

دانشگاه آزاد اسلامی واحد تبریز



نام درس: طراحی و تحلیل الگوریتم‌های پیشرفته

بخش: بازه جستجو در محیط‌های دوبعدی

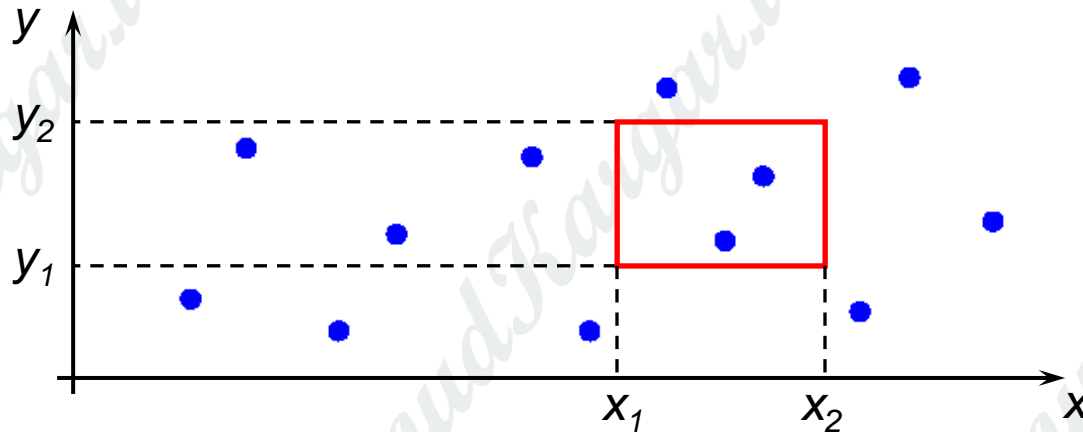
نام استاد: دکتر مسعود کارگر

# Range Searching in 2D

- Main goals of the lecture:
  - to understand and to be able to analyze
    - the **kd-trees** and the **range trees**;
  - to see how data structures can be used to trade the space used for the running time of queries

# Range queries

- How do you efficiently find points that are inside of a rectangle?
  - **Orthogonal range query** ( $[x_1, x_2], [y_1, y_2]$ ): find all points  $(x, y)$  such that  $x_1 < x < x_2$  and  $y_1 < y < y_2$
  - Useful also as a multi-attribute database query

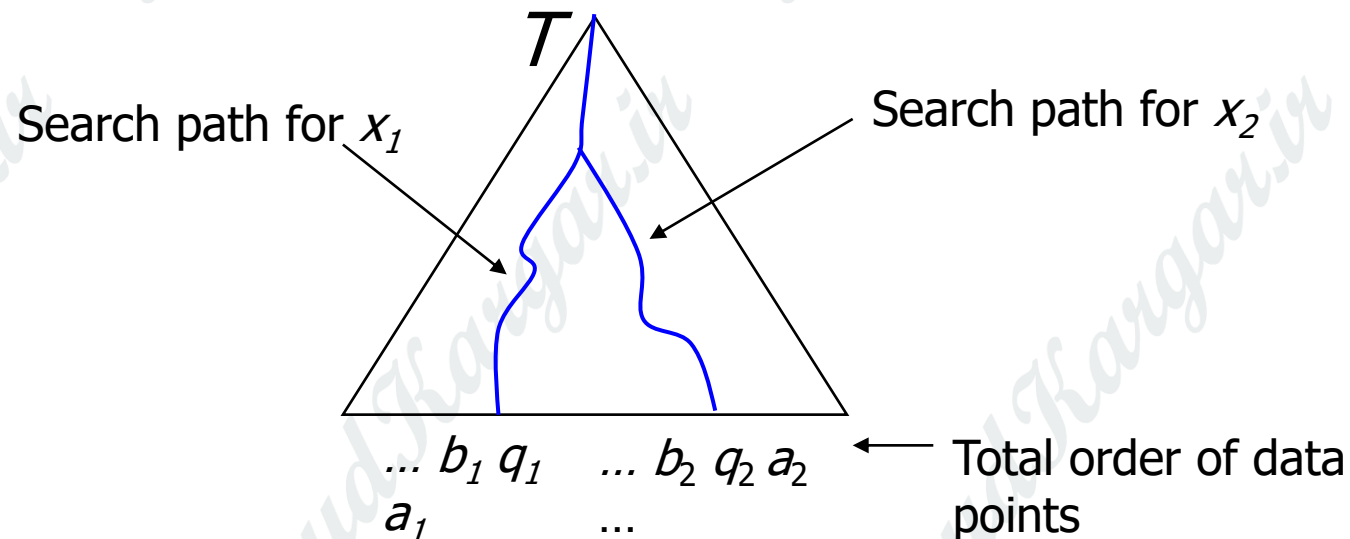


# Preprocessing

- How much time such a query would take?
- Rules of the game:
  - We **preprocess** the data into a data structure
  - Then, we perform queries and updates on the data structure
  - Analysis:
    - Preprocessing time
    - Efficiency of queries (and updates)
    - The size of the structure
  - Assumption: no two points have the same x- coordinate (the same is true for y-coordinate).

# 1D range query

- How do we do a 1D range query  $[x_1, x_2]$ ?
  - Balanced BST where all data points are stored in the leaves
    - The size of it?
  - Where do we find the answer to a query?



# 1D range query

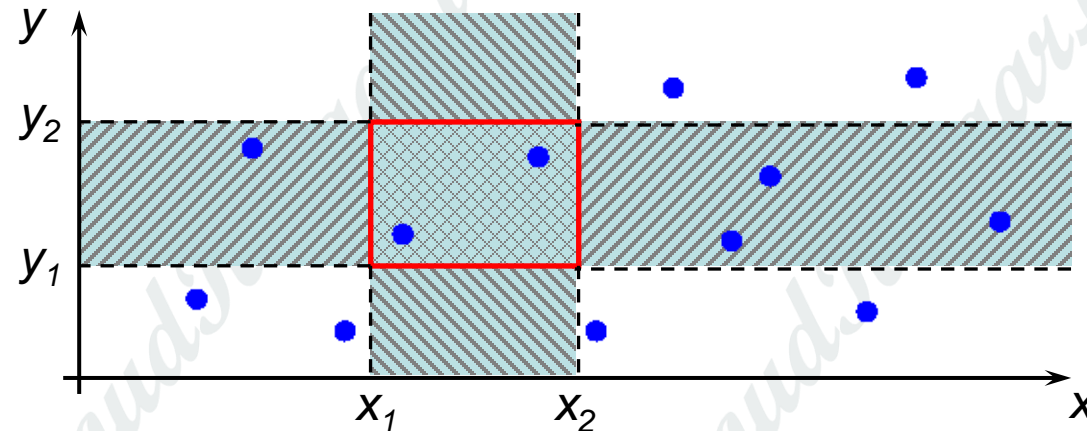
- How do we find all these leaf nodes?
  - A possibility: have a linked list of leaves and traverse from  $q_1$  to  $q_2$ 
    - but, will not work for more dimensions...
  - Sketch of the algorithm:
    - Find **the split node**
    - Continue searching for  $x_1$ , report all right-subtrees
    - Continue searching for  $x_2$ , report all left-subtrees
    - When leaves  $q_1$  and  $q_2$  are reached, check if they belong to the range
  - Why is this correct?

# Analysis of 1D range query

- What is the worst-case running time of a query?
  - It is output-sensitive: two traversals down the tree plus the  $O(k)$ , where  $k$  is the number of reported data points:  $O(\log n + k)$
- What is the time of construction?
  - Sort, construct by dividing into two, creating the root and conquering the two parts recursively
  - $O(n \log n)$
- Size:  $O(n)$

# 2D range query

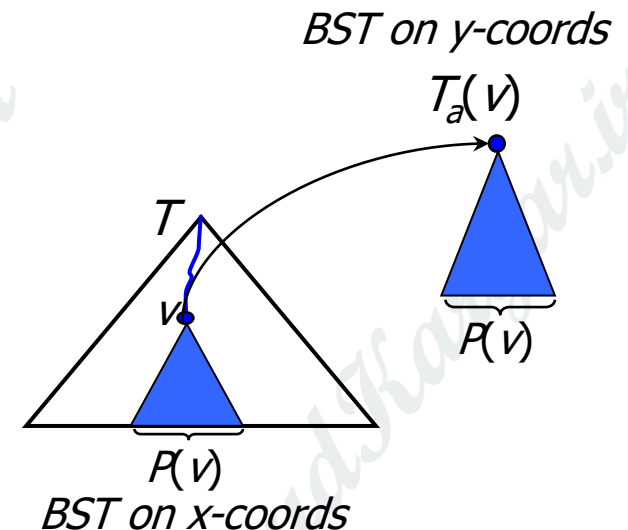
- How can we solve a 2D range query?
  - Observation – 2D range query is a conjunction of two 1D range queries:  $x_1 < x < x_2$  **and**  $y_1 < y < y_2$
  - Naïve idea:
    - have two BSTs (on x-coordinate and on y-coordinate)
    - Ask two 1D range queries
    - Return the intersection of their results
  - What is the worst-case running time (and when does it happen)? Is it output-sensitive?





# Range tree

- Idea: when performing search on x-coordinate, we need to start filtering points on y-coordinate earlier!
  - **Canonical subset**  $P(v)$  of a node  $v$  in a BST is a set of points (leaves) stored in a subtree rooted at  $v$
  - **Range tree** is a multi-level data structure:
    - The main tree is a BST  $T$  on the x-coordinate of points
    - Any node  $v$  of  $T$  stores a pointer to a BST  $T_a(v)$
    - (**associated structure** of  $v$ ), which stores canonical subset  $P(v)$  organized on the y-coordinate
    - 2D points are stored in all leaves!



# Querying the range tree

- How do we query such a tree?
  - Use the 1DRangeSearch on  $T$ , but replace ReportSubtree( $w$ ) with 1DRangeSearch( $T_a(w)$ ,  $y_1$ ,  $y_2$ )
- What is the worst-case running time?
  - Worst-case: We query the associated structures on all nodes on the path down the tree
  - On level  $j$ , the depth of the associated structure is  $\log \frac{n}{2^j} = \log n - j$
- Total running time:  $O(\log^2 n + k)$

# Size of the range tree

- What is the size of the range tree?
  - At each level of the main tree associated structures store all the data points once (with constant overhead) (Why?) :  $O(n)$
  - There are  $O(\log n)$  levels
  - Thus, the total size is  $O(n \log n)$

# Building the range tree

- How do we efficiently build the range tree?
  - Sort the points on  $x$  and on  $y$  (two arrays:  $X, Y$ )
  - Take the median  $v$  of  $X$  and create a root, build its associated structure using  $Y$
  - Split  $X$  into sorted  $X_L$  and  $X_R$ , split  $Y$  into sorted  $Y_L$  and  $Y_R$  (s.t. for any  $p \in X_L$  or  $p \in Y_L$ ,  $p.x < v.x$  and for any  $p \in X_R$  or  $p \in Y_R$ ,  $p.x \geq v.x$ )
  - Build recursively the left child from  $X_L$  and  $Y_L$  and the right child from  $X_R$  and  $Y_R$
- What is the running time of this?
  - $O(n \log n)$

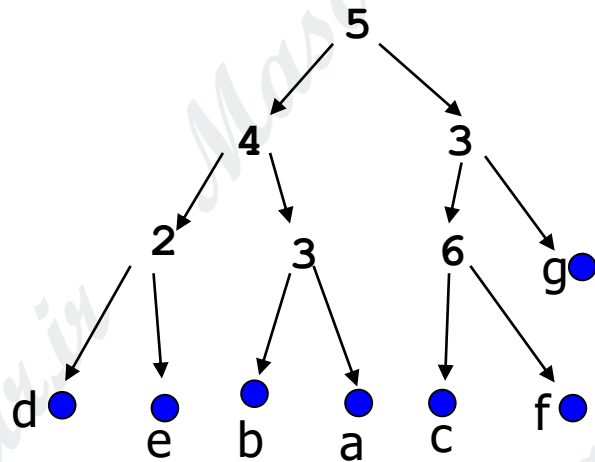
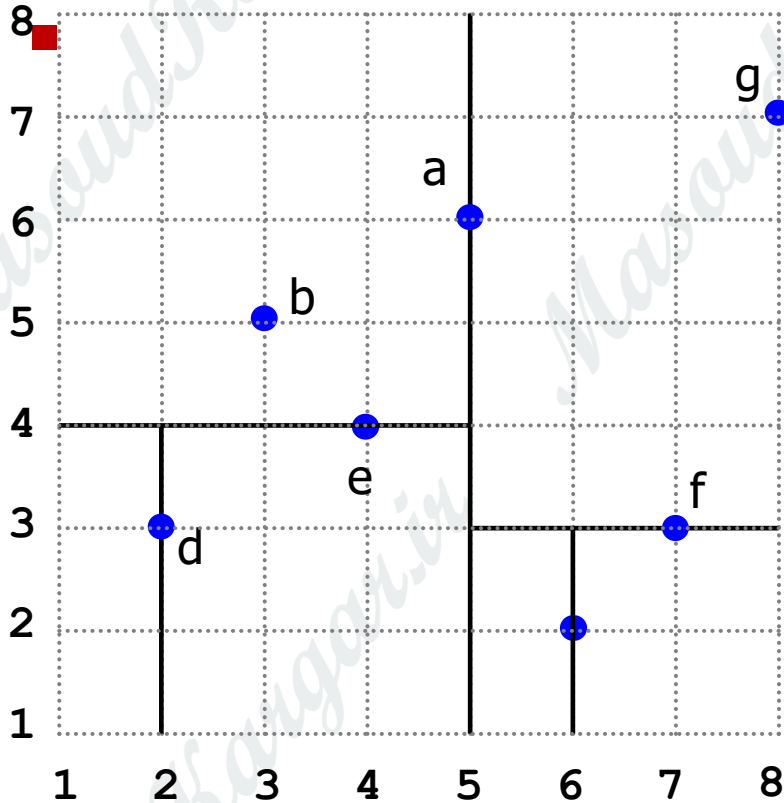
# Range trees: summary

- **Range trees**
  - Building (preprocessing time):  $O(n \log n)$
  - Size:  $O(n \log n)$
  - Range queries:  $O(\log^2 n + k)$
- Running time can be improved to  $O(\log n + k)$  without sacrificing the preprocessing time or size
  - Layered range trees (uses fractional cascading)
  - Priority range trees (uses priority search trees as associated structures)

# Kd-trees

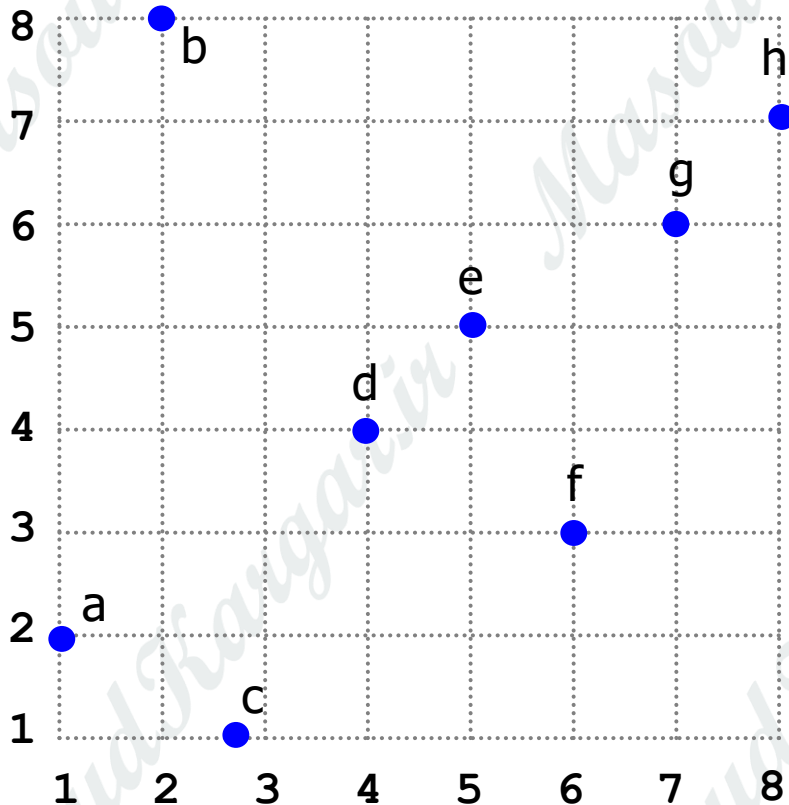
- What if we want linear space?
  - Idea: **partition trees** – generalization of binary search trees
  - Kd-tree: a binary tree
    - Data points are at leaves
    - For each internal node  $v$ :
      - $x$ -coords of left subtree  $\leq v < x$ -coords of right subtree, **if** depth of  $v$  is even (split with vertical line)
      - $y$ -coords of left subtree  $\leq v < y$ -coords of right subtree, **if** depth of  $v$  is odd (split with horizontal line)
  - Space:  $O(n)$  – points are stored once.

# Example kd-tree



# Draw a kd-tree

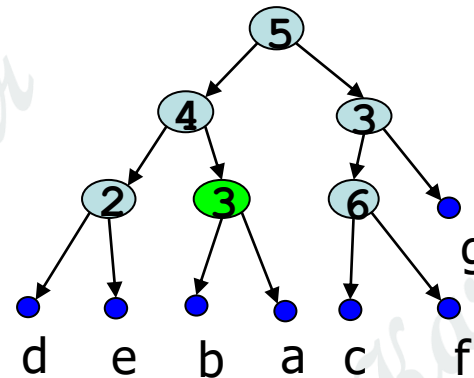
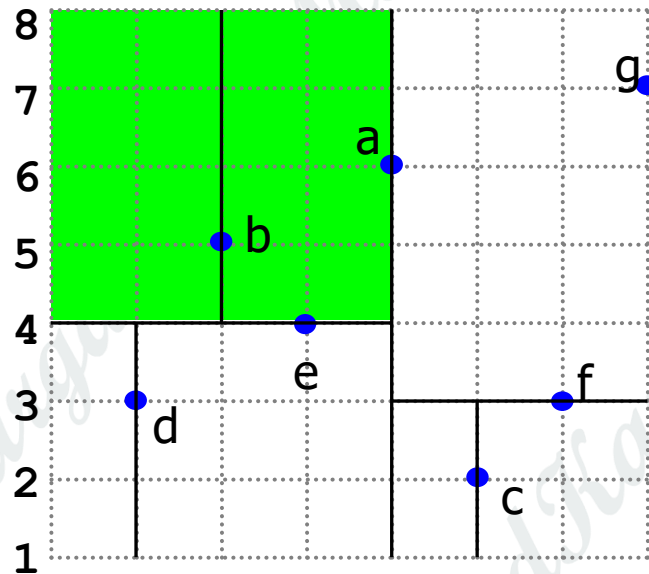
- Draw a kd-tree storing the following data points





# Querying the kd-tree

- How do we answer a range query?
  - Observation: Each internal node  $v$  corresponds to a region( $v$ ) (where all its children are included).
  - We can maintain region( $v$ ) as we traverse down the tree



# Querying the kd-tree

- The range query algorithm (query range  $R$ ):
  - If  $\text{region}(v)$  **does not intersect**  $R$ , do not go deeper into the subtree rooted at  $v$
  - If  $\text{region}(v)$  is **fully contained** in  $R$ , report all points in the subtree rooted at  $v$
  - If  $\text{region}(v)$  **only intersects** with  $R$ , go recursively into  $v$ 's children.

# Analysis of the search alg.

- What is the worst-case running time of the search?
  - Traversal of subtrees  $v$ , such that  $\text{region}(v)$  is fully contained in  $R$  adds up to  $O(k)$ .
  - We need to find the number of regions that intersect  $R$  – the regions which are crossed by some border of  $R$ 
    - As an upper bound for that, let's find how many regions a crossed by a vertical (or horizontal) line
    - What recurrence can we write for it?

$$T(n) = 2 + 2T(n/4)$$

Solution:  $O(\sqrt{n})$  Total time:  $O(\sqrt{n} + k)$

# Building the kd-tree

- How do we build the kd-tree?
  - Sort the points on  $x$  and on  $y$  (two arrays:  $X, Y$ )
  - Take the median  $v$  of  $X$  (if depth is even) or  $Y$  (if depth is odd) and create a root
  - Split  $X$  into sorted  $X_L$  and  $X_R$ , split  $Y$  into sorted  $Y_L$  and  $Y_R$ , s.t.
    - for any  $p \in X_L$  or  $p \in Y_L$ ,  $p.x < v.x$  (if depth is even) or  $p.y < v.y$  (if depth is odd)
    - for any  $p \in X_R$  or  $p \in Y_R$ ,  $p.x \geq v.x$  (if depth is even) or  $p.y \geq v.y$  (if depth is odd)
  - Build recursively the left child from  $X_L$  and  $Y_L$  and the right child from  $X_R$  and  $Y_R$
- What is the running time of this?
  - $O(n \log n)$

# Kd-trees: summary

- Kd-tree:
  - Building (preprocessing time):  $O(n \log n)$
  - Size:  $O(n)$
  - Range queries:  $O(\sqrt{n} + k)$

# Quadrees

- Quadtree – a four-way partition tree
  - **region** quadtrees vs. **point** quadtrees
    - kd-trees can also be point or region
  - Linear space
  - Good average query performance

