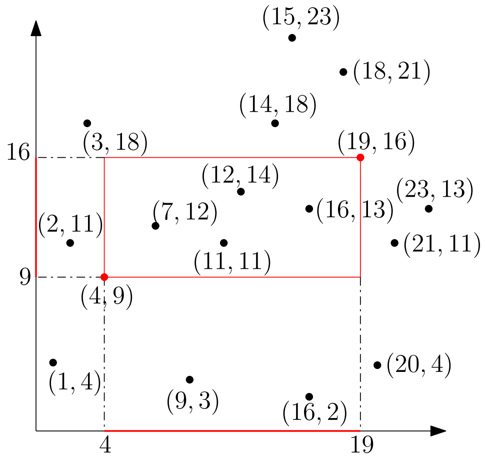


A 1-dimensional range query with $[34, 80]$

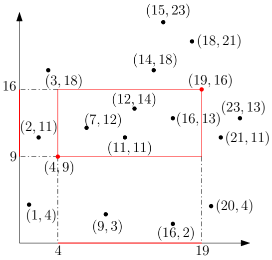


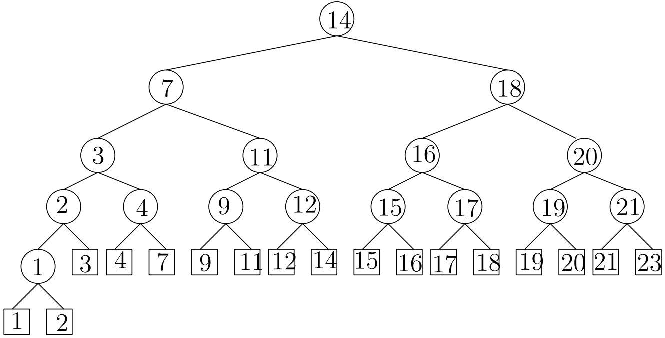
- **White nodes:** never visited by the query
- **Grey nodes:** visited by the query, unclear if they lead to output
- **Black nodes:** Visited by the query, whole subtree is output

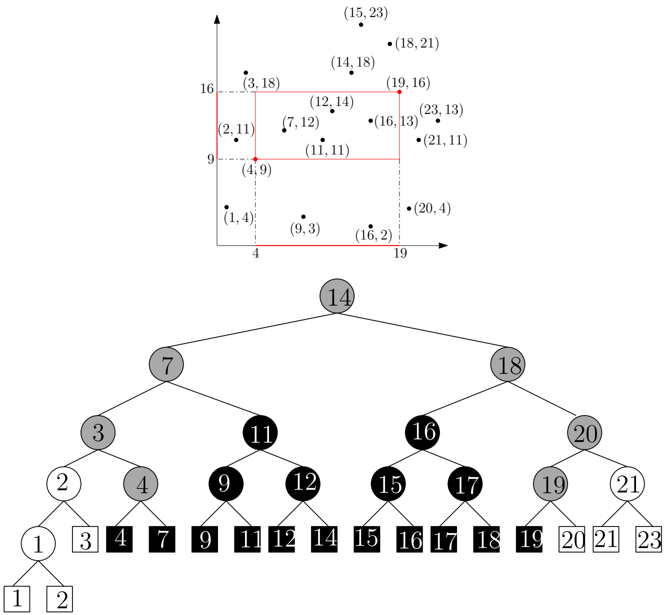
2-dimensional Range Searching



2-dimensional Range Searching

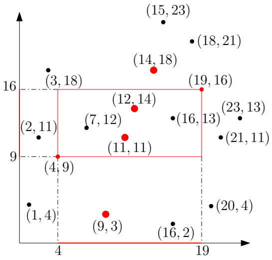


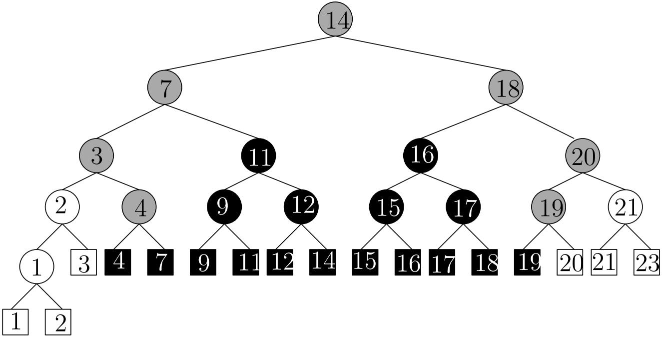




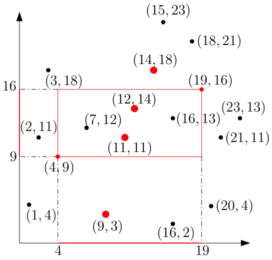
(15, 23)
(18, 21)
(14, 18)
(3, 18)
(19, 16)
16
(12, 14)
(23, 13)
(2, 11)
(7, 12)
(16, 13)
(21, 11)
(11, 11)
9
(4, 9)
(1, 4)
(20, 4)
(9, 3)
(16, 2)
4
19
14
7
18
3
11
16
20
2
4
9
12
15
17
19
21
1
3
4
7
9
11
12
14
15
16
17
18
19
20
21
23
1
2

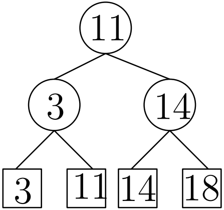
2-dimensional Range Searching



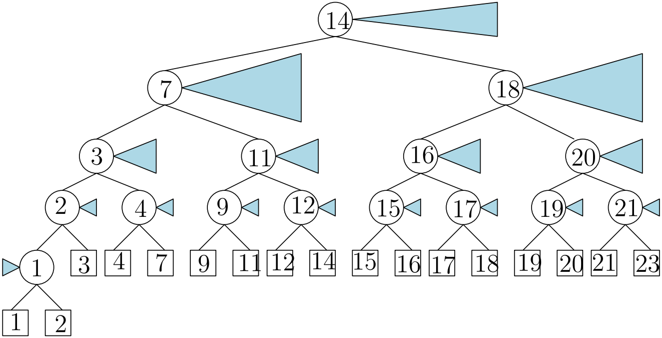


2-dimensional Range Searching





2-dimensional Range Searching



Storage of 2D range trees

- By level: On each level, any point is stored exactly once. So all associated trees on one level together have $O(n)$ size
- By point: For any point, it is stored in the associated structures of its search path. So it is stored in $O(\log n)$ of them

Construction algorithm

Algorithm 1 Build2DRangeTree(P)

```
Construct the associated structure: Build a binary search tree T_y with
  the y-coordinates in P
if P contains only one point then
    Create a leaf v storing this point, and make T assoc the associated
    structure of v.
else
    Split P into P_left and P_right , WRT the median x-coordinate, x_mid
    v_left=Build2DRangeTree(P_left)
    v_right=Build2DRangeTree(P_right)
    Create a node v storing x_mid, make v_left the left child of v, make v_right
    the right child of v, and make T_y the associated structure of v
    return v
end if
```

Efficiency of construction

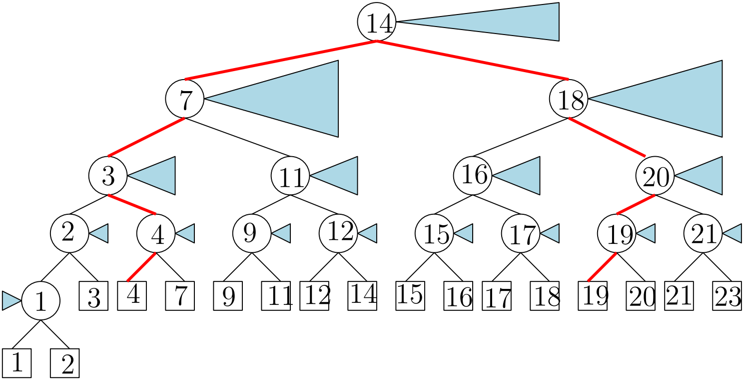
- $T(n) = 2 \cdot T(n/2) + O(n \log n)$
- The construction algorithm takes $O(n \log^2 n)$ time

Efficiency of construction

- $T(n) = 2 \cdot T(n/2) + O(n \log n)$
- The construction algorithm takes $O(n \log^2 n)$ time

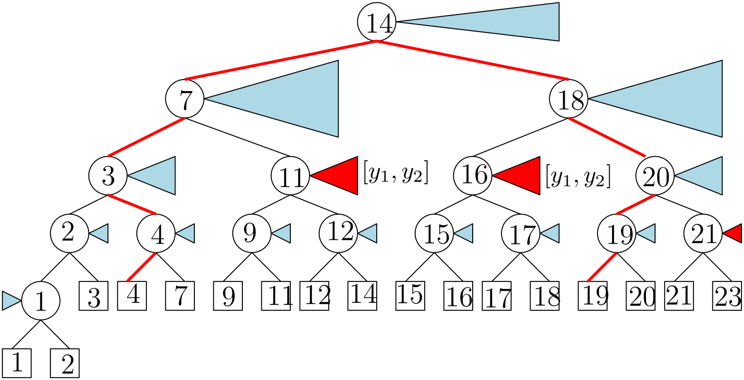
Query algorithm

- Query $[4 : 19] \times [y_1 : y_2]$



Query algorithm

- Query $[4 : 19] \times [y_1 : y_2]$



Query algorithm

Algorithm 2 2DRangeQuery($T, [x_1 : x_2] \times [y_1 : y_2]$)

```
v_split ← FindSplitNode(T, x1, x2)
if v_split is a leaf then
    Check if the point in v_split must be reported.
else
    v ← lc(v_split)
    while  v is not a leaf do
        if x1 ≤ value(v) then
            1DRangeQuery(T_y(rc(v)), [y1 : y2])
            v ← lc(v)
        else
            v ← rc(v)
        end if
    end whileCheck if the point in v must be reported.
    Similarly, follow the path from rc(v_split) to x2
end if
```

2D range query efficiency

- We search in $O(\log n)$ associated structures to perform a 1D range query; at most two per level of the main tree
- The query time is $O(\log^2 n + k)$, where k is the size of the output

Theorem

A set of n points in the plane can be preprocessed in $O(n \log^2 n)$ 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.

Recall that a kd-tree has $O(n)$ size and answers queries in $O(\sqrt{n} + k)$ time

2D range query efficiency

- We search in $O(\log n)$ associated structures to perform a 1D range query; at most two per level of the main tree
- The query time is $O(\log^2 n + k)$, where k is the size of the output

Theorem

A set of n points in the plane can be preprocessed in $O(n \log^2 n)$ 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.

Recall that a kd-tree has $O(n)$ size and answers queries in $O(\sqrt{n} + k)$ time

Higher dimensional range trees

Theorem

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

Recall that a kd-tree has $O(n)$ size and answers queries in $O(n^{1-1/d} + k)$ time

- Can we do better?
- We can improve the query time of a 2D range tree from $O(\log^2 n)$ to $O(\log n)$ by a technique called fractional cascading.
- The idea illustrated best by a different query problem: Suppose that we have a collection of sets $S_1 \ldots S_m$, where $|S_1| = n$ and where $S_{i+1} \subset S_i$
- We want a data structure that can report for a query number x, the smallest value greater than equals to x in all sets $S_1, \ldots, S_m$

Range searching

- Can we do better?
- We can improve the query time of a 2D range tree from $O(\log^2 n)$ to $O(\log n)$ by a technique called fractional cascading.
- The idea illustrated best by a different query problem: Suppose that we have a collection of sets $S_1 \dots S_m$, where $|S_1| = n$ and where $S_{i+1} \subset S_i$
- We want a data structure that can report for a query number x, the smallest value greater than equals to x in all sets $S_1, \dots, S_m$

Range searching

- Can we do better?
- We can improve the query time of a 2D range tree from $O(\log^2 n)$ to $O(\log n)$ by a technique called fractional cascading.
- The idea illustrated best by a different query problem: Suppose that we have a collection of sets $S_1 \dots S_m$, where $|S_1| = n$ and where $S_{i+1} \subset S_i$
- We want a data structure that can report for a query number x, the smallest value greater than equals to x in all sets $S_1, \ldots, S_m$

Fractional Cascading

- The idea illustrated best by a different query problem: Suppose that we have a collection of sets $S_1 \ldots S_m$, where $|S_1| = n$ and where $S_{i+1} \subset S_i$
- We want a data structure that can report for a query number x, the smallest value greater than equals to x in all sets $S_1, \ldots, S_m$.
- Say $x = 5$

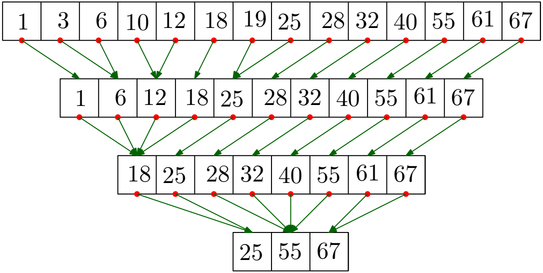
1	3	6	10	12	18	19	25	28	32	40	55	61	67

1	6	12	18	25	28	32	40	55	61	67

18	25	28	32	40	55	61	67

25	55	67

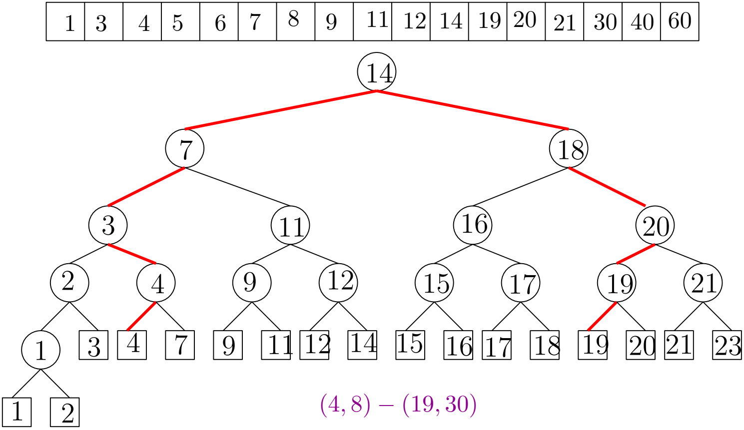
- The idea illustrated best by a different query problem: Suppose that we have a collection of sets $S_1 \ldots S_m$, where $|S_1| = n$ and where $S_{i+1} \subset S_i$
- We want a data structure that can report for a query number x, the smallest value greater than equals to x in all sets $S_1, ..., S_m$.
- Say $x = 5$



Fractional Cascading

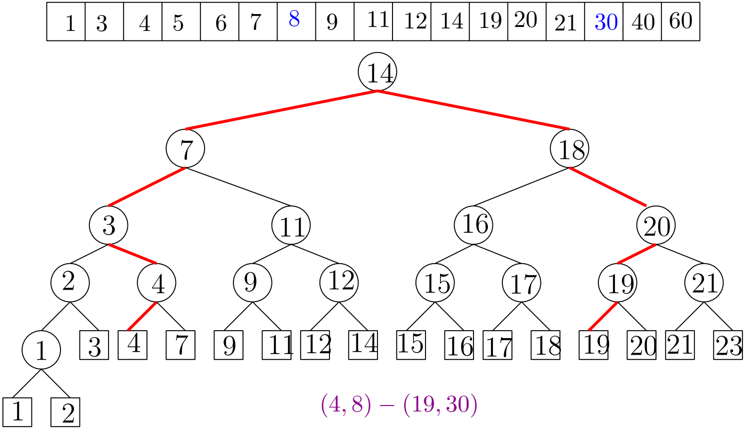
- Now we do "the same" on the associated structures of a 2-dimensional range tree
- Note that in every associated structure, we search with the same values y_1 and y_2.
- Replace all associated structures (y trees) with sorted lists
- For every list element (and leaf of the associated structure of the root), store two pointers to the appropriate list elements in the lists of the left child and of the right child

- Query $(4, 8) - (19, 30)$



$(1, 90), (2, 30), (3, 8), (4, 12), (7, 14), (9, 7), (11, 20), (12, 19), (14, 21),$
$(15, 4), (16, 1), (17, 6), (18, 5), (19, 60), (20, 3), (21, 11), (23, 40)$

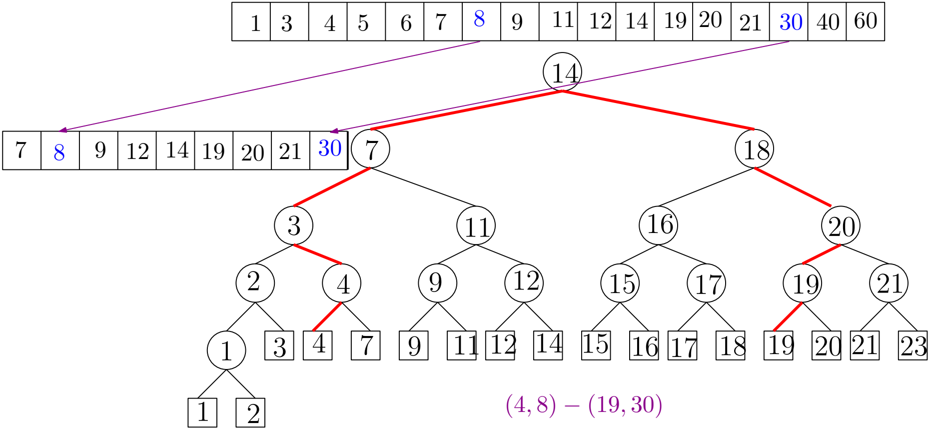
- Query $(4, 8) - (19, 30)$



$(1, 90), (2, 30), (3, 8), (4, 12), (7, 14), (9, 7), (11, 20), (12, 19), (14, 21),$
$(15, 4), (16, 1), (17, 6), (18, 5), (19, 60), (20, 3), (21, 11), (23, 40)$

Fractional Cascading

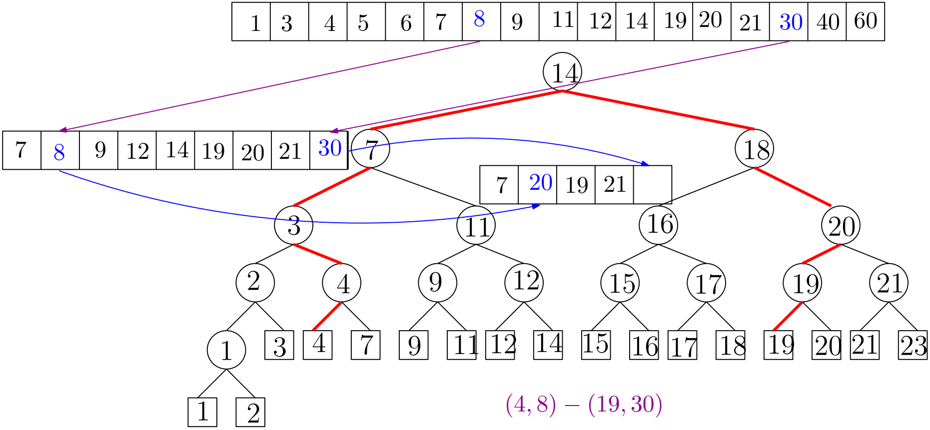
- Query $(4, 8) - (19, 30)$



$(4, 8) - (19, 30)$

$(1, 90), (2, 30), (3, 8), (4, 12), (7, 14), (9, 7), (11, 20), (12, 19), (14, 21),$
$(15, 4), (16, 1), (17, 6), (18, 5), (19, 60), (20, 3), (21, 11), (23, 40)$

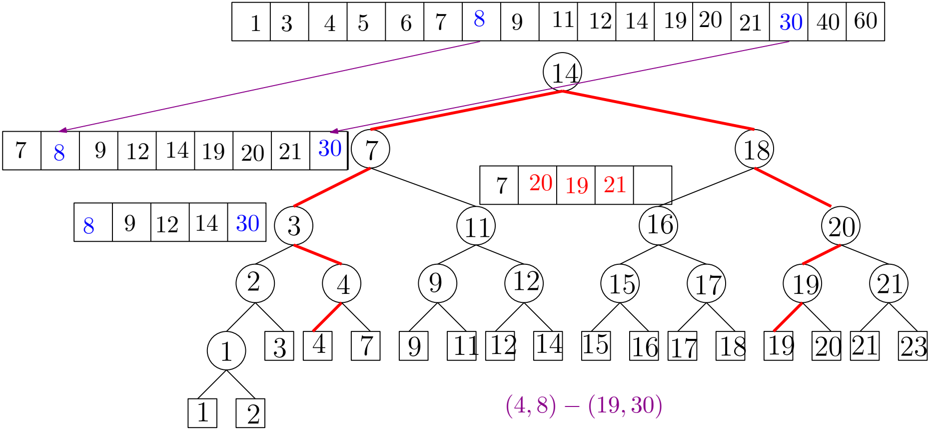
- Query $(4, 8) - (19, 30)$



$(1, 90), (2, 30), (3, 8), (4, 12), (7, 14), (9, 7), (11, 20), (12, 19), (14, 21),$
$(15, 4), (16, 1), (17, 6), (18, 5), (19, 60), (20, 3), (21, 11), (23, 40)$

Fractional Cascading

- Query $(4, 8) - (19, 30)$



$(1, 90), (2, 30), (3, 8), (4, 12), (7, 14), (9, 7), (11, 20), (12, 19), (14, 21),$
$(15, 4), (16, 1), (17, 6), (18, 5), (19, 60), (20, 3), (21, 11), (23, 40)$

Higher dimensional range trees

Theorem

A set of n points in in d-dimensional space can be preprocessed in $O(n \log^d n)$ time into a data structure of $O(n \log^d n)$ size so that any 2D range query can be answered in $O(\log^{d-1} n + k)$ time, where k is the number of answers reported.