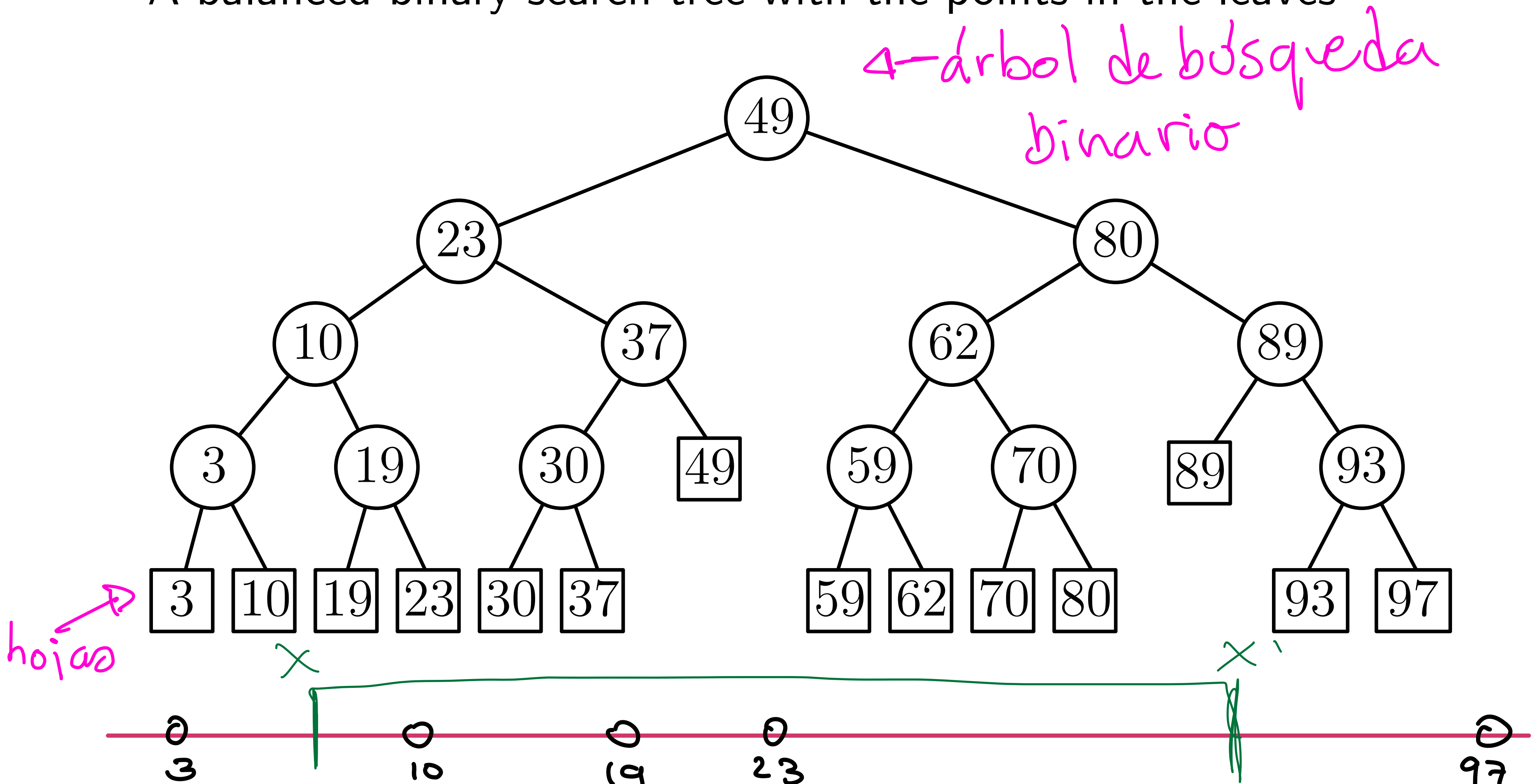
Range trees

Balanced binary search trees

1D.

A balanced binary search tree with the points in the leaves

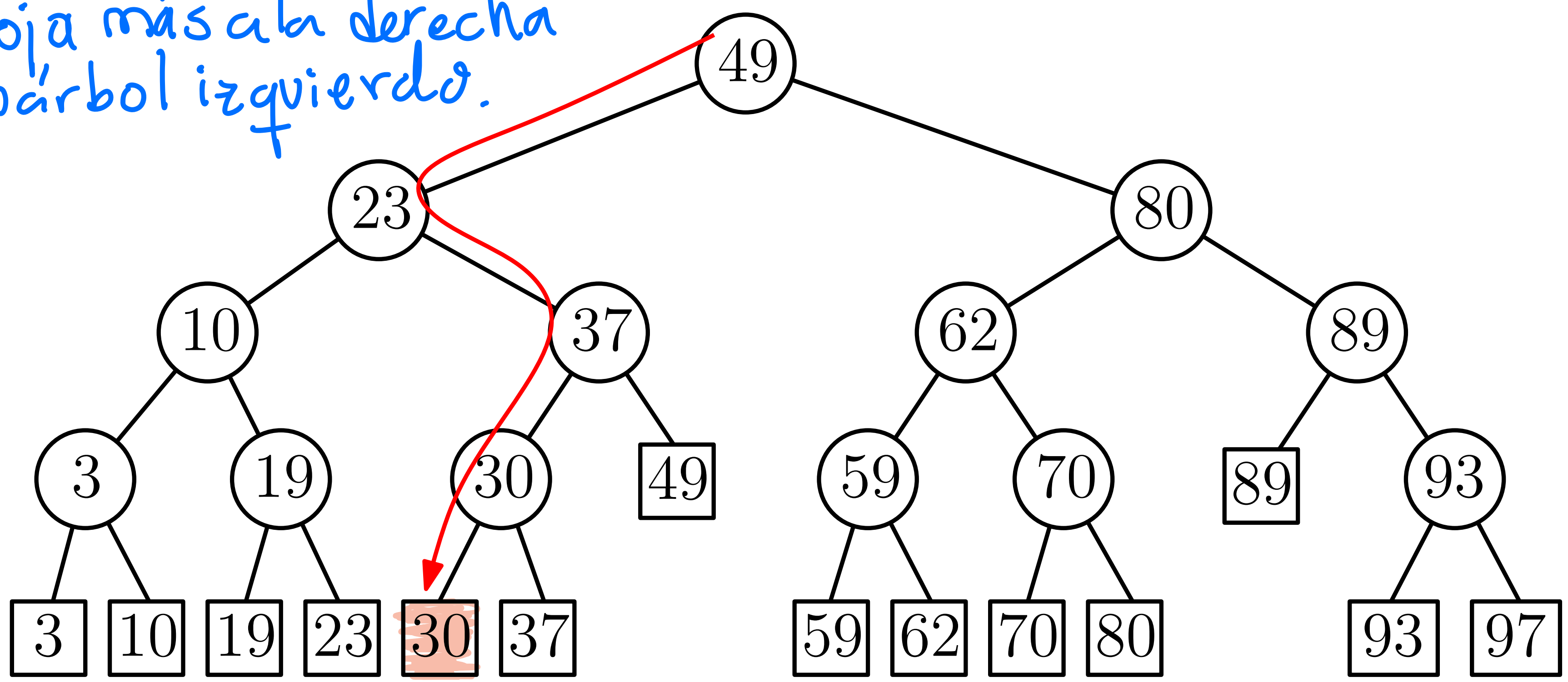


Balanced binary search trees

[25,90]

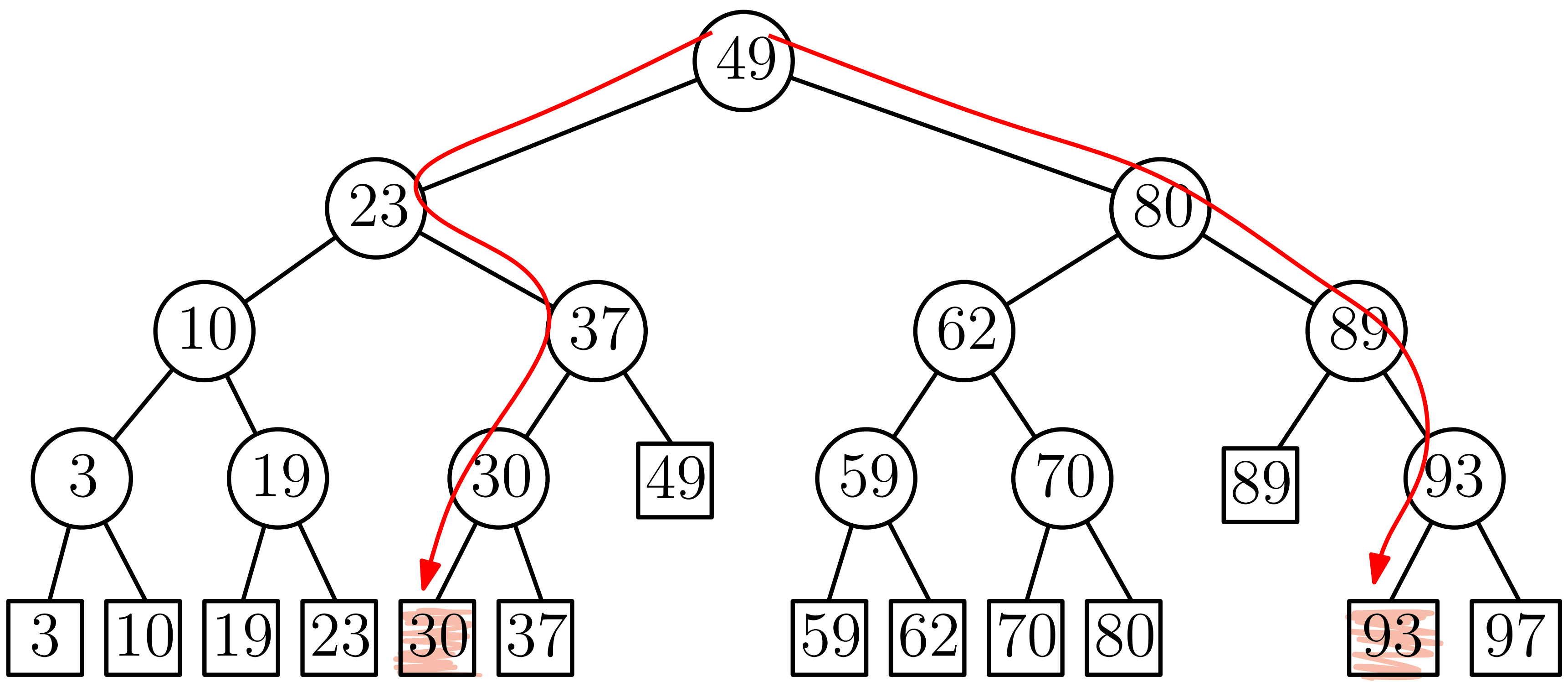
The search path for 25

Raíz: hoja más a la derecha del subárbol izquierdo.



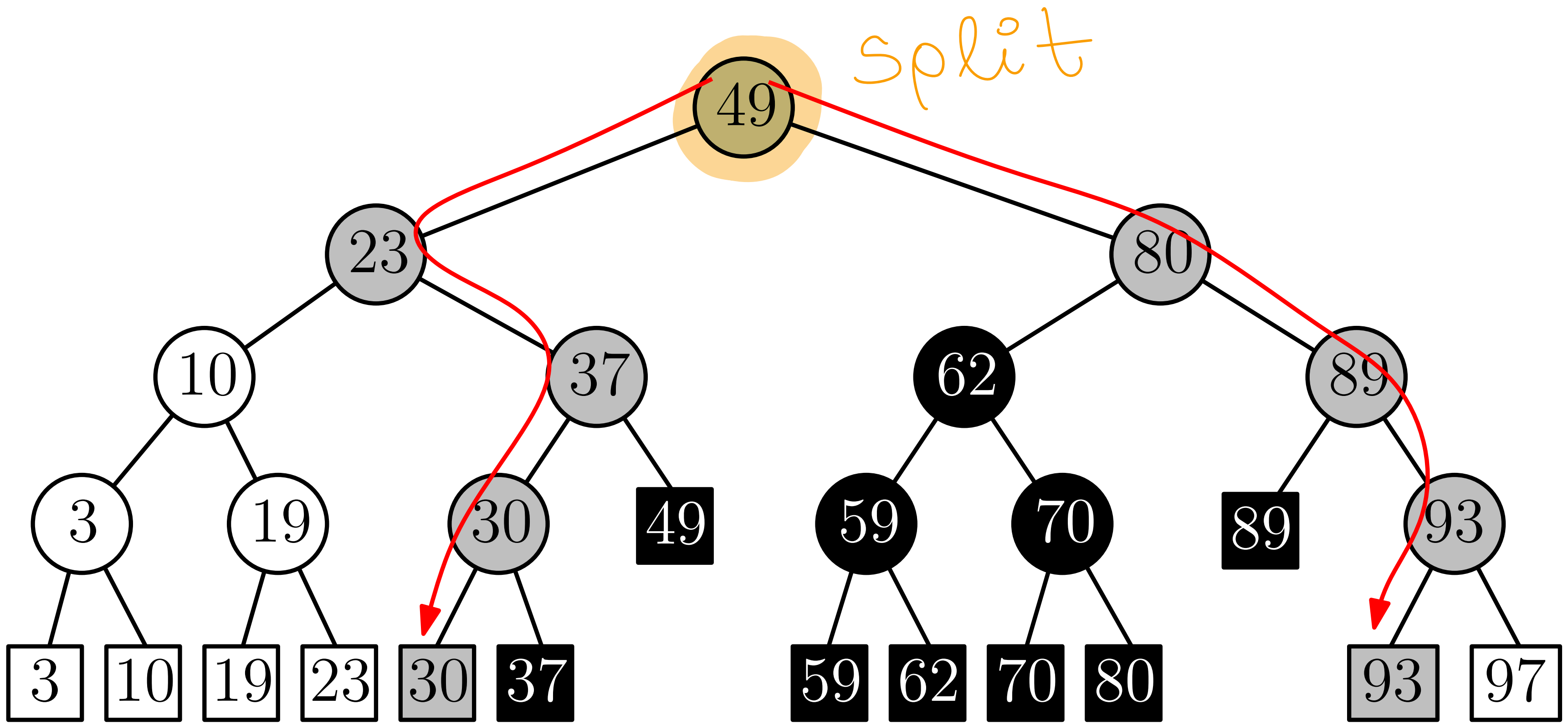
Balanced binary search trees

The search paths for 25 and for 90



Example 1D range query

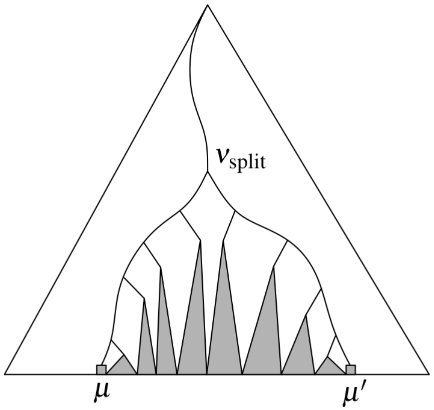
A 1-dimensional range query with $[25, 90]$



1D range query algorithm

Algorithm $\text{1DRANGEQUERY}(\mathcal{T}, [x : x'])$

1. $v_{\text{split}} \leftarrow \text{FINDSPLITNODE}(\mathcal{T}, x, x')$
2. **if** v_{split} is a leaf
3. **then** Check if the point in v_{split} must be reported.
4. **else** $v \leftarrow lc(v_{\text{split}})$
5. **while** v is not a leaf
6. **do if** $x \leq x_v$
7. **then** $\text{REPORTSUBTREE}(rc(v))$
8. $v \leftarrow lc(v)$
9. **else** $v \leftarrow rc(v)$
10. Check if the point stored in v must be reported.
11. $v \leftarrow rc(v_{\text{split}})$
12. Similarly, follow the path to x', and ...



Query time analysis

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: a (sub)tree with m leaves has $m-1$ internal nodes; traversal visits $O(m)$ nodes and finds m points for the output

nodos reportados.

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

$O(1) \leq K \leq O(n)$

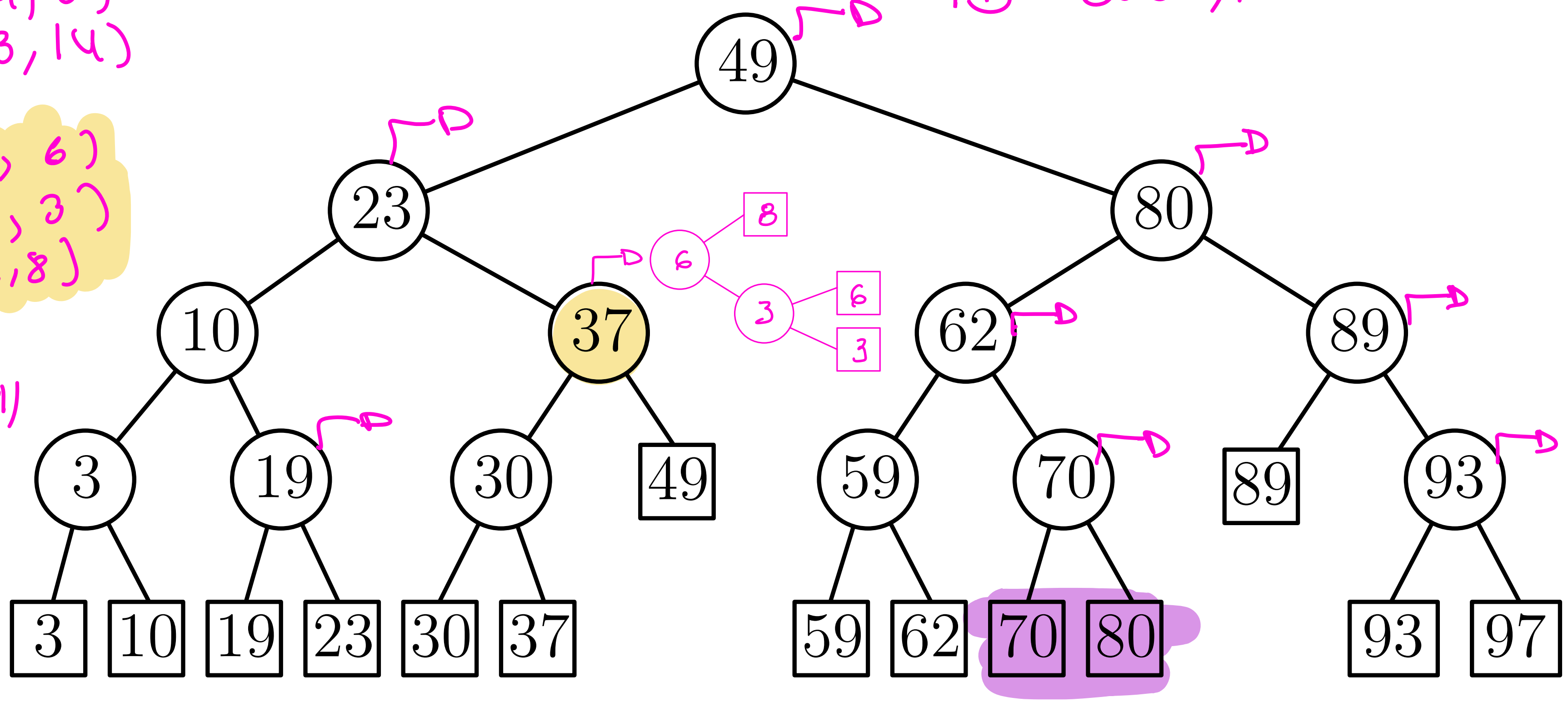
$\Rightarrow O(\lg n) \leq O(\lg n + k) \leq O(n)$

Range Trees.

Dado un conjunto de puntos en $\mathbb{R}^2$.

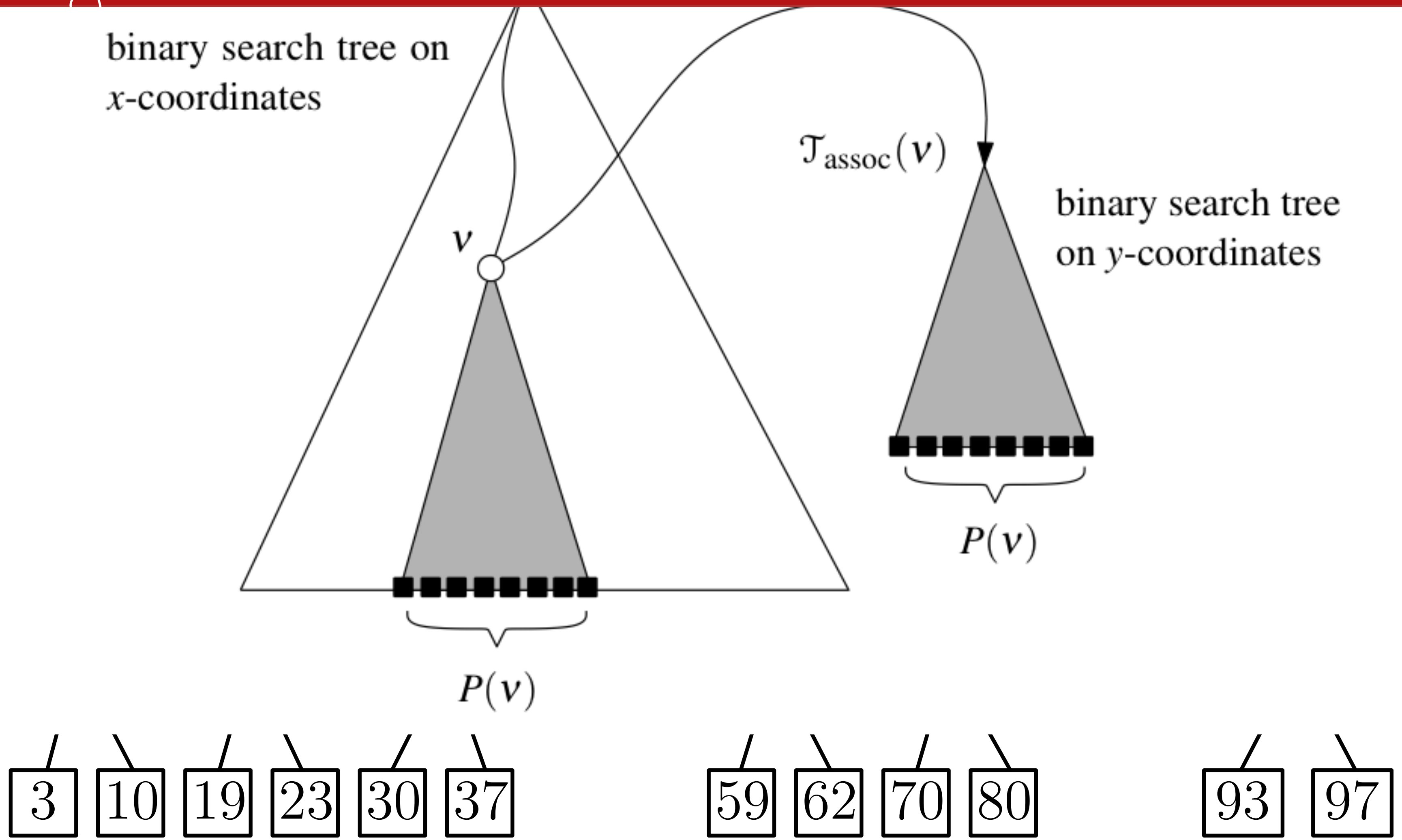
(3,5)
(10,2)
(19,6)
(23,14)
⋮
(30,6)
(37,3)
(49,8)
⋮
(97,14)

1D en X

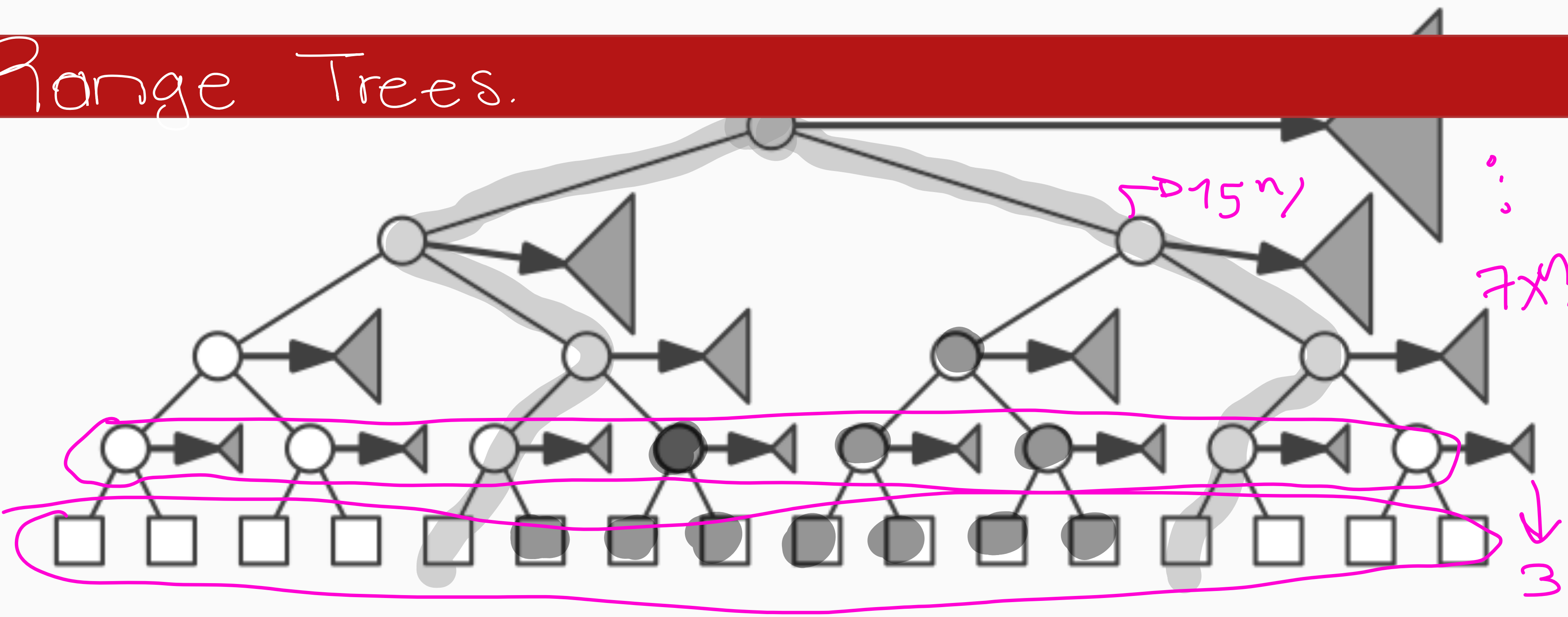


coord x

Range Trees.

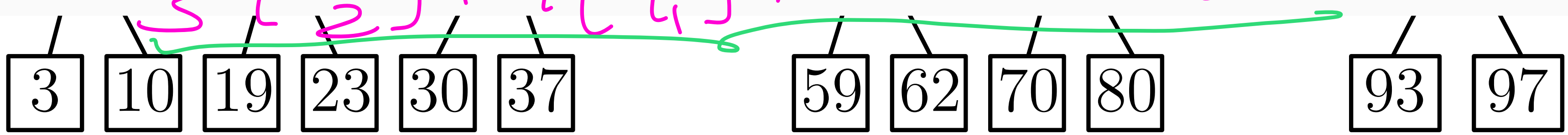

binary search tree on
x-coordinates
v
$\mathcal{T}_{assoc}(v)$
binary search tree
on y-coordinates
$P(v)$
$P(v)$
3
10
19
23
30
37
59
62
70
80
93
97

Range Trees.

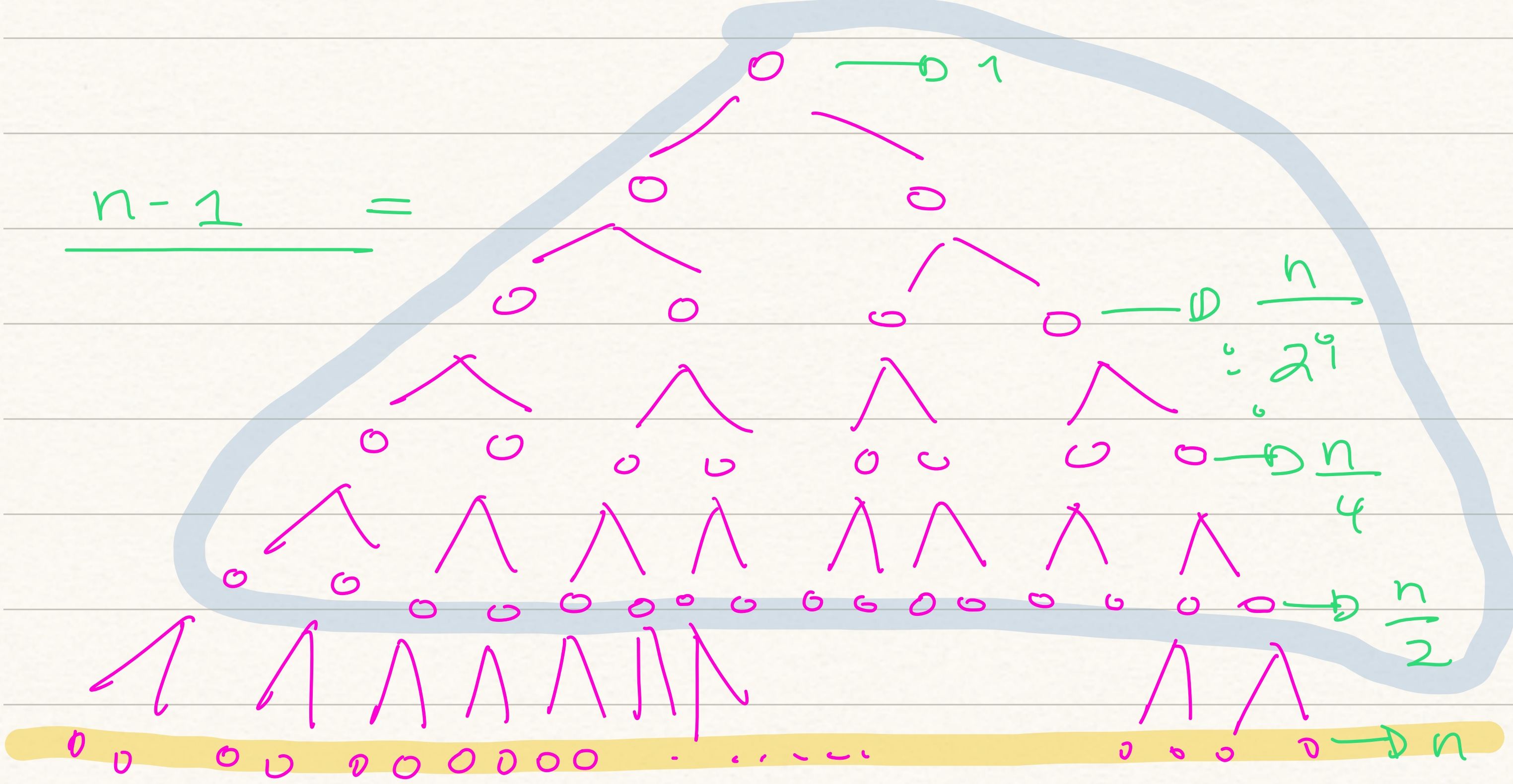


Question: How much storage does this take?

$$\underbrace{3\left(\frac{n}{2}\right) + 7\left(\frac{n}{4}\right) + \dots + 7(n)}_{\lg n}$$



$$\frac{n-1}{} =$$



En cada nivel hay $\frac{n}{2^i}$ nodos, con $0 \le i \le \lg n$.

Suma tamaño de Jassoc:

$$3\left(\frac{n}{2^1}\right)+7\left(\frac{n}{2^2}\right)+15\left(\frac{n}{2^3}\right)+\dots+2^{i+1}-1\left(\frac{n}{2^i}\right)+$$

$$\dots+\underbrace{2^{\lg n+1}-1\left(\frac{n}{2^{\lg n}}\right)}_{2n-1.}$$

Notemos que cada término de la forma

$$2^{i+1}-1\left(\frac{n}{2^i}\right)$$ se puede escribir como:

$$\frac{2^i \cdot 2}{2^i}-1\,(n)$$

$$= 2-1\,(n) = \underline{n}.$$

Entonces, la suma se puede escribir como:

$$\underbrace{n+n+\dots+n}_{\lg n\text{-veces}} = O(n \lg n).$$

To analyze storage, two arguments can be used:

- 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

Lemma 5.6 *A range tree on a set of n points in the plane requires $O(n \log n)$ storage.*

Algorithm BUILD2DRANGETREE(P)
Input. A set P of points in the plane.
Output. The root of a 2-dimensional range tree.

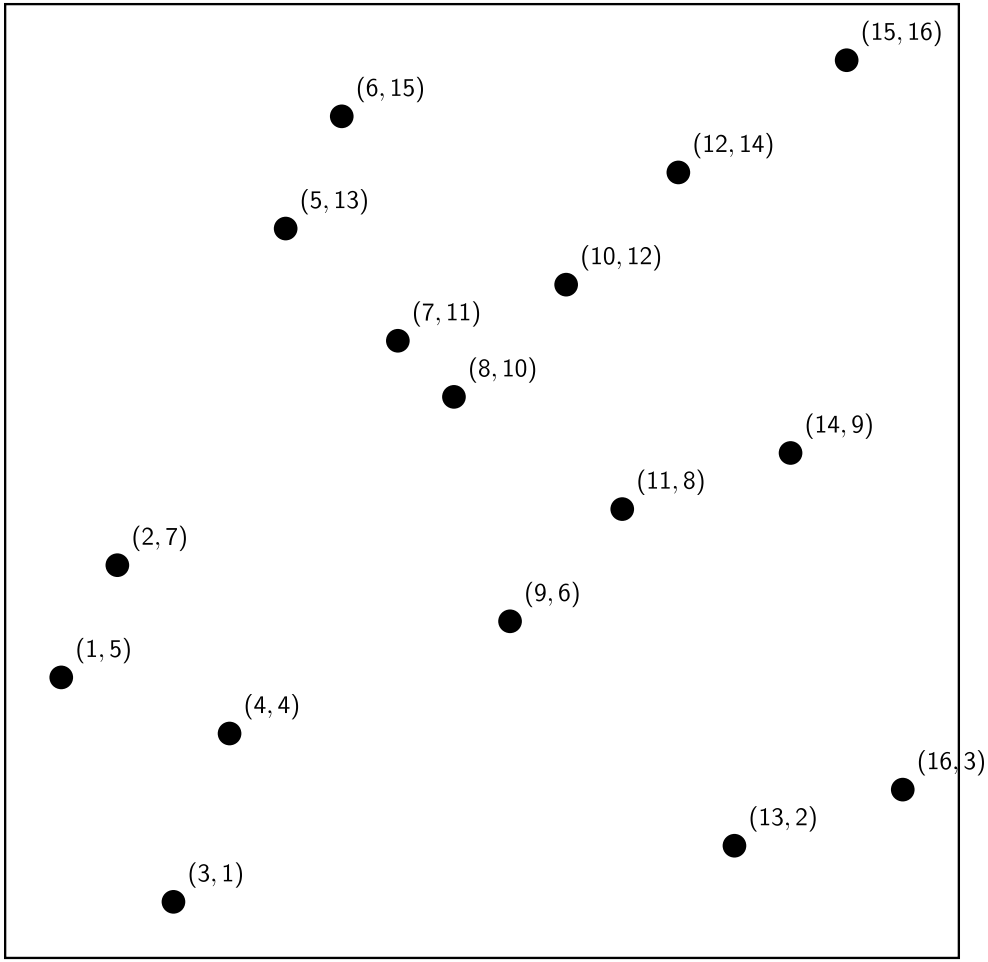
O(n)

si el conjunto está ordenado.

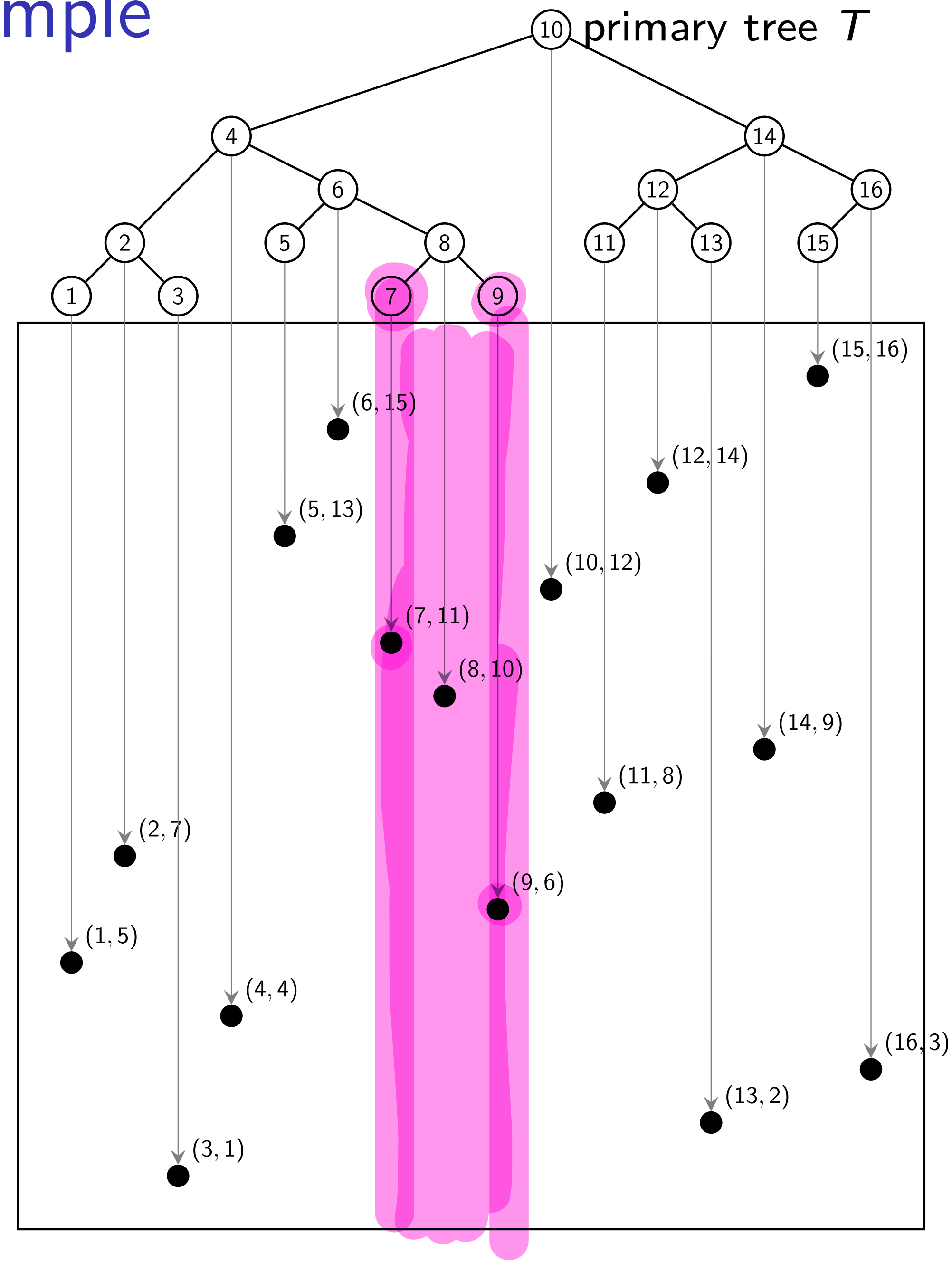
1. Construct the associated structure: Build a binary search tree $\mathcal{T}_{\text{assoc}}$ on the set P_y of y-coordinates of the points in P. Store at the leaves of $\mathcal{T}_{\text{assoc}}$ not just the y-coordinate of the points in P_y, but the points themselves.
2. **if** P contains only one point
3. **then** Create a leaf ν storing this point, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of ν.
4. **else** Split P into two subsets; one subset P_{left} contains the points with x-coordinate less than or equal to x_{mid}, the median x-coordinate, and the other subset P_{right} contains the points with x-coordinate larger than x_{mid}.
5. $\nu_{\text{left}} \leftarrow$ BUILD2DRANGETREE(P_{left})
6. $\nu_{\text{right}} \leftarrow$ BUILD2DRANGETREE(P_{right})
7. Create a node ν storing x_{mid}, make ν_{left} the left child of ν, make ν_{right} the right child of ν, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of ν.
8. **return** ν

¿Complejidad? O(n lg n).

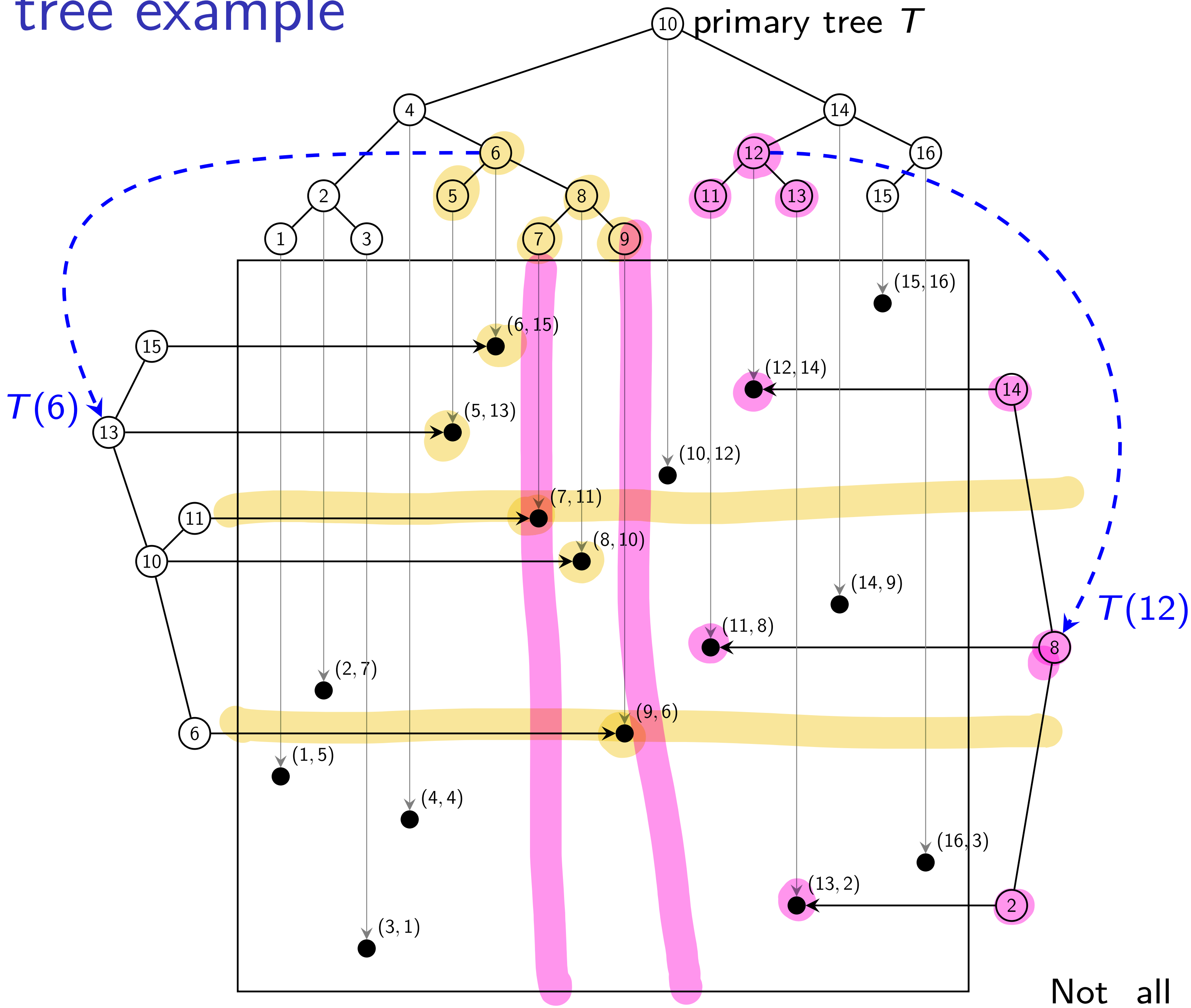
Range tree example



Range tree example

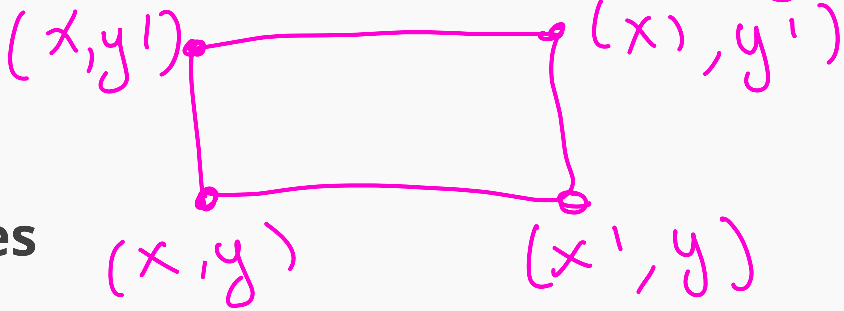


Range tree example



Not all associate trees are shown.

Query : $[x, x'] \times [y, y']$



2D Range trees

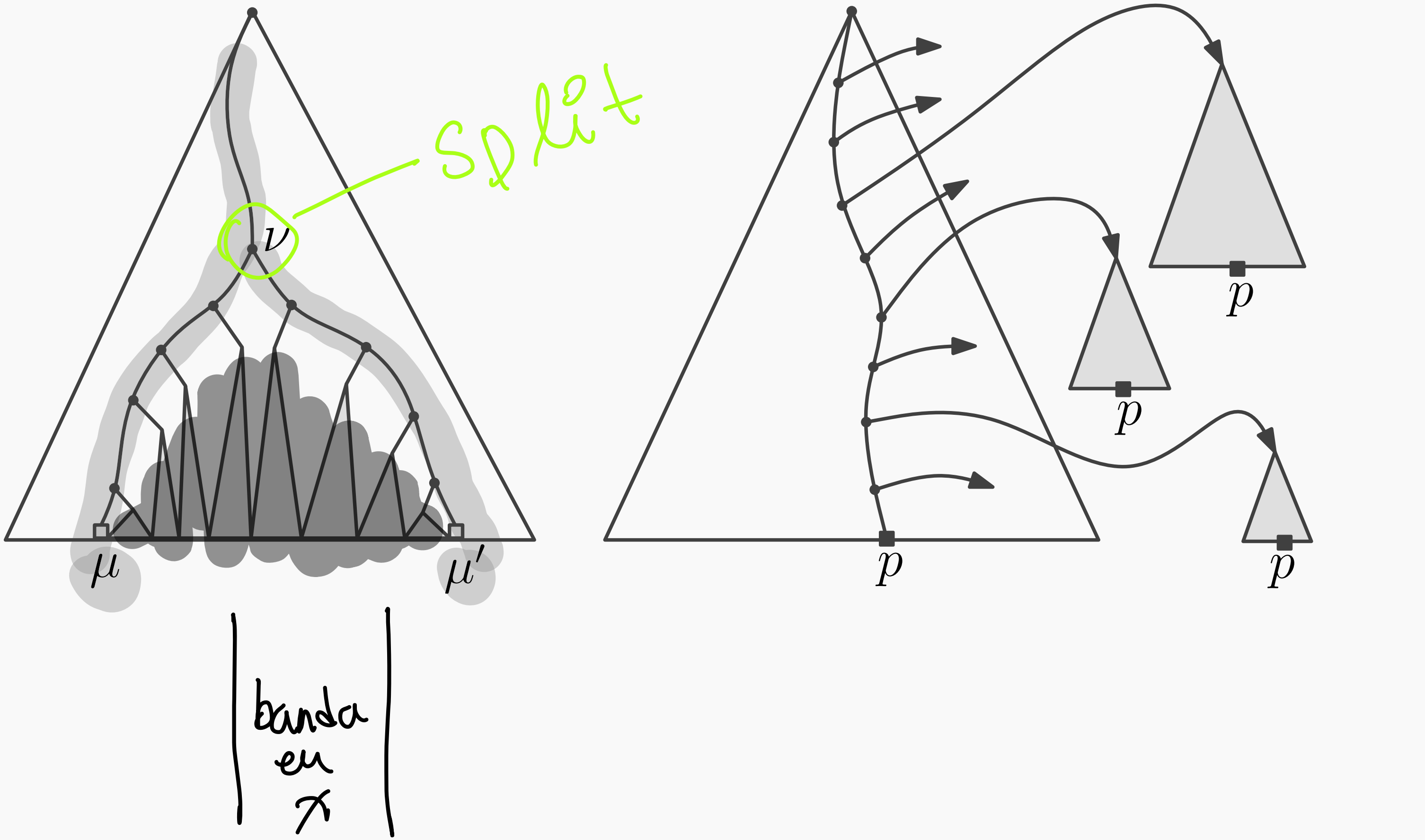
Querying

2D range queries

How are queries performed and why are they correct?

- Are we sure that each answer is found?
- Are we sure that the same point is found only once?

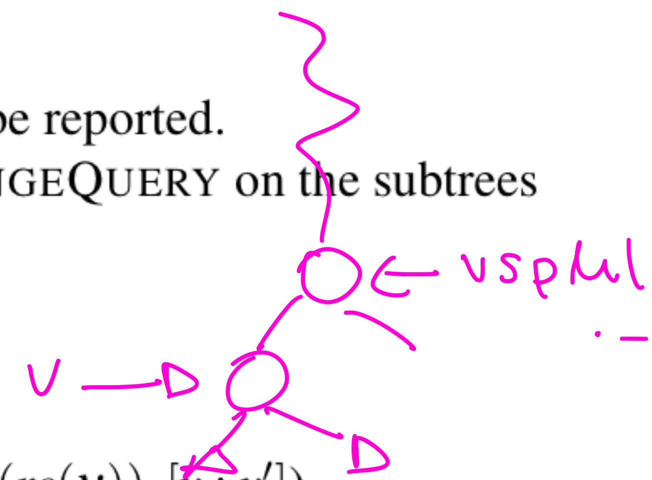
2D range queries



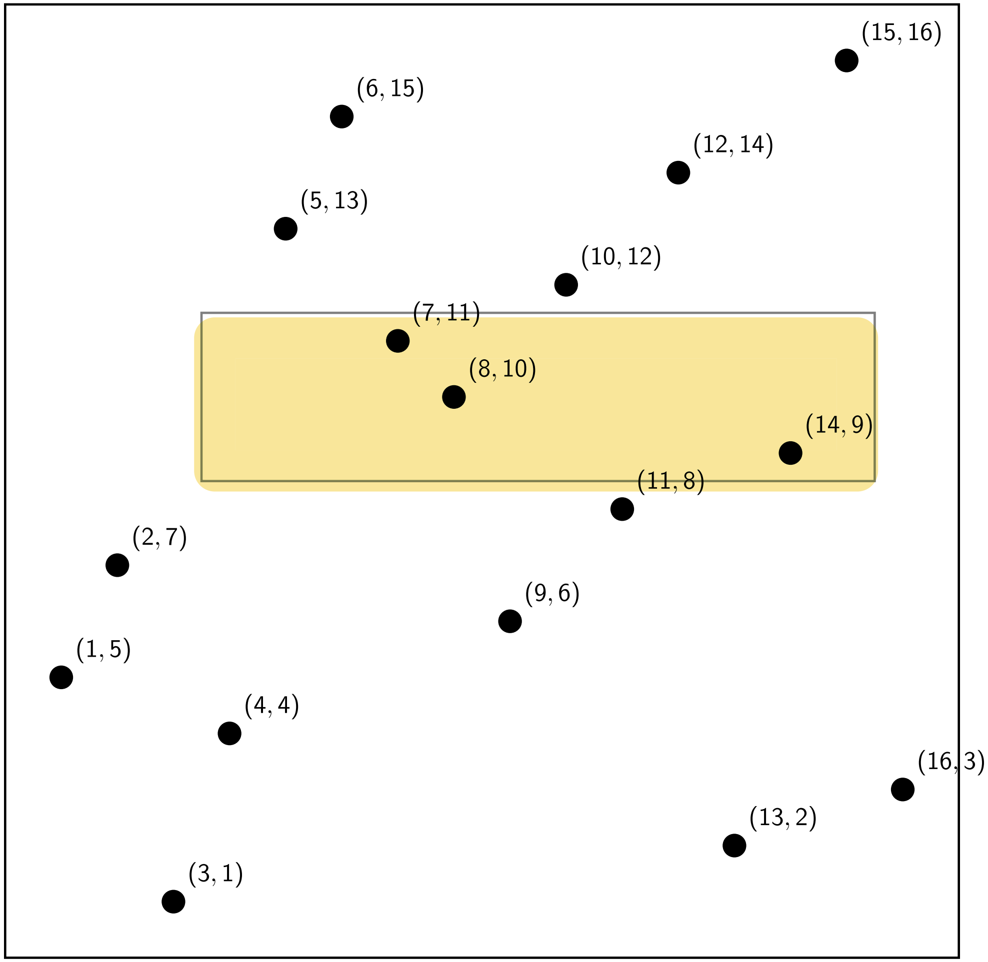
Balanced binary search trees

Algorithm 2DRANGEQUERY($\mathcal{T}, [x:x'] \times [y:y']$)
Input. A 2-dimensional range tree $\mathcal{T}$ and a range $[x:x'] \times [y:y']$.
Output. All points in $\mathcal{T}$ that lie in the range.

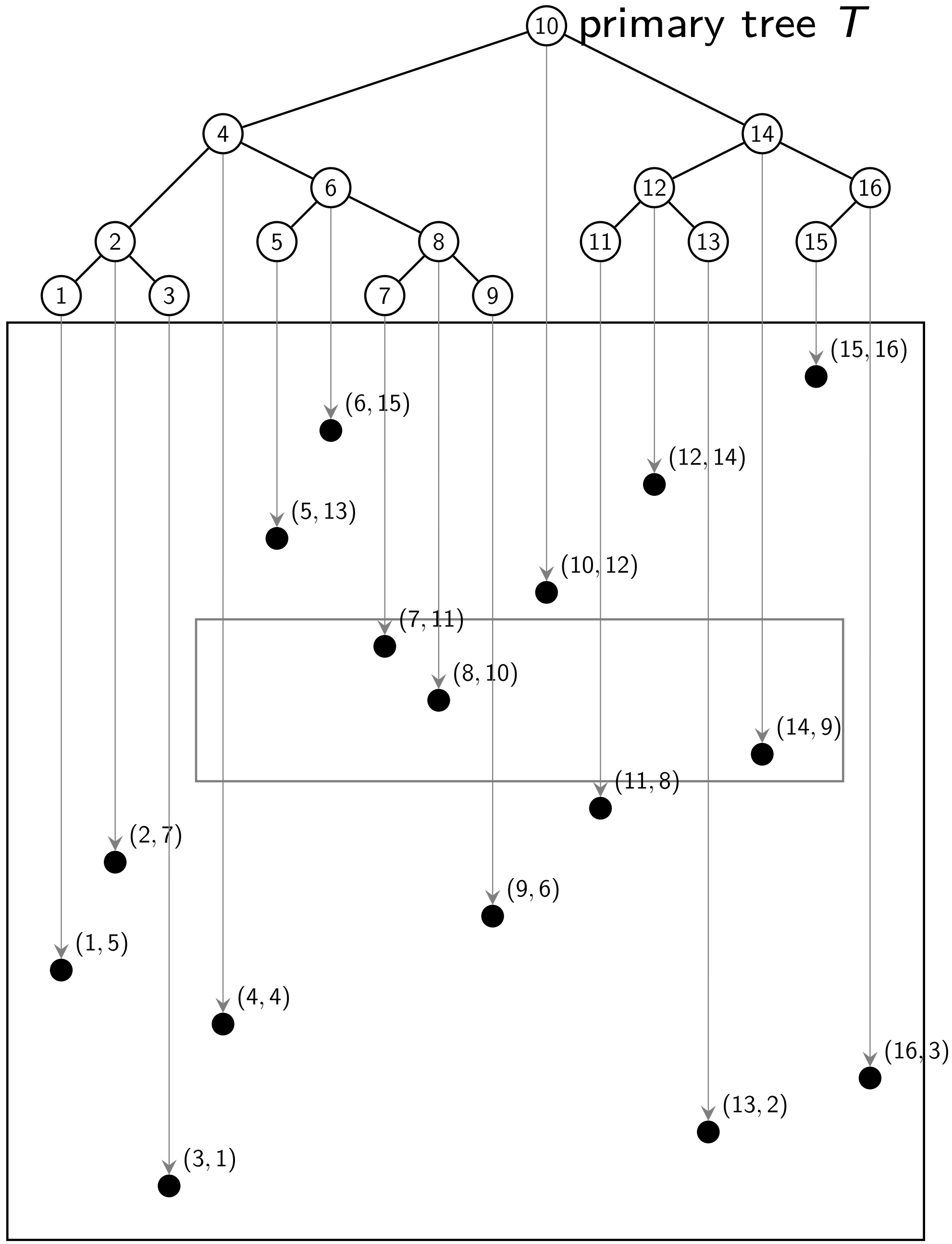
1. $\nu_{\text{split}} \leftarrow$ FINDSPLITNODE($\mathcal{T}, x, x'$)
2. **if** ν_{split} is a leaf
3.   **then** Check if the point stored at ν_{split} must be reported.
4.   **else** (∗ Follow the path to x and call 1DRANGEQUERY on the subtrees right of the path. ∗)
5. $\nu \leftarrow lc(\nu_{\text{split}})$
6.   **while** ν is not a leaf
7.   **do if** $x \leqslant x_\nu$
8.   **then** 1DRANGEQUERY($\mathcal{T}_{\text{assoc}}(rc(\nu)), [y:y']$)
9. $\nu \leftarrow lc(\nu)$
10.   **else** $\nu \leftarrow rc(\nu)$
11.   Check if the point stored at ν must be reported.
12.   Similarly, follow the path from $rc(\nu_{\text{split}})$ to x', call 1DRANGE-QUERY with the range $[y:y']$ on the associated structures of subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.



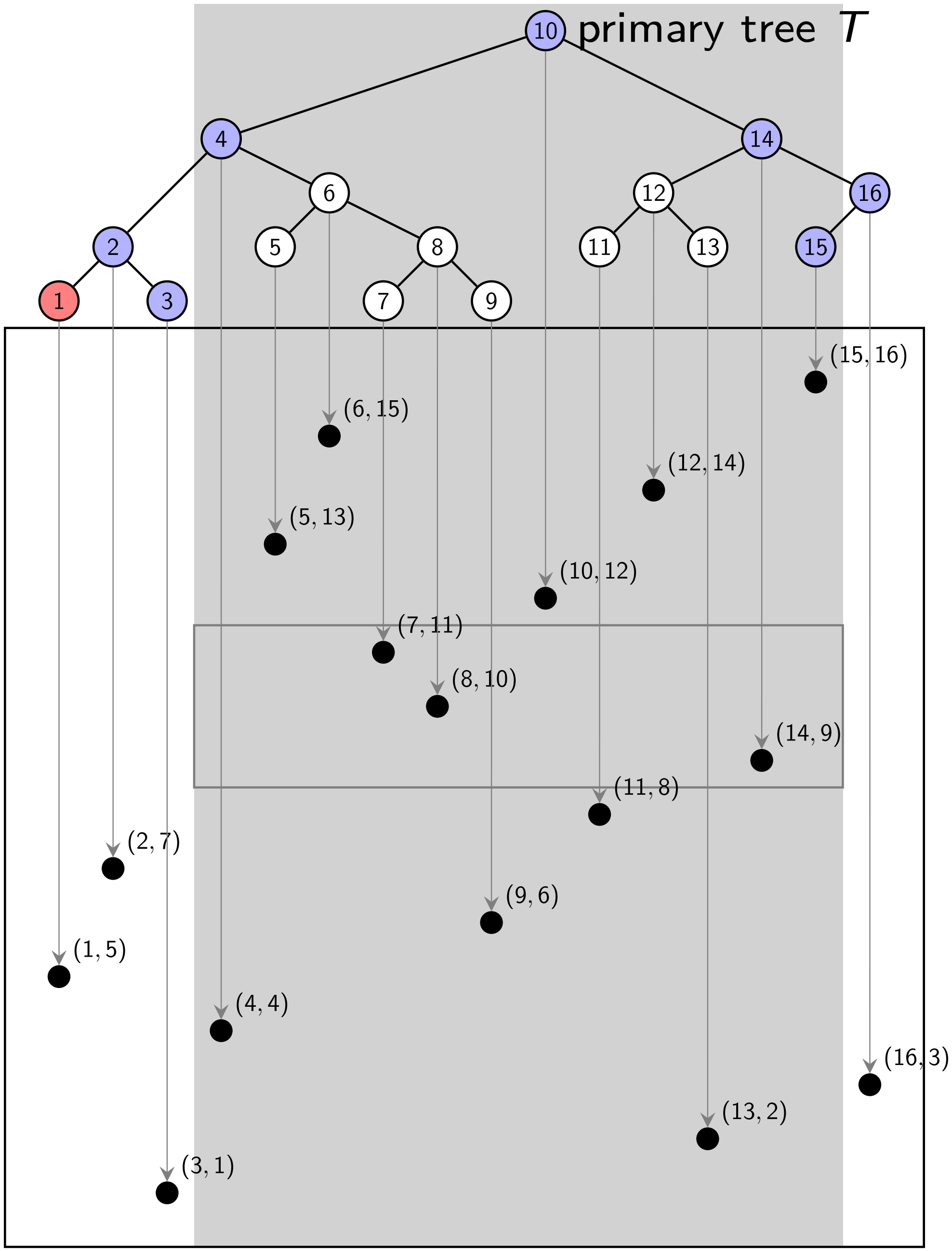
Range tree range search example



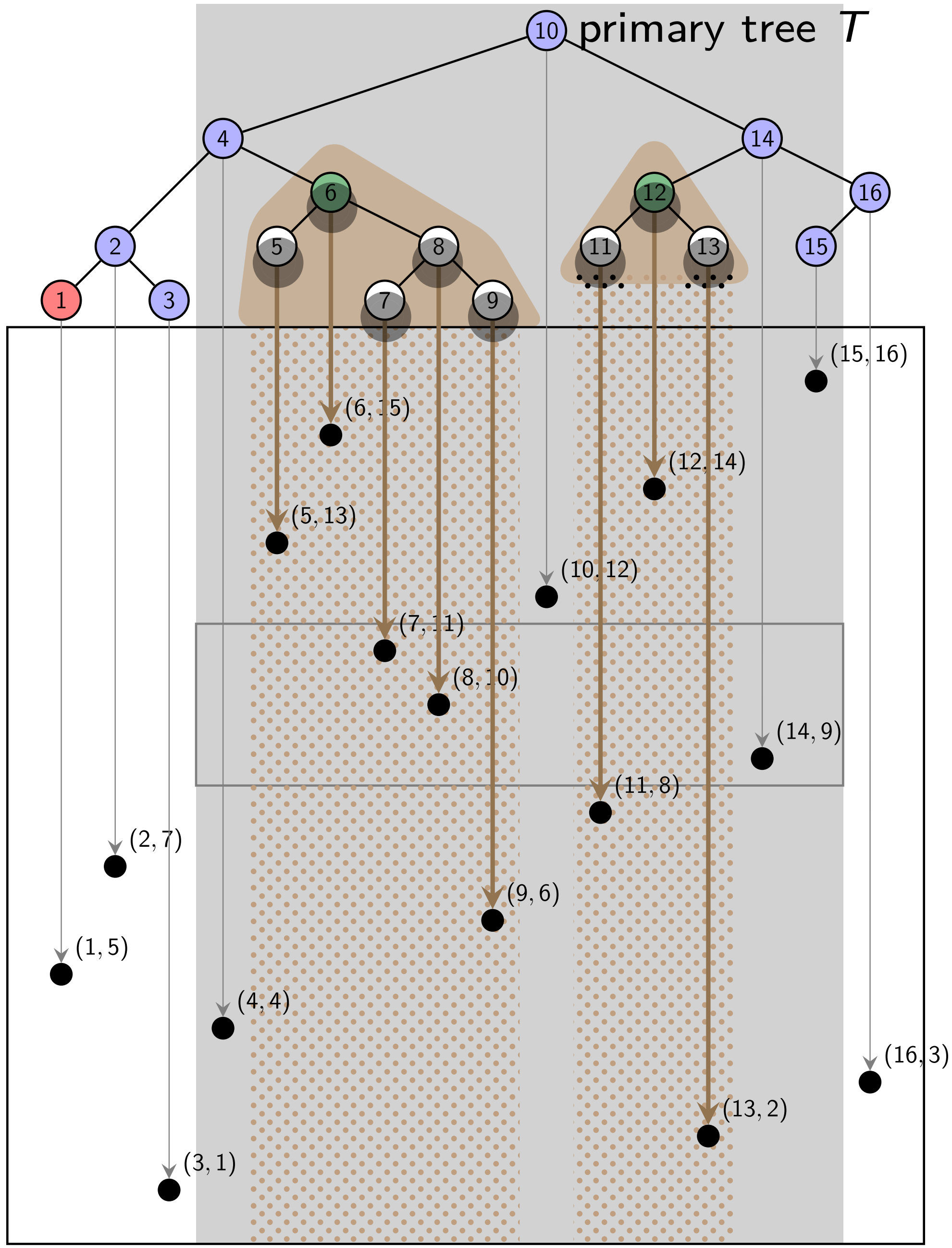
Range tree range search example



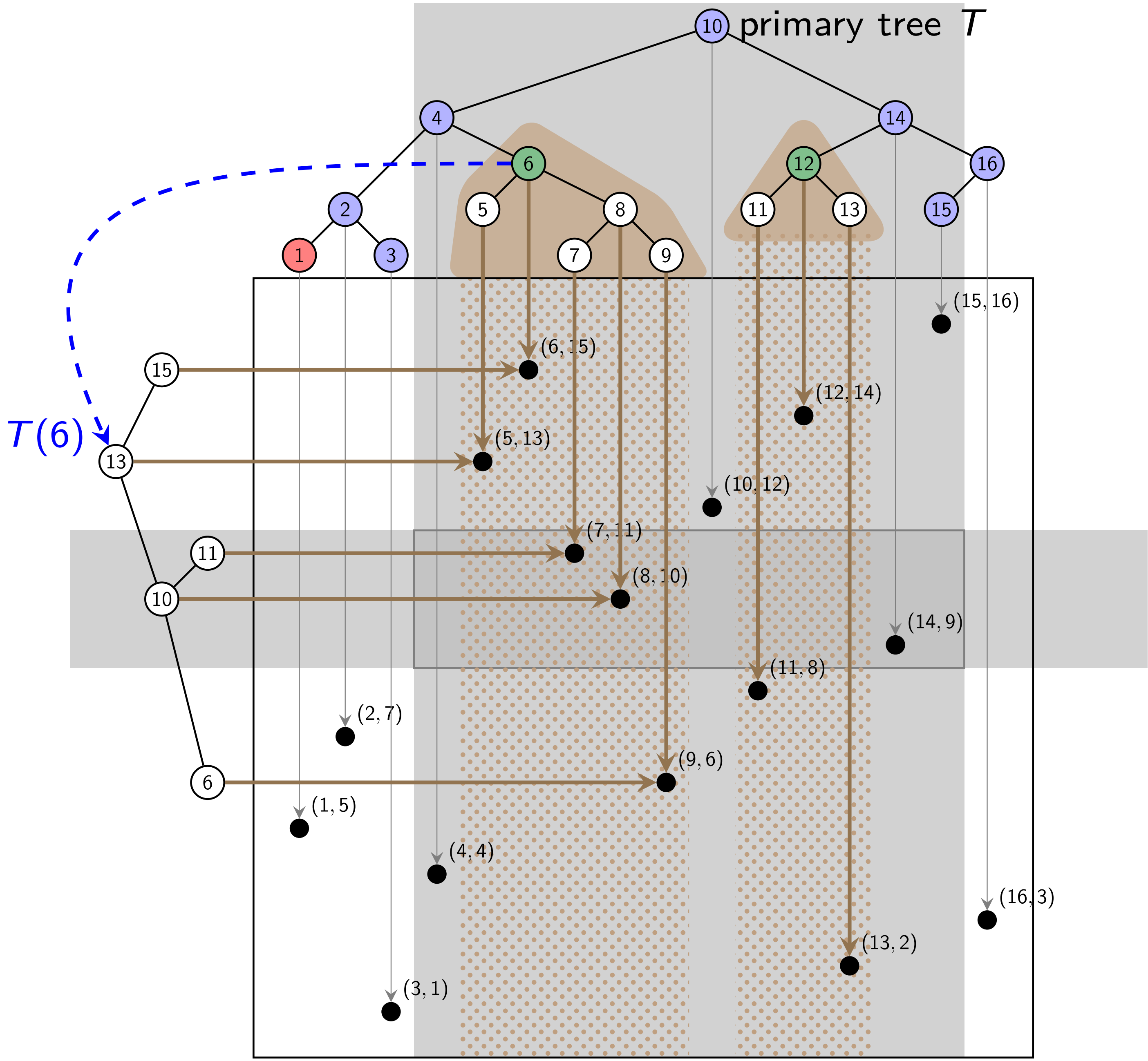
Range tree range search example



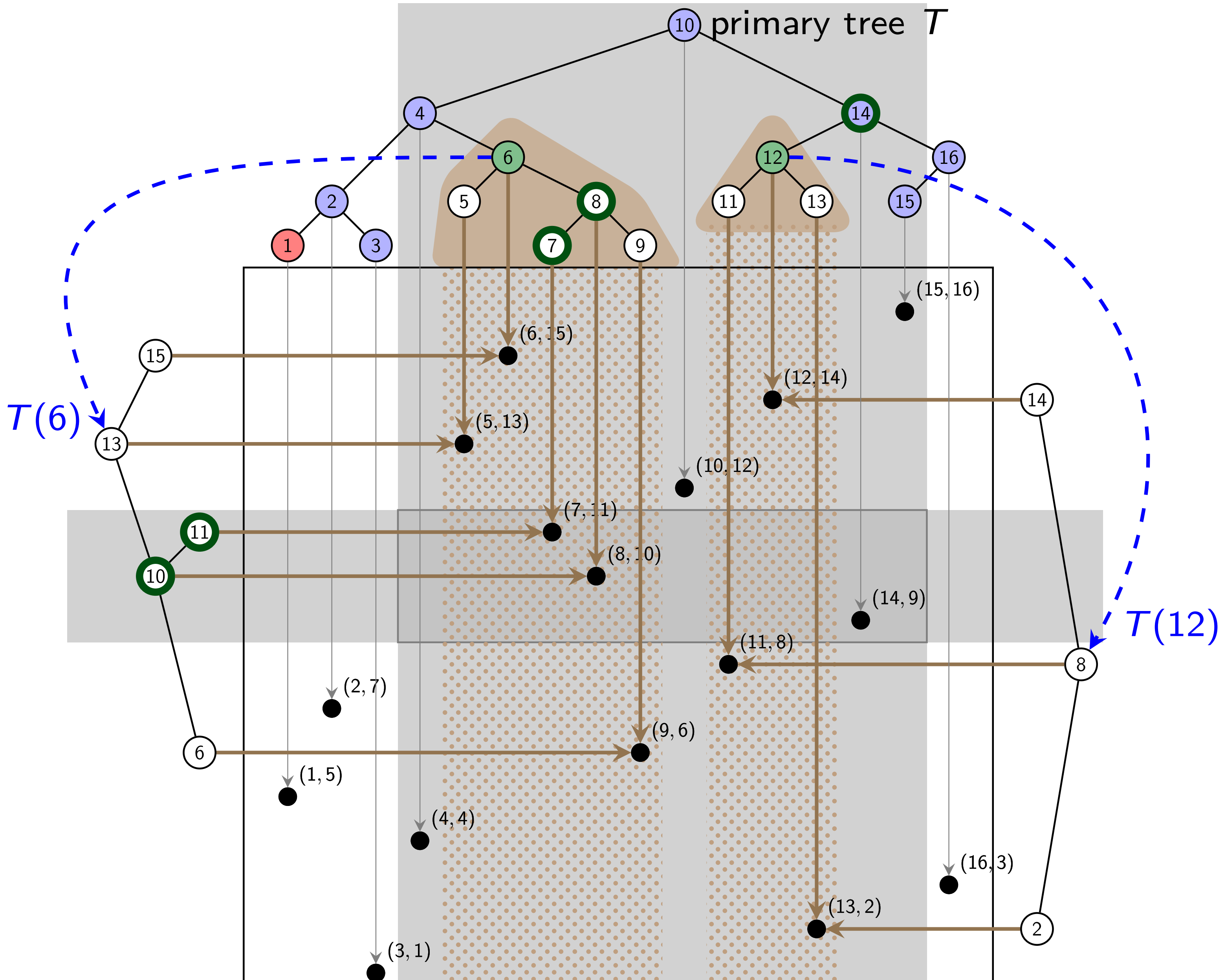
Range tree range search example



Range tree range search example



Range tree range search example



Algorithm 2DRANGEQUERY($\mathcal{T}$, $[x:x'] \times [y:y']$)

Input. A 2-dimensional range tree $\mathcal{T}$ and a range $[x:x'] \times [y:y']$.

Output. All points in $\mathcal{T}$ that lie in the range.

¿Complejidad

```
v_split ← FINDSPLITNODE(T, x, x')
if v_split is a leaf
   then Check if the point stored at v_split must be reported.
   else (* Follow the path to x and call 1DRANGEQUERY on the subtrees
           right of the path. *)
        v ← lc(v_split)
        while v is not a leaf
            do if x ≤ x_v
                  then 1DRANGEQUERY(T_assoc(rc(v)), [y : y'])
                       v ← lc(v)
                  else v ← rc(v)
        Check if the point stored at v must be reported.
        Similarly, follow the path from rc(v_split) to x', call 1DRANGE-
        QUERY with the range [y : y'] on the associated structures of sub-
        trees left of the path, and check if the point stored at the leaf where
        the path ends must be reported.
```

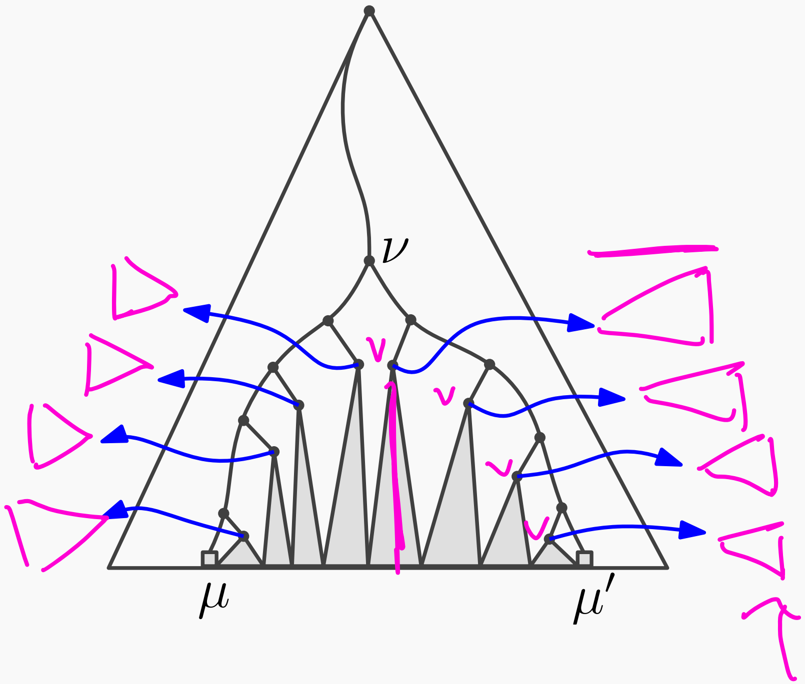
?

3 10 19 23 30 37 59 62 70 80 93 97

2D range query time

Question: How much time does a 2D range query take?

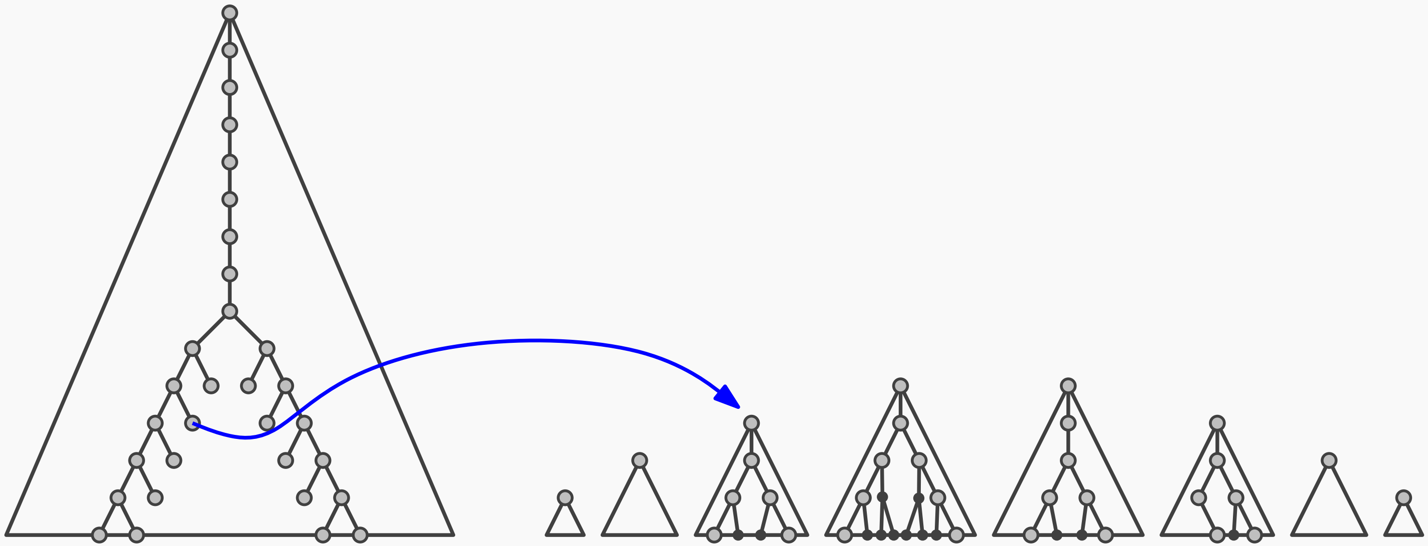
Subquestions: In how many associated structures do we search? How much time does each such search take?



Query:
- #nodos grises
- #nodos negros.

2D range query efficiency

Use the concept of grey and black nodes again:



2D range query efficiency

We visit $O(\log n)$ grey nodes in the main structure.

We perform a 1D range query using the associated structure of $O(\log n)$ nodes v; at most two per level.

Each such query visits $O(\log n_v)$ grey nodes and $O(k_v)$ black nodes, and thus takes $O(\log n_v + k_v)$ time, where

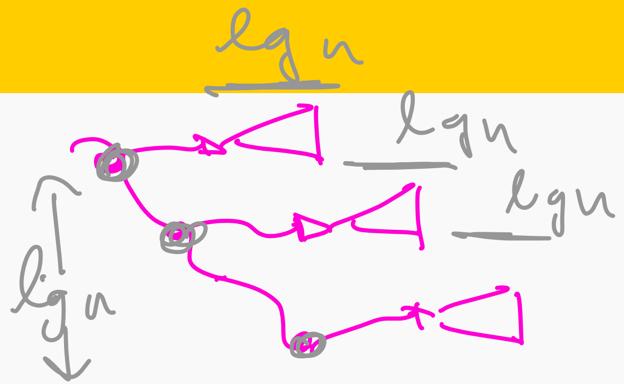
$n_v =$ #leaves in subtree v, and

$k_v =$ #reported points from subtree of v.

So the query time is

$$\sum_v O(\log n_v + k_v) = \sum_v \lg n_v + \sum_v k_v$$

2D range query efficiency



$$= \sum_v \lg n_v + \sum_v k_v$$

So the number of grey nodes is $\sum_v O(\log n_v) = O(\log^2 n)$, since $n_v \leq n$

The number of black nodes is $\sum_v O(k_v) = O(k)$ if k points are reported (since $k = \sum_v k_v$).

The query time is $O(\log^2 n + k)$, where k is the size of the output

2D range query efficiency

So the number of grey nodes is $\sum_v O(\log n_v) = O(\log^2 n)$, since $n_v \leq n$

The number of black nodes is $\sum_v O(k_v) = O(k)$ if k points are reported (since $k = \sum_v k_v$).

The query time is $O(\log^2 n + k)$, where k is the size of the output

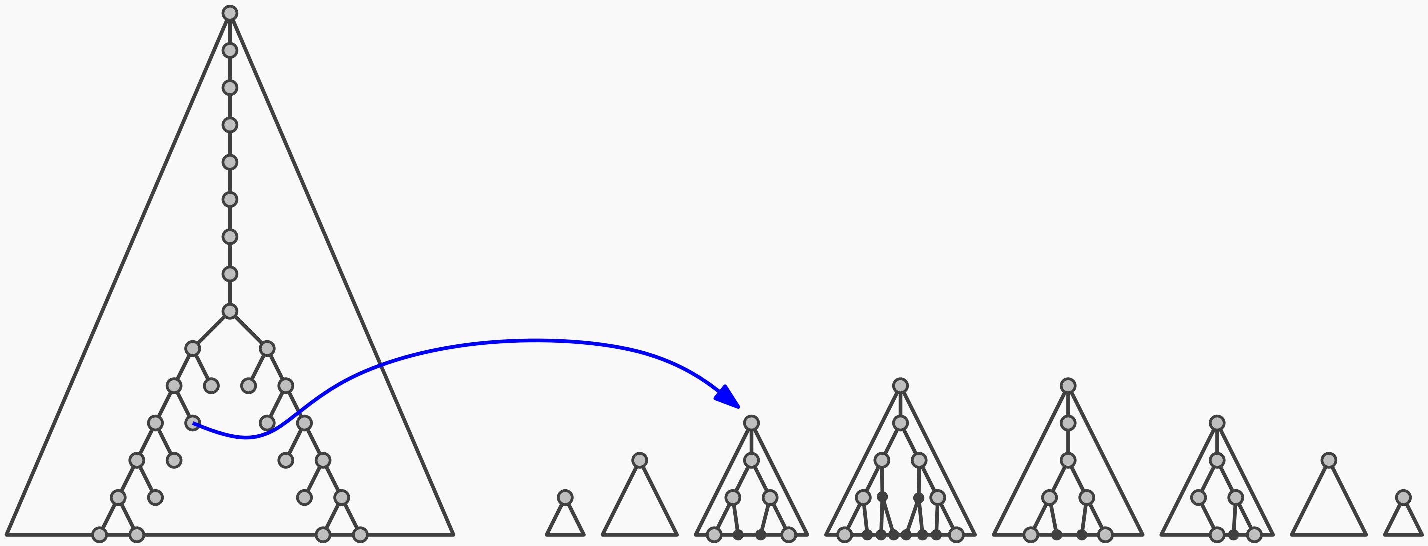
Result

Theorem: 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

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

2D range query efficiency

Question: How about range *counting* queries?

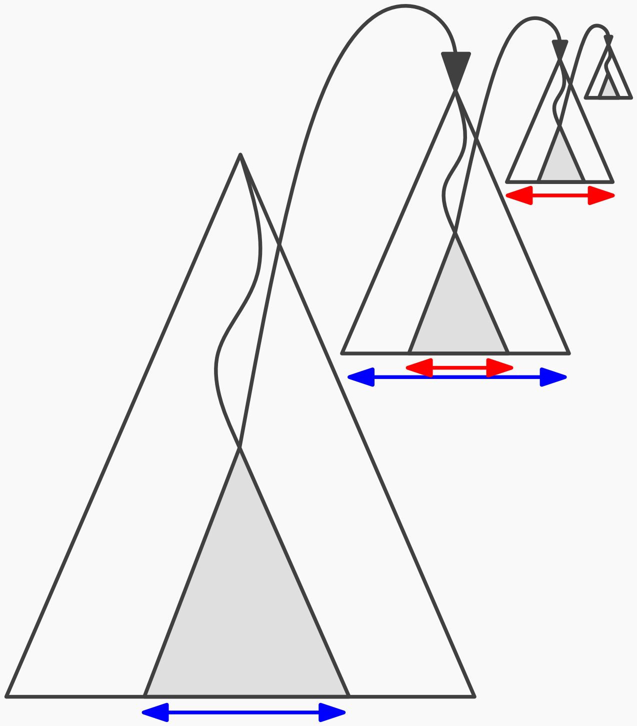


2D Range trees

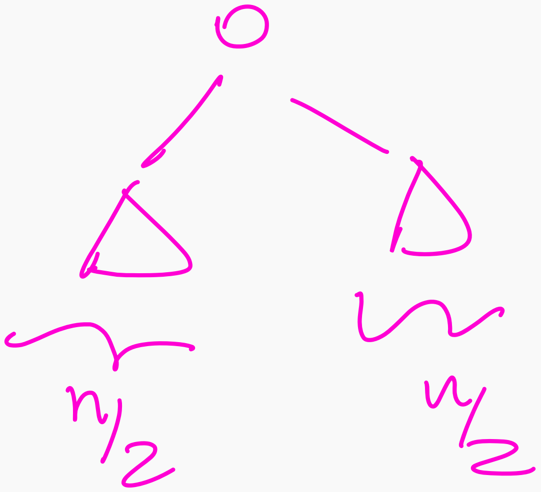
Higher dimensions

Higher dimensional range trees

A d-dimensional range tree has a main tree which is a one-dimensional balanced binary search tree on the first coordinate, where every node has a pointer to an associated structure that is a $(d-1)$-dimensional range tree on the other coordinates



The size $S_d(n)$ of a d-dimensional range tree satisfies:



$$S_1(n) = O(n) \quad \text{for all } n$$

$$S_d(1) = O(1) \quad \text{for all } d$$

$$S_d(n) \leq \underline{2 \cdot S_d(n/2)} + S_{d-1}(n) \quad \text{for } d \geq 2$$

$d=3$ | $d=4$

$O(n \lg^3 n)$

This solves to $S_d(n) = O(n \log^{d-1} n)$

$O(n \lg^2 n)$

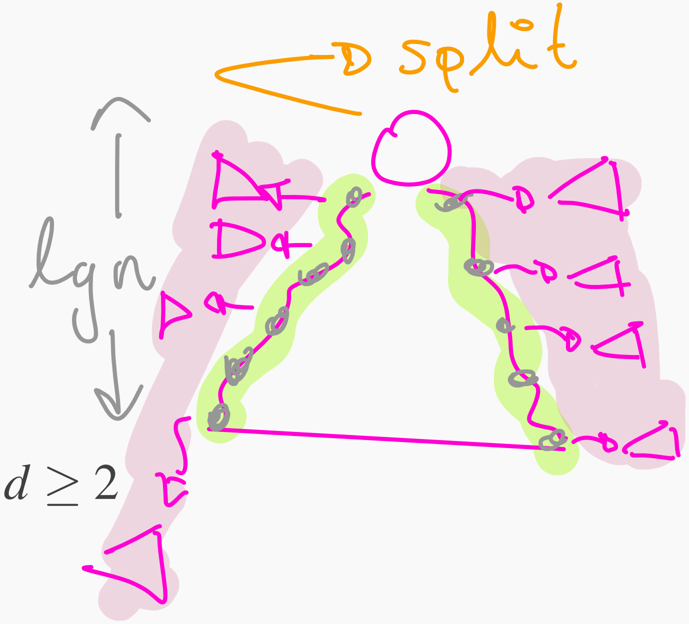
Query time

The number of grey nodes $G_d(n)$ satisfies:

$$G_1(n) = O(\log n) \quad \text{for all } n$$

$$G_d(1) = O(1) \quad \text{for all } d$$

$$G_d(n) \leq 2 \cdot \log n + 2 \cdot \log n \cdot G_{d-1}(n) \quad \text{for } d \geq 2$$



This solves to $G_d(n) = O(\log^d n)$

Result

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

↓ usando fractional cascading se mejora a $O(\lg^{d-1} n + k)$.

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