

Range Trees

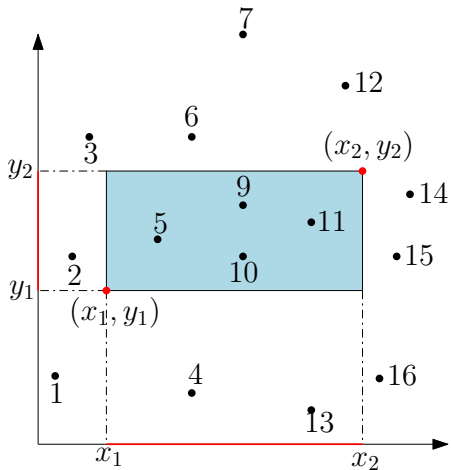
Aritra Banik¹

Assistant Professor
National Institute of Science Education and Research

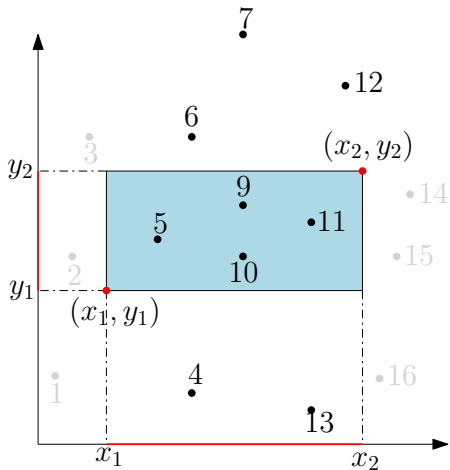


¹Slide ideas borrowed from Marc van Kreveld and Subhash Suri

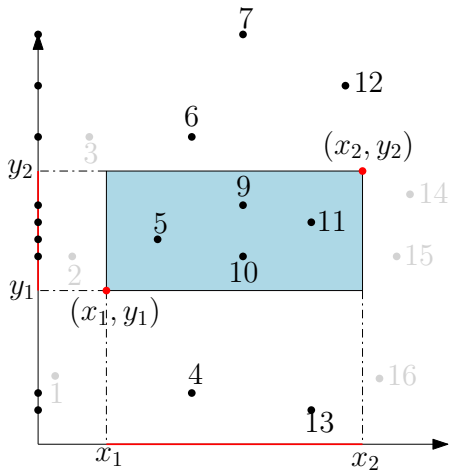
2-dimensional Range Searching



2-dimensional Range Searching

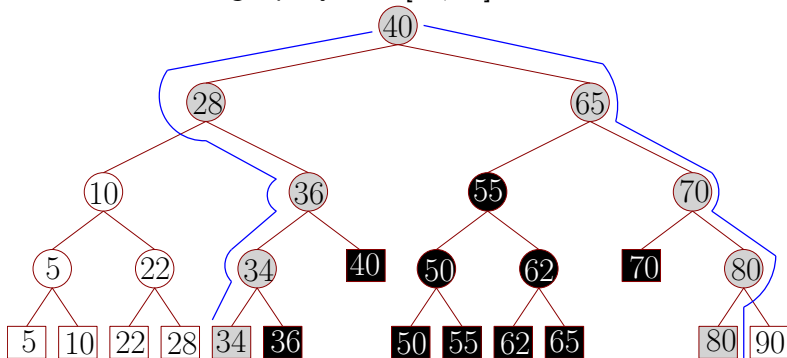


2-dimensional Range Searching



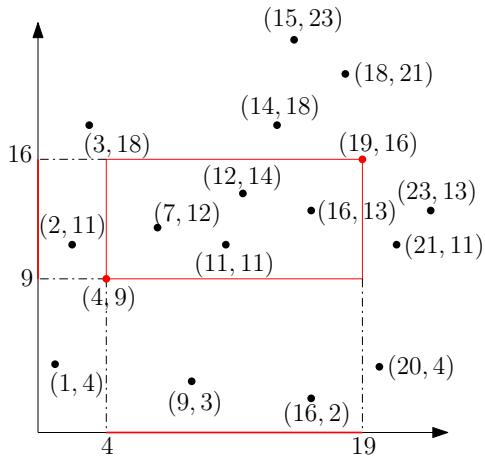
1-dimensional Range Tree

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

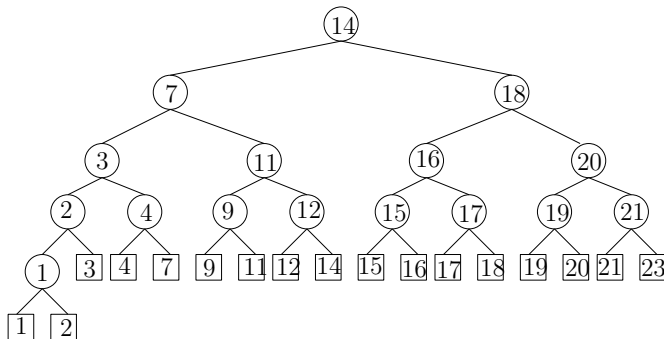
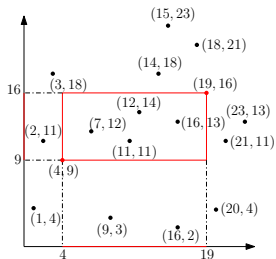


- **White nodes:** never visited by the query
- **Grey nodes:** visited by the query, unclear if they lead to output
- **Black nodes:** Visited by the query, whole subtree is output

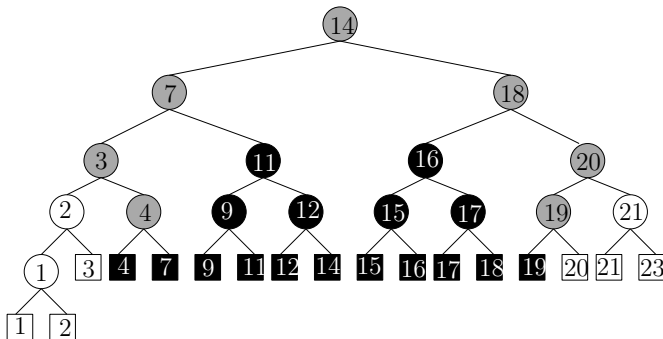
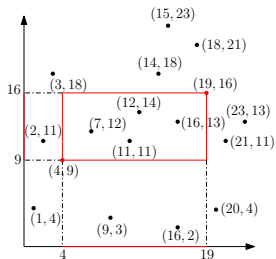
2-dimensional Range Searching



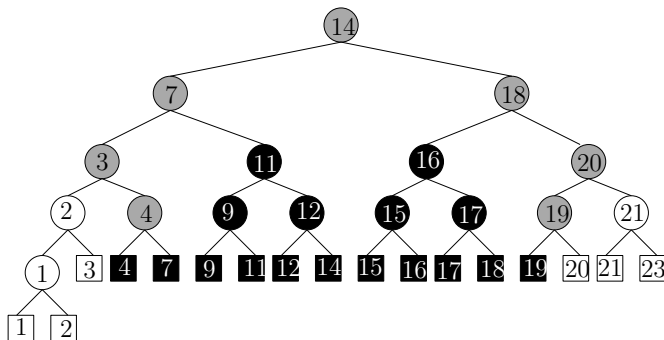
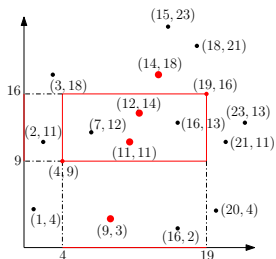
2-dimensional Range Searching



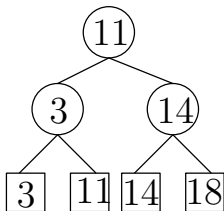
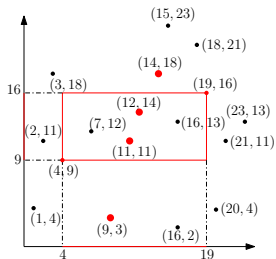
2-dimensional Range Searching



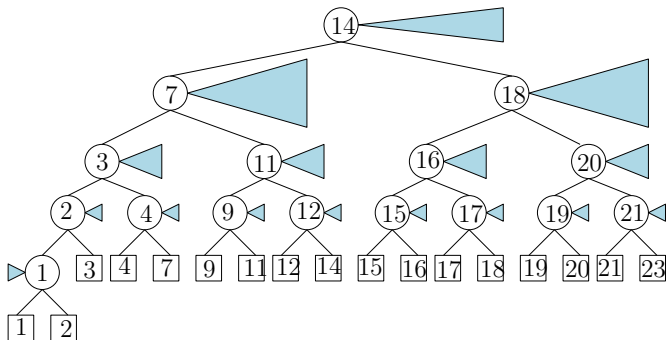
2-dimensional Range Searching



2-dimensional Range Searching



2-dimensional Range Searching



- 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

Algorithm 1 Build2DRangeTree(P)

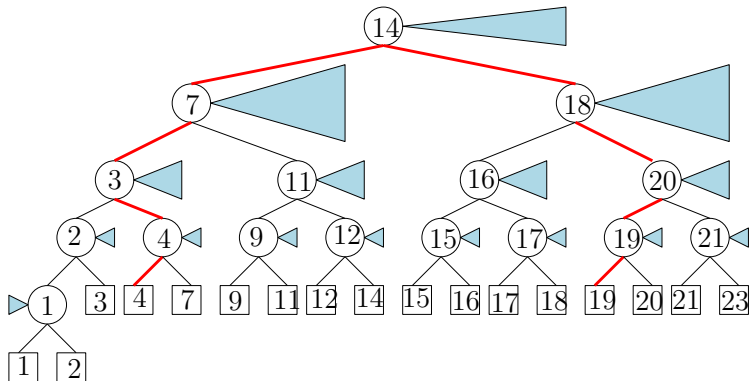
- 1: Construct the associated structure: Build a binary search tree T_y with the y-coordinates in P
 - 2: **if** P contains only one point **then**
 - 3: Create a leaf v storing this point, and make T assoc the associated structure of v .
 - 4: **else**
 - 5: Split P into P_{left} and P_{right} , WRT the median x-coordinate, x_{mid}
 - 6: v_{left} = Build2DRangeTree(P_{left})
 - 7: v_{right} = Build2DRangeTree(P_{right})
 - 8: 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
 - 9: return v
 - 10: **end if**
-

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

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

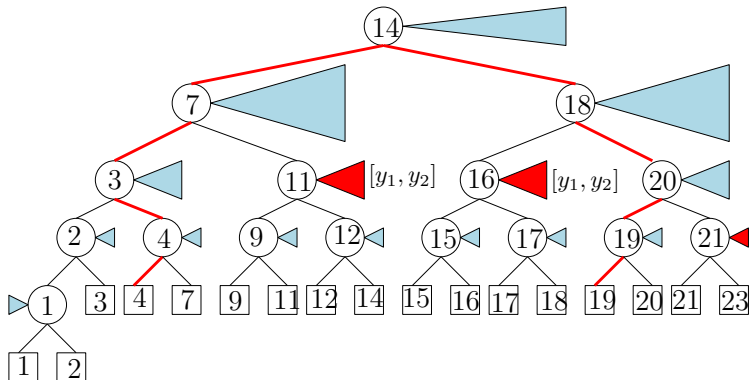
Query algorithm

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



Query algorithm

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



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

```
1:  $v_{split} \leftarrow \text{FindSplitNode}(T, x_1, x_2)$ 
2: if  $v_{split}$  is a leaf then
3:   Check if the point in  $v_{split}$  must be reported.
4: else
5:    $v \leftarrow lc(v_{split})$ 
6:   while  $v$  is not a leaf do
7:     if  $x_1 \leq \text{value}(v)$  then
8:       1DRangeQuery( $T_y(rc(v)), [y_1 : y_2]$ )
9:        $v \leftarrow lc(v)$ 
10:    else
11:       $v \leftarrow rc(v)$ 
12:    end if
13:  end while Check if the point in  $v$  must be reported.
14:  Similarly, follow the path from  $rc(v_{split})$  to  $x_2$ 
15: 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

- 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

Theorem

A set of n points 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

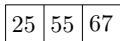
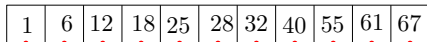
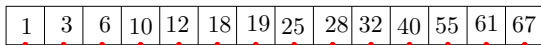
- Can we do better?
- We can improve the query time of a 2D range tree from $O(\log^2 n)$ to $O(\log n)$ by a technique called **fractional cascading**.
- The idea illustrated best by a different query problem:
Suppose that we have a collection of sets $S_1 \dots S_m$, where $|S_1| = n$ and where $S_{i+1}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

- 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

- 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

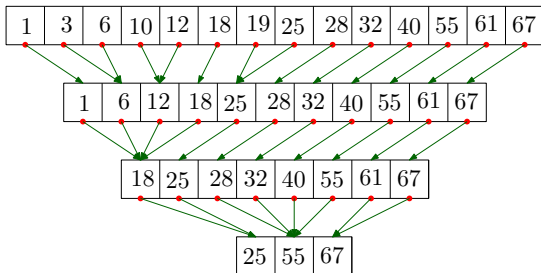
Fractional Cascading

- 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 .
- Say $x = 5$



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 .
- Say $x = 5$

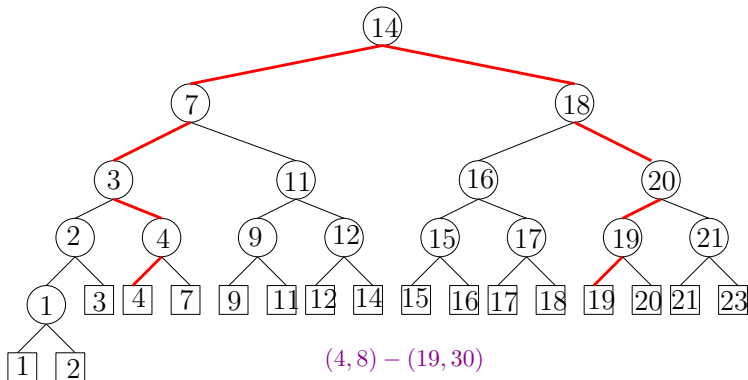


- Now we do "the same" on the associated structures of a 2-dimensional range tree
- Note that in every associated structure, we search with the same values y_1 and y_2 .
- Replace all associated structures (y trees) with sorted lists
- For every list element (and leaf of the associated structure of the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

Fractional Cascading

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

1	3	4	5	6	7	8	9	11	12	14	19	20	21	30	40	60
---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----

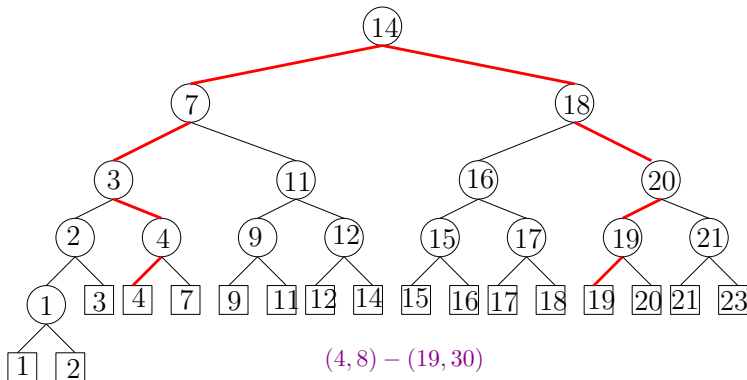


$(1, 90), (2, 30), (3, 8), (4, 12), (7, 14), (9, 7), (11, 20), (12, 19), (14, 21),$
 $(15, 4), (16, 1), (17, 6), (18, 5), (19, 60), (20, 3), (21, 11), (23, 40)$

Fractional Cascading

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

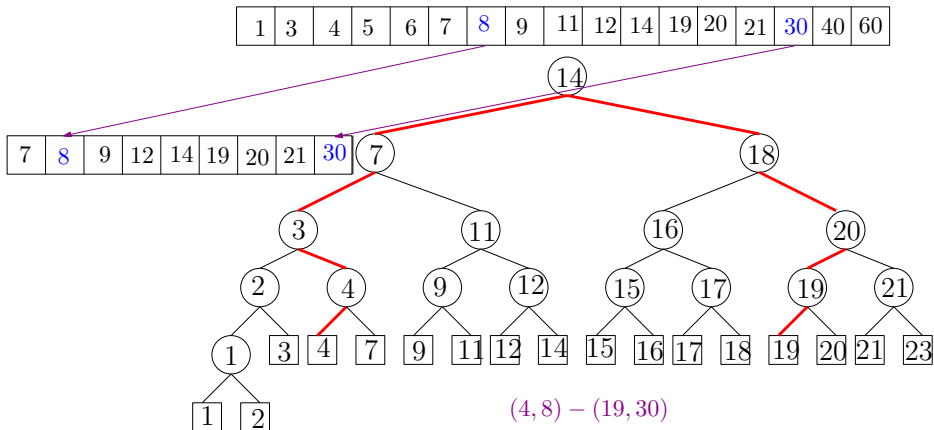
1	3	4	5	6	7	8	9	11	12	14	19	20	21	30	40	60
---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----



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

Fractional Cascading

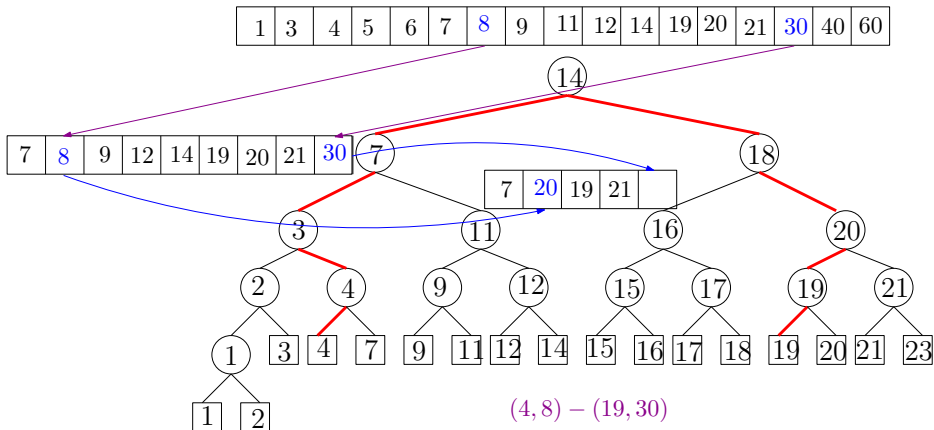
- Query $(4, 8) - (19, 30)$



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

Fractional Cascading

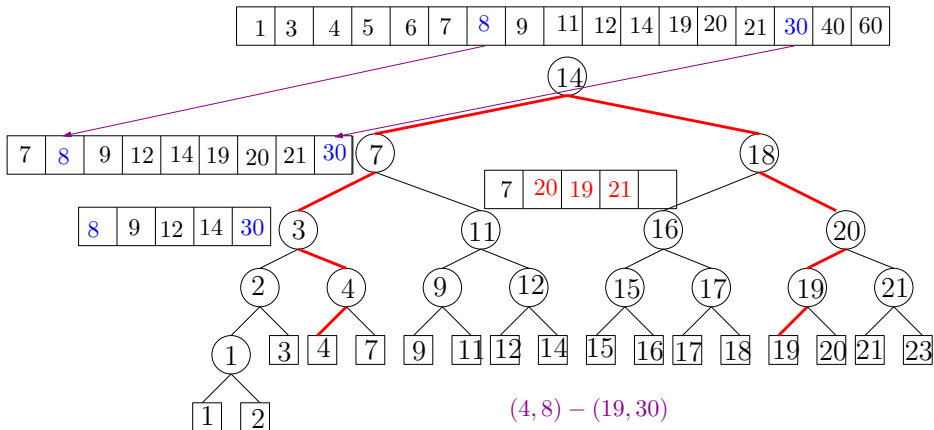
- Query $(4, 8) - (19, 30)$



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

Fractional Cascading

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



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

Theorem

A set of n points 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.