OI-OPPA. European Social Fund
Prague & EU: We invest in your future.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Cutting trees

Jakub Bokšanský

boksajak@fel.cvut.cz

29.11.2012

Motivation

- What is cutting tree and why should I care?
  - Solves planar range searching problem
  - Query can be answered in logarithmic time using cutting tree

- Example of usage: Counting points in desired area on a map
  - Query = area in which we are counting

- Query can be polygonal area
  - Not just axis aligned rectangle or circle

Planar range query example

- How many citizens live in this area?

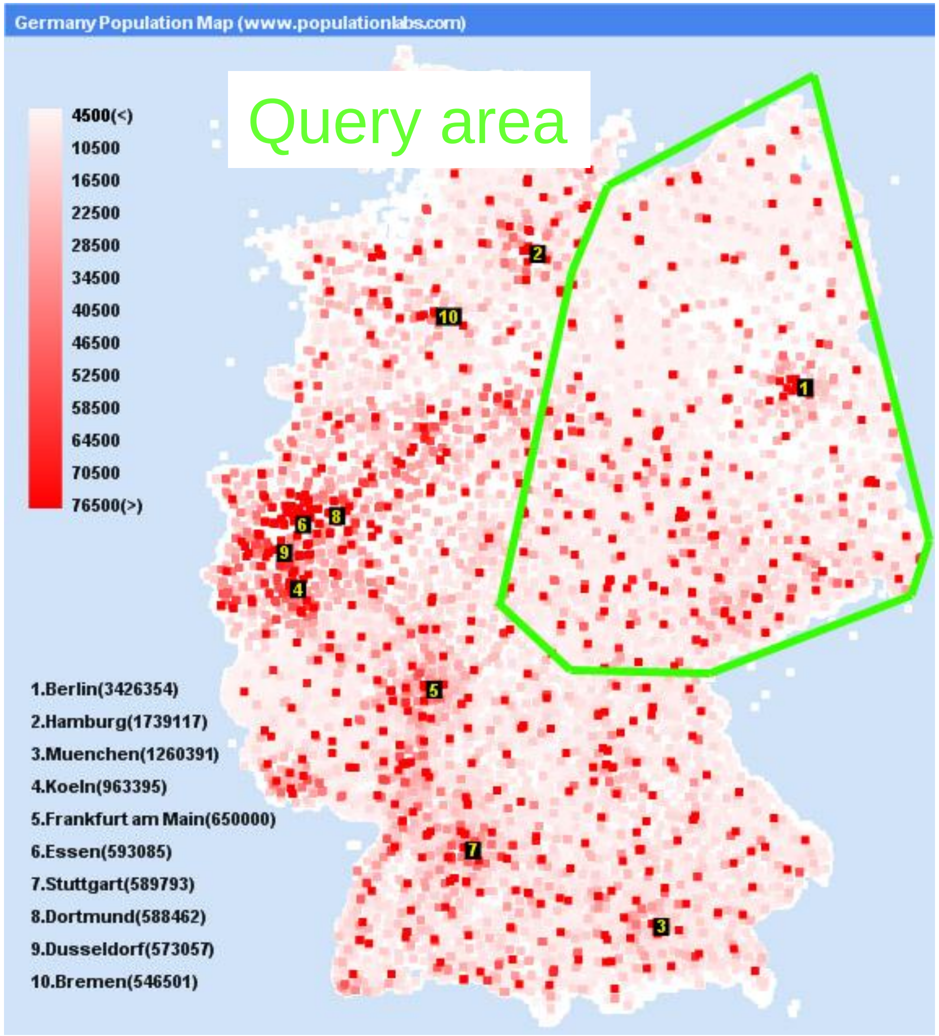


Image source: http://www.populationlabs.com/

Planar range query example

- How many citizens live in this area?
  1. Triangluate query
  2. Query each triangle separately
  3. Sum up results. Handle properly points on borders of triangles!

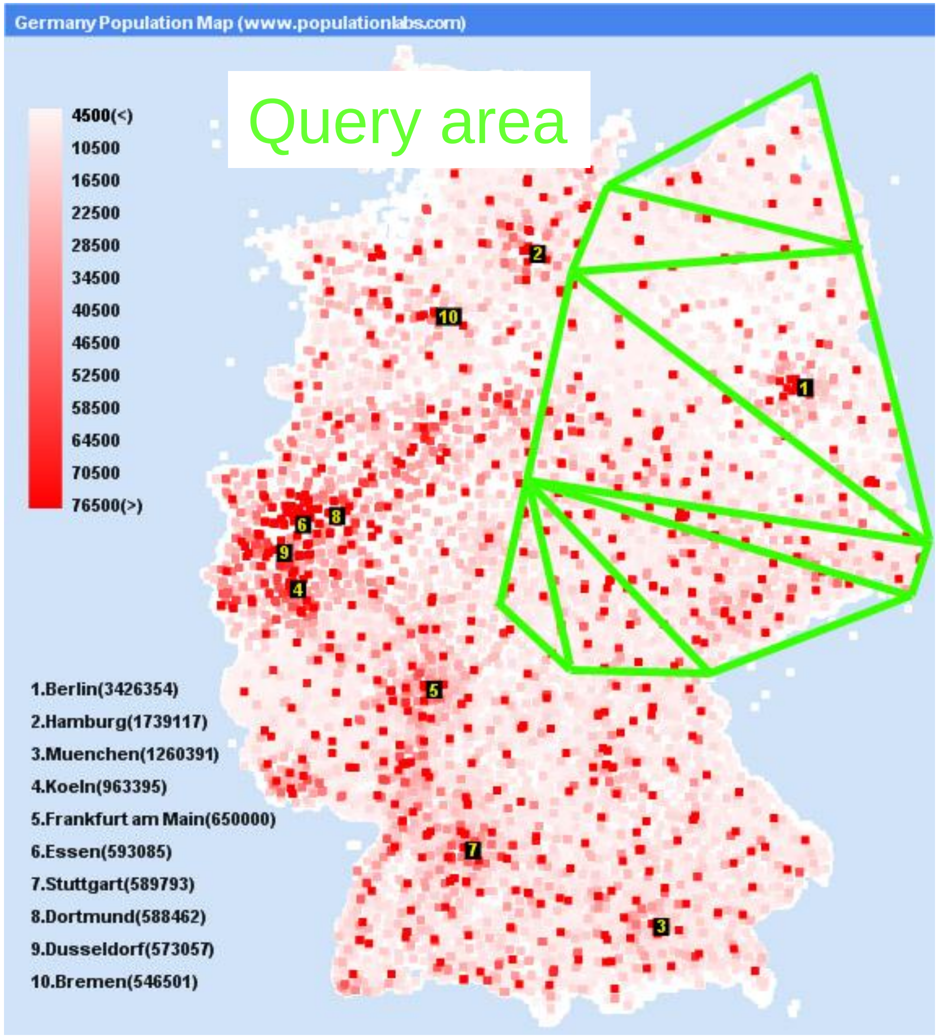
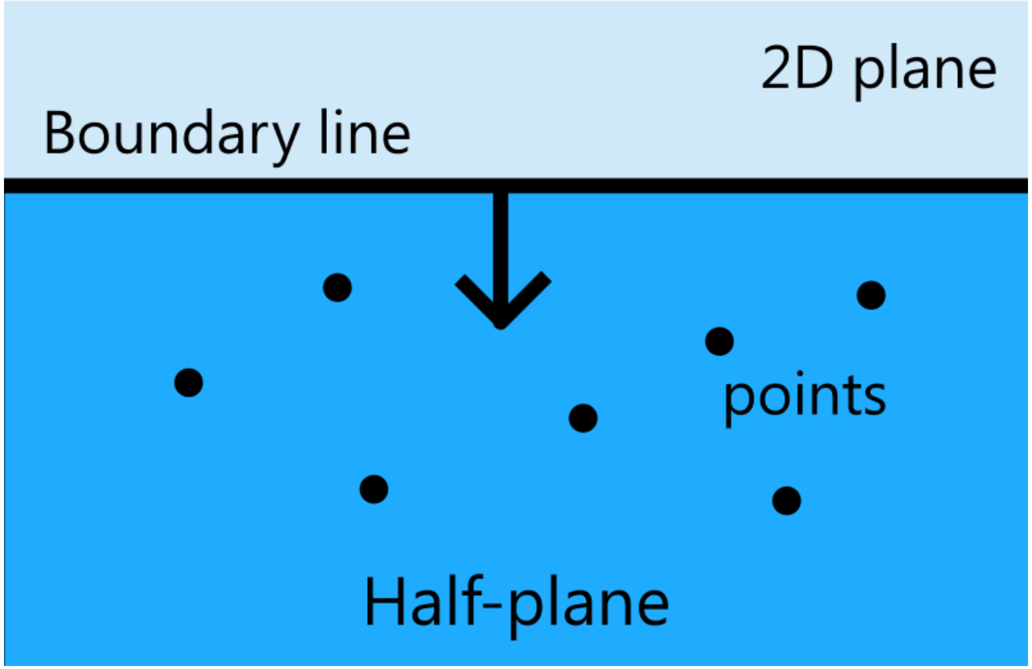


Image source: http://www.populationlabs.com/

Idea of algorithm 1- half-plane search

- First, we simplify **triangluar range searching problem** to **half-plane range searching problem**
- Triangle is intersection of three half-planes
- We will convert it back to triangular range searching later
- **Half-plane range searching problem**
  - **Simply count points below a boundary line of half plane**



Idea of algorithm 2 – dual plane

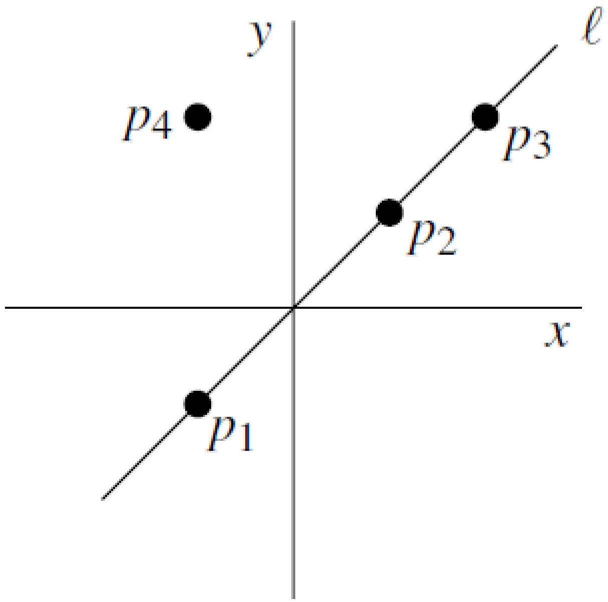
- To achieve better than O($\sqrt{n}$) query time (partition tree) we cannot use simplical partitions
- We solve half-plane range search in dual plane
- Duality transform:
  - Maps points in primal plane to lines in dual plane
  - Points has 2 parameters (X and Y position), line has also two parameters (Slope and intersection with Y axis)
  - Several mappings exist
  - Our case: property of such transformation must preserve order in a way that if points in primal plane lie above query line, then they (transormed to lines) lie below query point in dual plane.
- We count lines lying below query point in dual plane

Idea of algorithm 2 – dual plane

Simple duality transform: Transform point $[p_x, p_y]$ to line expressed in slope–intercept form $y = k * x + q$

$$y = p_x * x - p_y$$

Primal plane



Dual plane

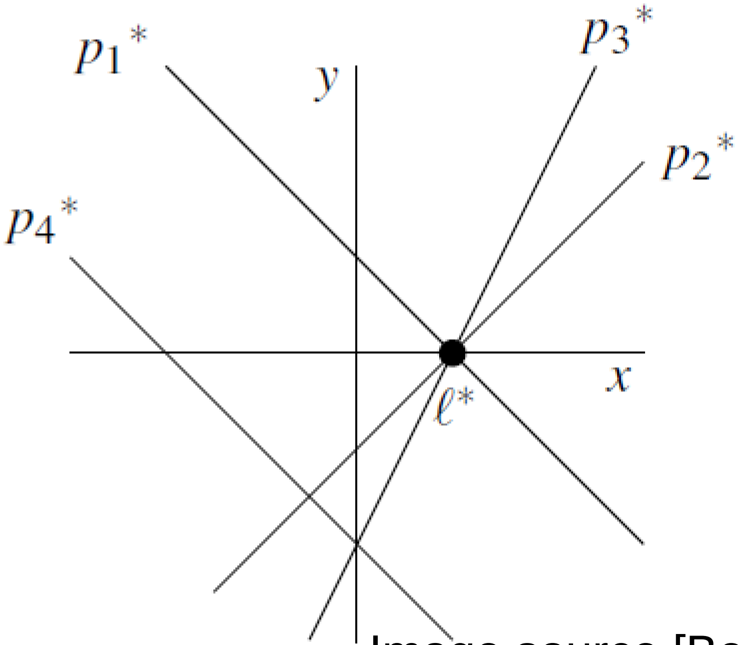


Image source [Berg]

Idea of algorithm 3 – counting lines quickly

1. We construct $\frac{1}{r}$ **cutting** of lines in plane
   - **$\frac{1}{r}$ cutting: set of triangles that together cover the plane with property: No triangle is crossed by more than $\frac{n}{r}$ lines.**
2. We preprocess it for lines counting – we store number of lines below (above) in each triangle.
   - We only need to count lines in triangle containing our query point
   - From previous we know that we need to count $\frac{n}{r}$ lines at max

Idea of algorithm 3 – counting lines quickly

- **1/r cutting example of 6 lines, chosen r = 2. No triangle is crossed by more than $\frac{n=6}{r=2}$ lines.**

Thin lines:
lines we are counting

Thick lines:
created 1/r cutting

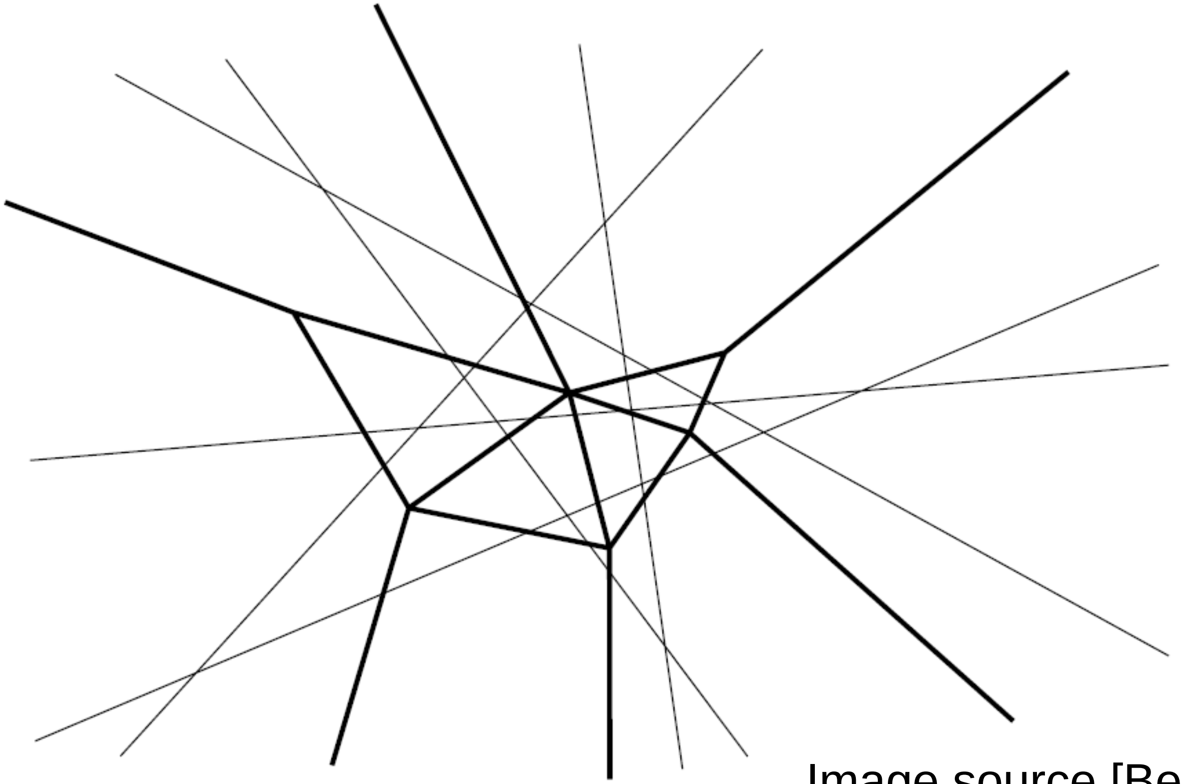


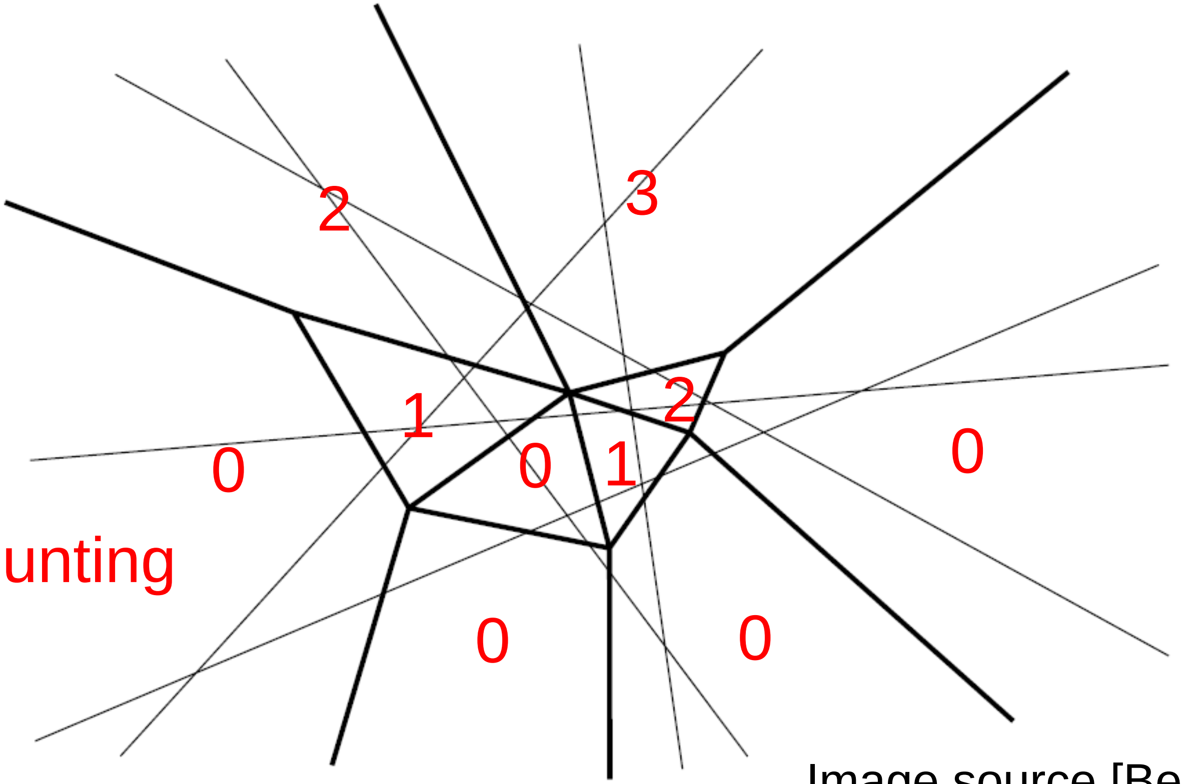
Image source [Berg]

Idea of algorithm 3 – counting lines quickly

- **1/r cutting example of 6 lines, chosen r = 2. No triangle is crossed by more than $\frac{n=6}{r=2}$ lines.**

Thin lines:
lines we are counting

Thick lines:
created 1/r cutting



Preprocessed for counting

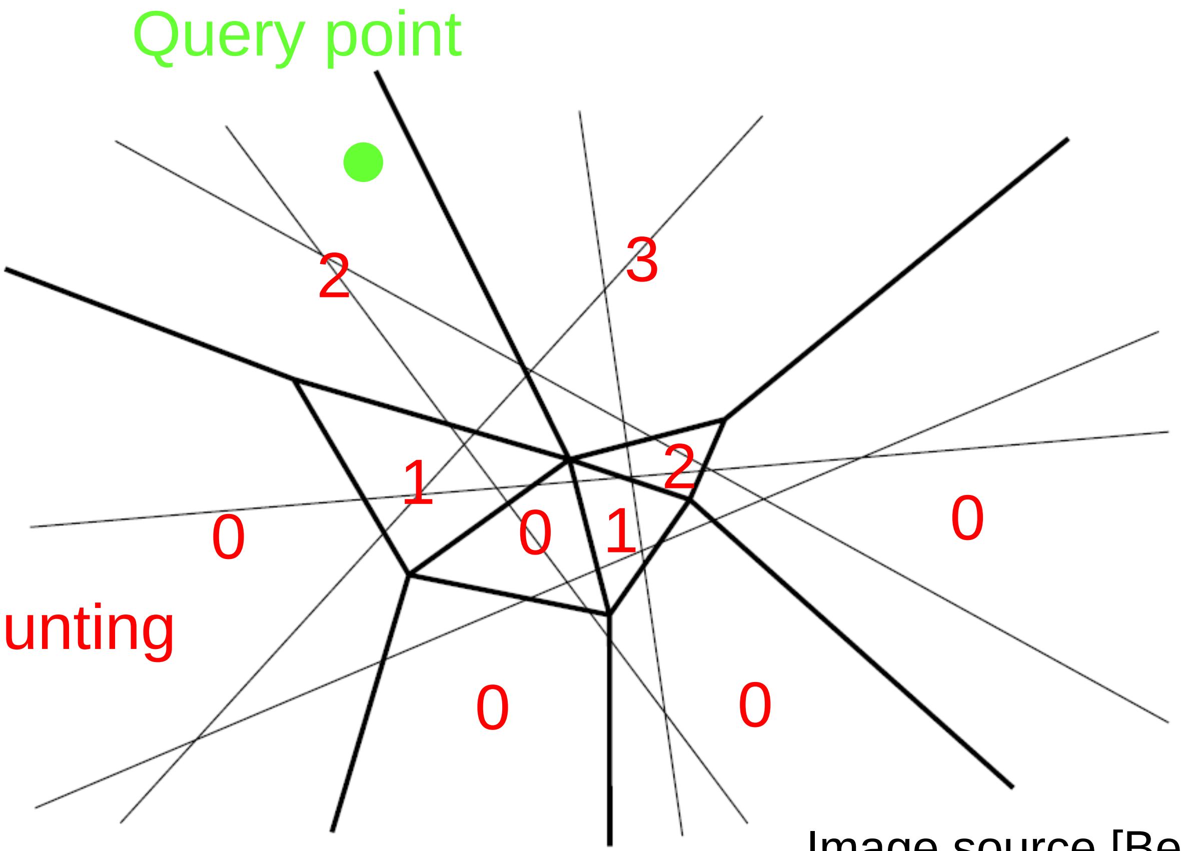
Image source [Berg]

Idea of algorithm 3 – counting lines quickly

- **1/r cutting example of 6 lines, chosen r = 2. No triangle is crossed by more than $\frac{n=6}{r=2}$ lines.**

Thin lines:
lines we are counting

Thick lines:
created 1/r cutting



Preprocessed for counting

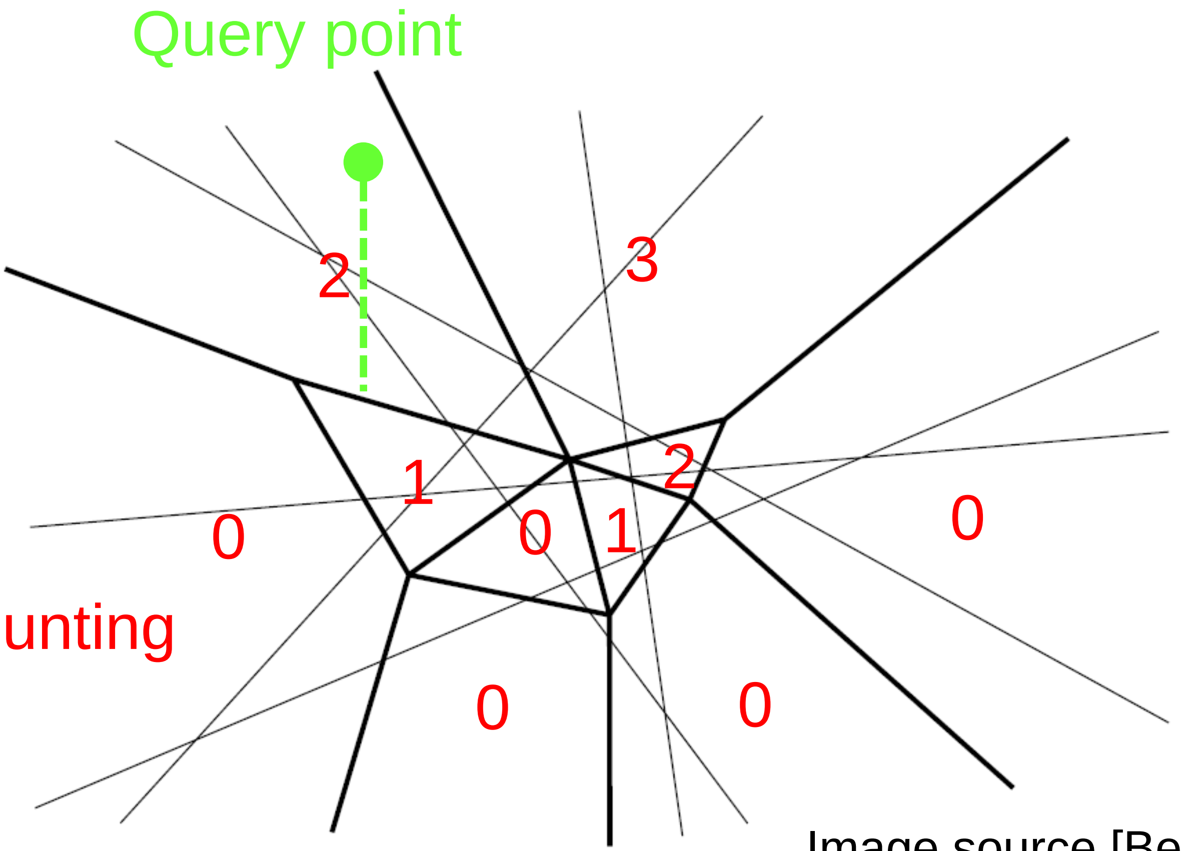
Image source [Berg]

Idea of algorithm 3 – counting lines quickly

- **1/r cutting example of 6 lines, chosen r = 2. No triangle is crossed by more than $\frac{n=6}{r=2}$ lines.**

Thin lines:
lines we are counting

Thick lines:
created 1/r cutting



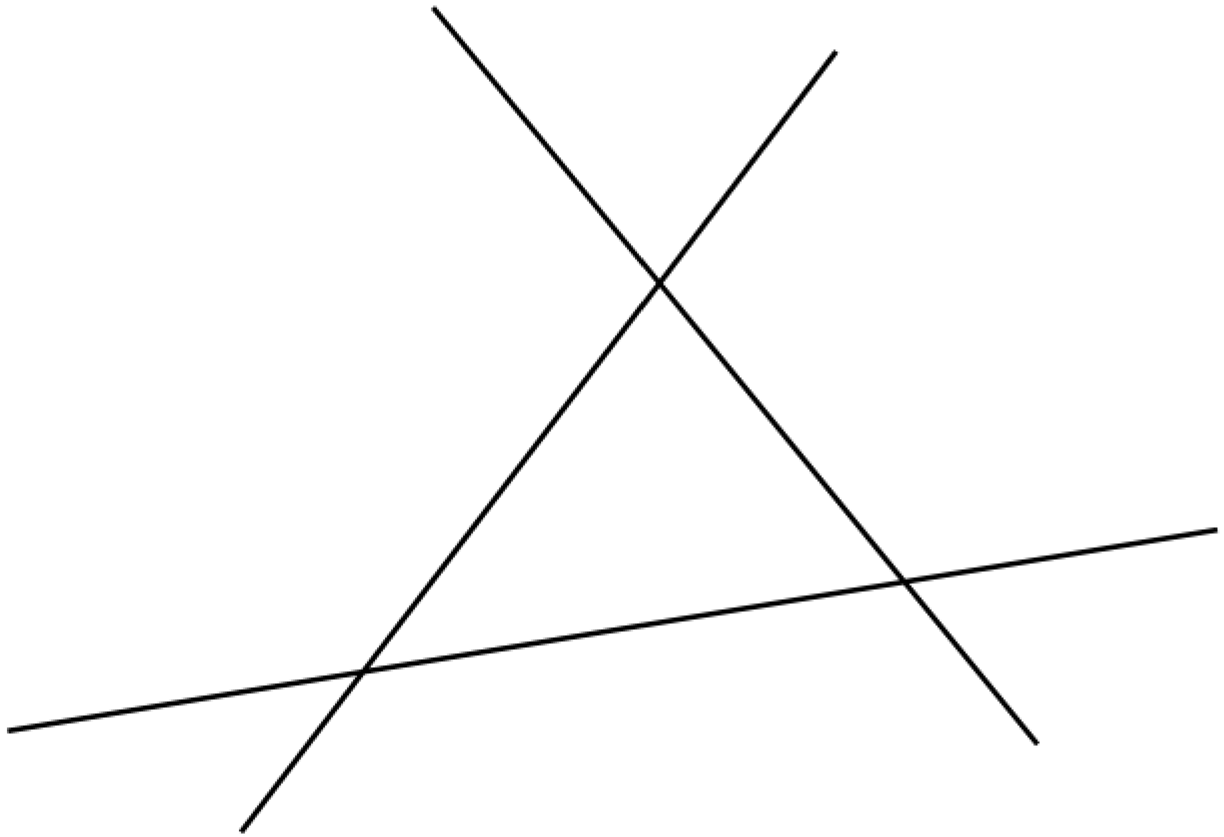
Preprocessed for counting

Image source [Berg]

Idea of algorithm 4 – counting lines in log(n)

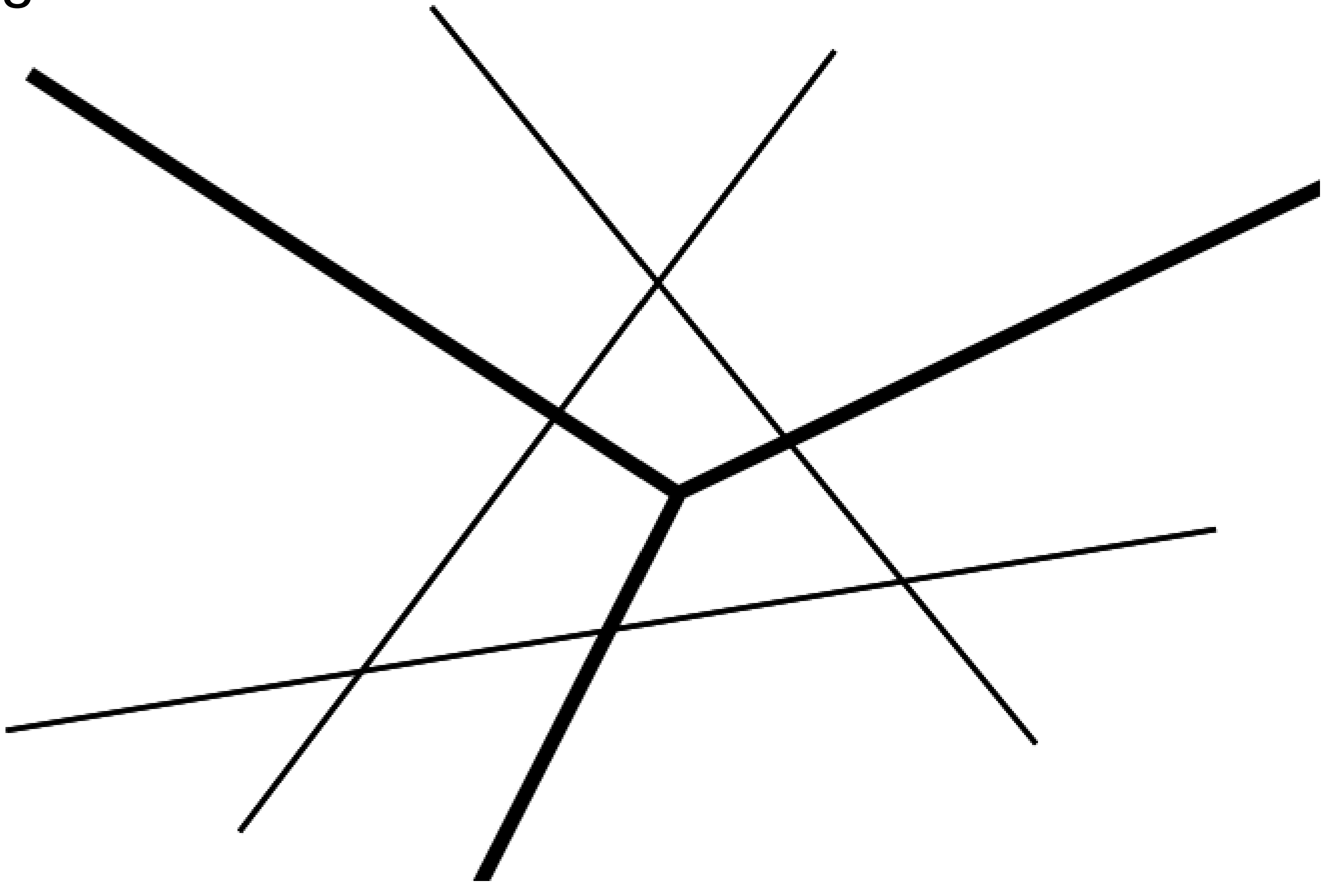
- We construct a tree of 1/r – cuttings.... a cutting tree.
  - Construct 1/r – cutting of whole plane. Created triangles are children of root
  - Take triangles which cross more than 1 line and construct 1/r cutting within them. Triangles in this second level cutting become children of corresponding triangle
  - Continue until all leaves cross only one line
- In such a structure we
  1. find triangle (one of root’s children) which contains our query,
  2. in log(n) time we traverse to corresponding leaf and sum points lying below these triangles
  3. test only one line within that leaf.

Idea of algorithm 4 – counting lines in log(n)



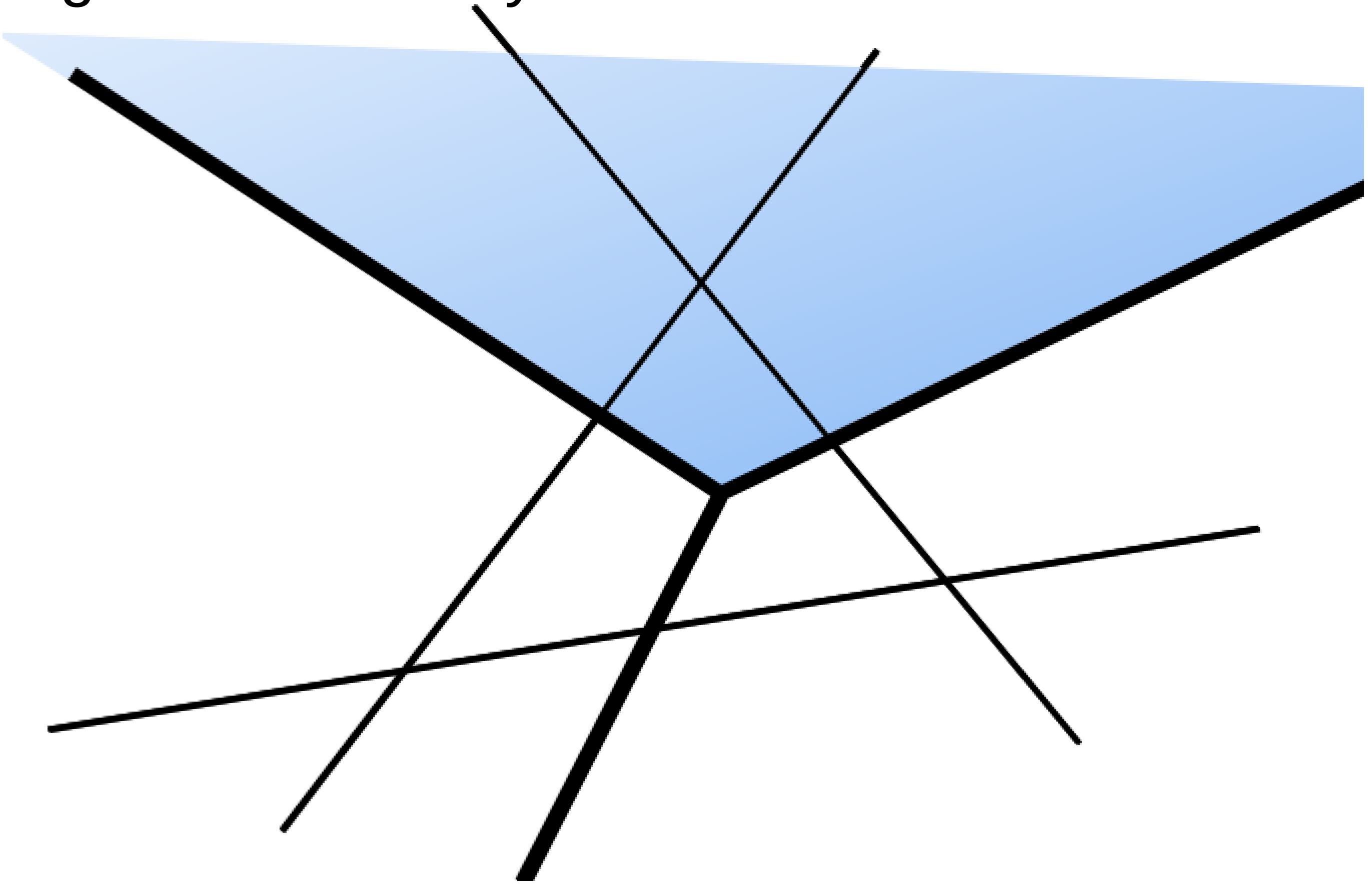
Idea of algorithm 4 – counting lines in log(n)

1/r cutting constructed



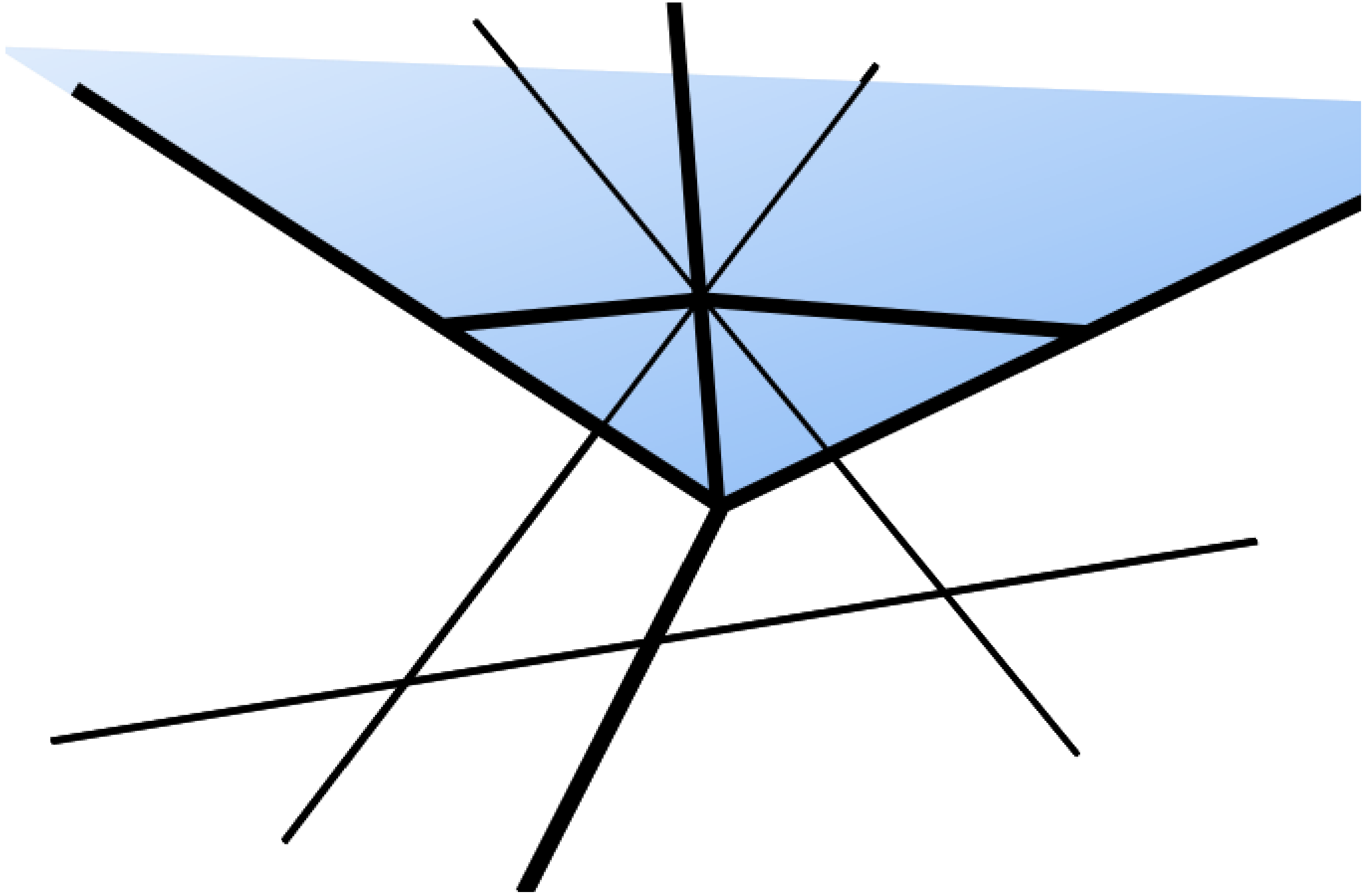
Idea of algorithm 4 – counting lines in log(n)

This triangle is crossed by more than 1 line



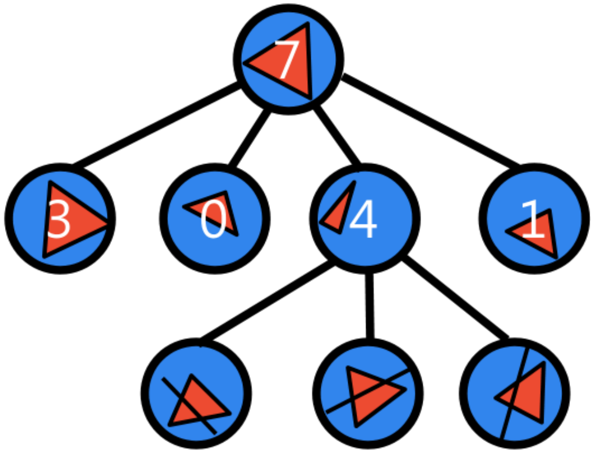
Idea of algorithm 4 – counting lines in log(n)

1/r cutting constructed within this triangle.



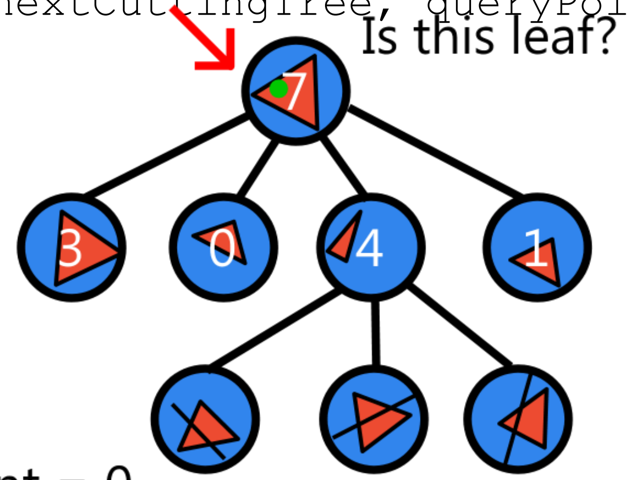
Conclusion - query algorithm

```
countBelowLines(cuttingTree, queryPoint) {
    int count = 0;
    if (cuttingTree.isSingleLeaf()) {
        if („line in leaf is below query point“) count++;
    } else {
        for each („child of root in cuttingTree“) {
            nextCuttingTree= „child that contains queryPoint“;
        }
        count += nextCuttingTree.belowLines +
Recursion is here ->  countBelowLines(nextCuttingTree, queryPoint);
    }
    return count;
}
```



Conclusion - query algorithm

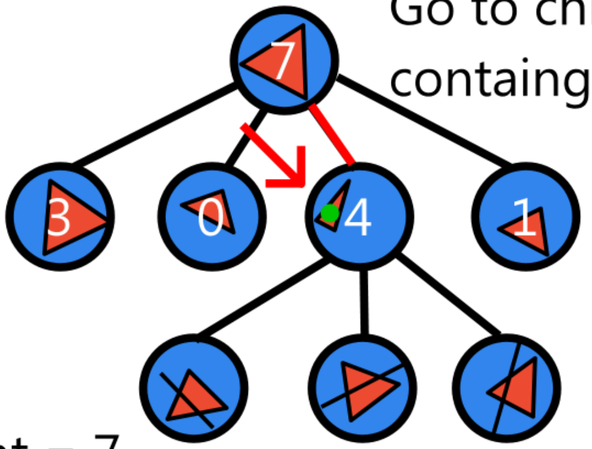
```
countBelowLines(cuttingTree, queryPoint) {
    int count = 0;
    if (cuttingTree.isSingleLeaf()) { <- We are here
        if („line in leaf is below query point“) count++;
    } else {
        for each („child of root in cuttingTree“) {
            nextCuttingTree= „child that contains queryPoint“;
        }
        count += nextCuttingTree.belowLines +
                 countBelowLines(nextCuttingTree, queryPoint);
    }
    return count;
}
```



Count = 0

Conclusion - query algorithm

```
countBelowLines(cuttingTree, queryPoint) {
    int count = 0;
    if (cuttingTree.isSingleLeaf()) {
        if („line in leaf is below query point“) count++;
    } else {
        for each („child of root in cuttingTree“) {
            nextCuttingTree= „child that contains queryPoint“;
        }
        count += nextCuttingTree.belowLines +
  We are here -> countBelowLines(nextCuttingTree, queryPoint);
    }
    return count;
}
```



Go to child containg query

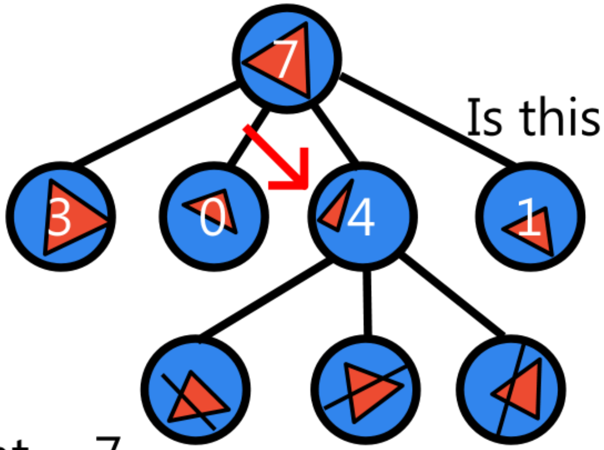
Count = 7

Increment count by 7

Conclusion - query algorithm

```
countBelowLines(cuttingTree, queryPoint) {
    int count = 0;
    if (cuttingTree.isSingleLeaf()) {
        if („line in leaf is below query point“) count++;
    } else {
        for each („child of root in cuttingTree“) {
            nextCuttingTree= „child that contains queryPoint“;
        }
        count += nextCuttingTree.belowLines +
                 countBelowLines(nextCuttingTree, queryPoint);
    }
    return count;
}
```

<- We are here

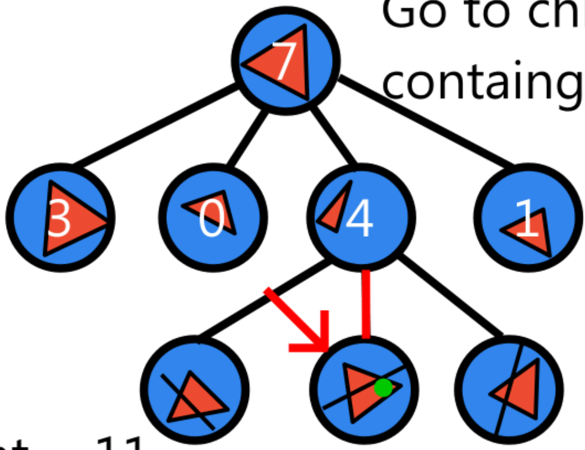


Is this leaf?

Count = 7

Conclusion - query algorithm

```
countBelowLines(cuttingTree, queryPoint) {
    int count = 0;
    if (cuttingTree.isSingleLeaf()) {
        if („line in leaf is below query point“) count++;
    } else {
        for each („child of root in cuttingTree“) {
            nextCuttingTree= „child that contains queryPoint“;
        }
        count += nextCuttingTree.belowLines +
 We are here -> countBelowLines(nextCuttingTree, queryPoint);
    }
    return count;
}
```



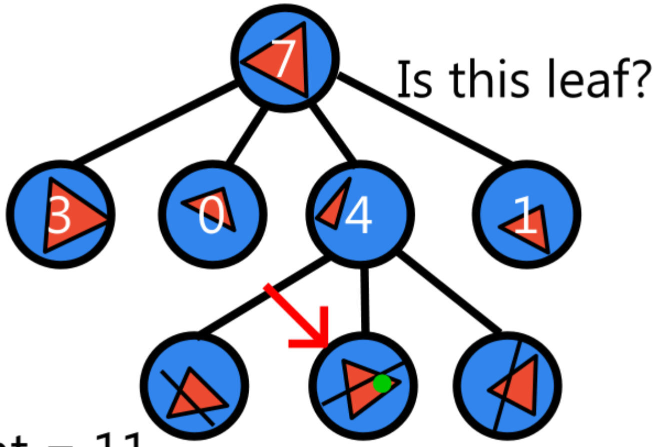
Go to child containg query

Count = 11

Increment count by 4

Conclusion - query algorithm

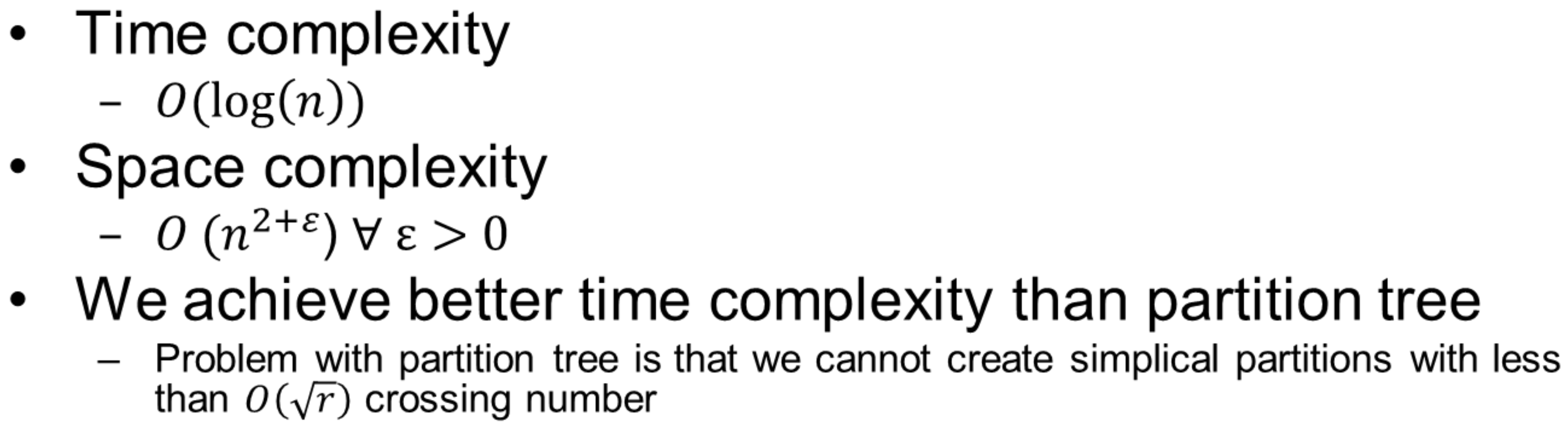
```
countBelowLines(cuttingTree, queryPoint) {
    int count = 0;
    if (cuttingTree.isSingleLeaf()) {
We are here if („line in leaf is below query point“) count++;
    } else {
        for each („child of root in cuttingTree“) {
            nextCuttingTree= „child that contains queryPoint“;
        }
        count += nextCuttingTree.belowLines +
                 countBelowLines(nextCuttingTree, queryPoint);
    }
    return count;
}
```



Count = 11

Check line and increment if below

Efficiency of cutting tree



- Time complexity
  - $O(\log(n))$
- Space complexity
  - $O\left(n^{2+\varepsilon}\right)\ \forall\ \varepsilon > 0$
- We achieve better time complexity than partition tree
  - Problem with partition tree is that we cannot create simplical partitions with less than $O(\sqrt{r})$ crossing number

- In each level of cutting tree our query intersects only one triangle, so we recursively visit only one child. (As opposed to partition tree, where line can intersect many triangles)

How to query triangle instead of half-plane

- Each node of our tree contains information about lines below it – we call these lines a **canonical subset**
- The information that we store about a **canonical subset** does not have to be a single number, like its cardinality. We can also store the elements of the canonical subset in another cutting tree
- Doing three levels of cutting trees in each node – we can query three times in a row – **once for each half-plane of triangle.**
- **Each query reduces set of possibly reported points**, similiar to branch and bound algorithms.
- After three queries we have selected a set of points of queried triangle
- We report numbert of these points
- Drawing on blackboard?

Literature

[Berg] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars: Computational Geometry: Algorithms and Applications, Springer-Verlag, 3rd rev. ed. 2008. 386 pages, 370 fig. ISBN: 978-3-540-77973-5, Chapters 3 and 9, http://www.cs.uu.nl/geobook/

Any questions?

OI-OPPA. European Social Fund
Prague & EU: We invest in your future.