CS-509

PG Software Lab.

KD-Trees

Monsoon - 2023

Dr. Shashi Shekhar Jha
shashi@iitrpr.ac.in

Indian Institute of Technology Ropar

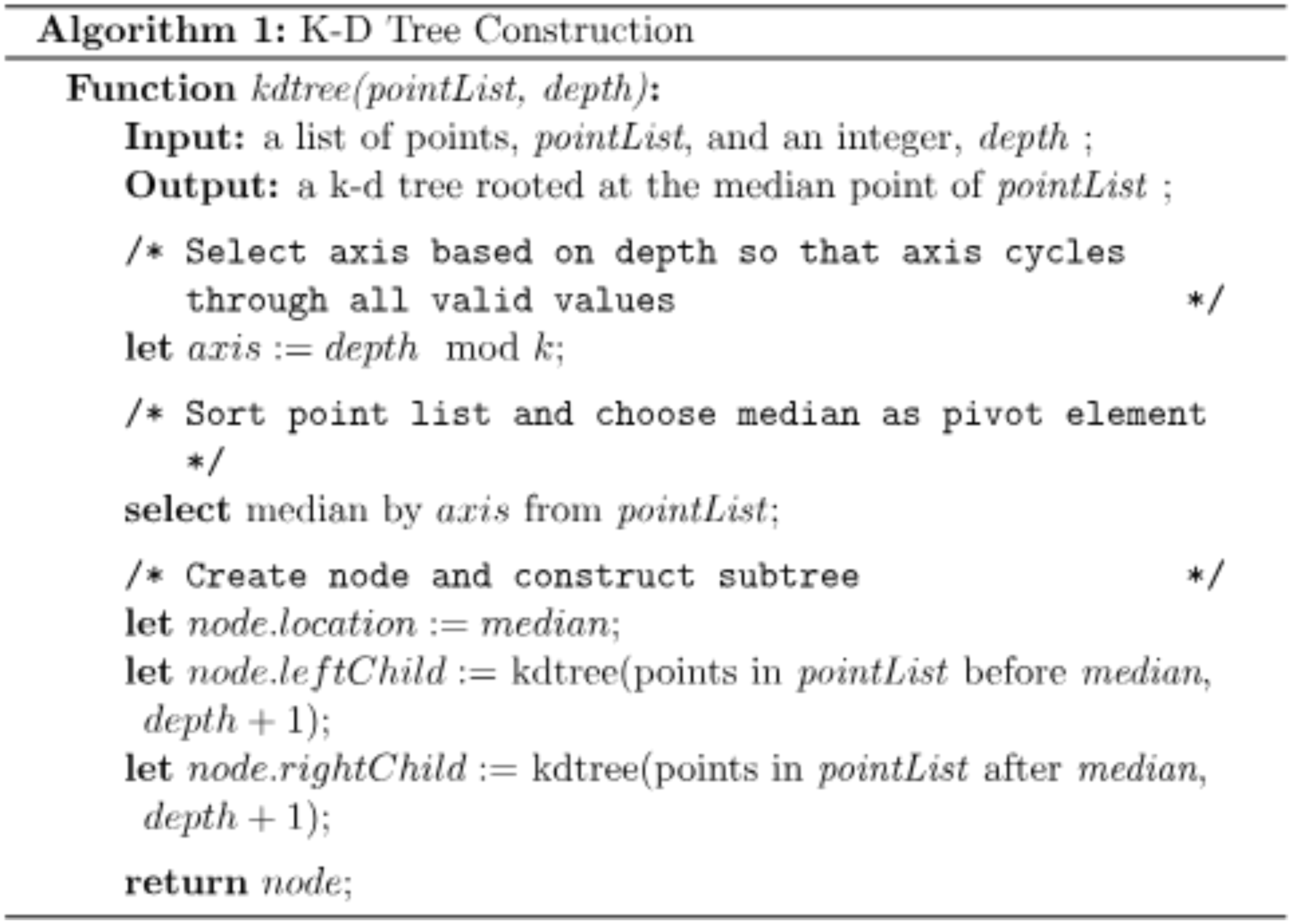
Python

- Class
- Attributes
- Constructor (requires ‘self’ as first arg)
- Method (requires ‘self’ as first arg)
- Inheritance : Pass parent class name while defining a class

KD Trees

- A K-D Tree (also called as K-Dimensional Tree) is a **binary search tree** where data in each node is a K-Dimensional point in space.
- A space partitioning data structure for organizing points in a K-Dimensional space.
- A non-leaf node acts as a hyper-plane that divides the space into two parts, called as half-spaces
- This hyperplane is perpendicular to the chosen axis, which is associated with one of the K dimensions
- There are different strategies for choosing an axis when dividing, but the most common one would be to cycle through each of the K dimensions repeatedly and **select a midpoint along it to divide the space**.

KD Tree Construction



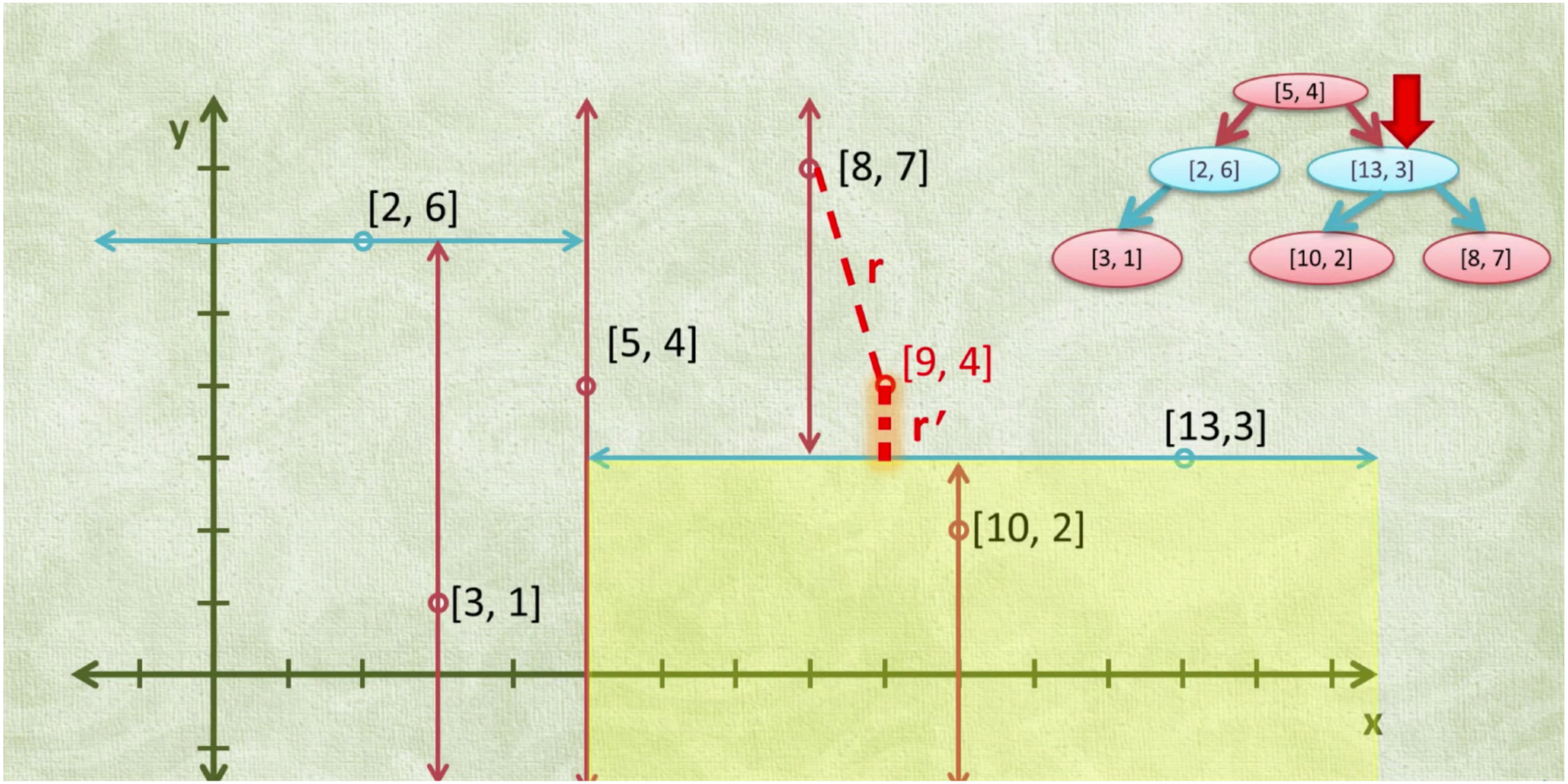
Algorithm 1: K-D Tree Construction

```
Function kdtree(pointList, depth):
    Input: a list of points, pointList, and an integer, depth ;
    Output: a k-d tree rooted at the median point of pointList ;

    /* Select axis based on depth so that axis cycles
       through all valid values                                   */
    let $axis := depth \mod k$;

    /* Sort point list and choose median as pivot element
       */
    select median by axis from pointList;

    /* Create node and construct subtree                          */
    let node.location := median;
    let node.leftChild := kdtree(points in pointList before median,
     $depth + 1$);
    let node.rightChild := kdtree(points in pointList after median,
     $depth + 1$);

    return node;
```

Nearest Neighbour Search

- K-D trees are widely used for nearest-neighbor searches, where the objective is to find the point in the tree that is closest to a given query point
- Traverse the tree and compare the distance between the query point and the points in each leaf node
- Starting at the root node, we recursively move down the tree until we reach a leaf node, following a similar process as when inserting a node
- At each level, we decide whether to go down the left or right subtree based on which side of the splitting hyperplane the query point lies
- Once we reach a leaf node, we compute the distance between the query point and that leaf node, and save it as the “current best”
- See whether the absolute distance between the splitting coordinate of the search point and the current node is lesser than the overall distance from the search point to the current best.
- If the hypersphere intersects the plane, there may be nearer points on the other side of the plane.

Nearest Neighbour Search



2-D Tree

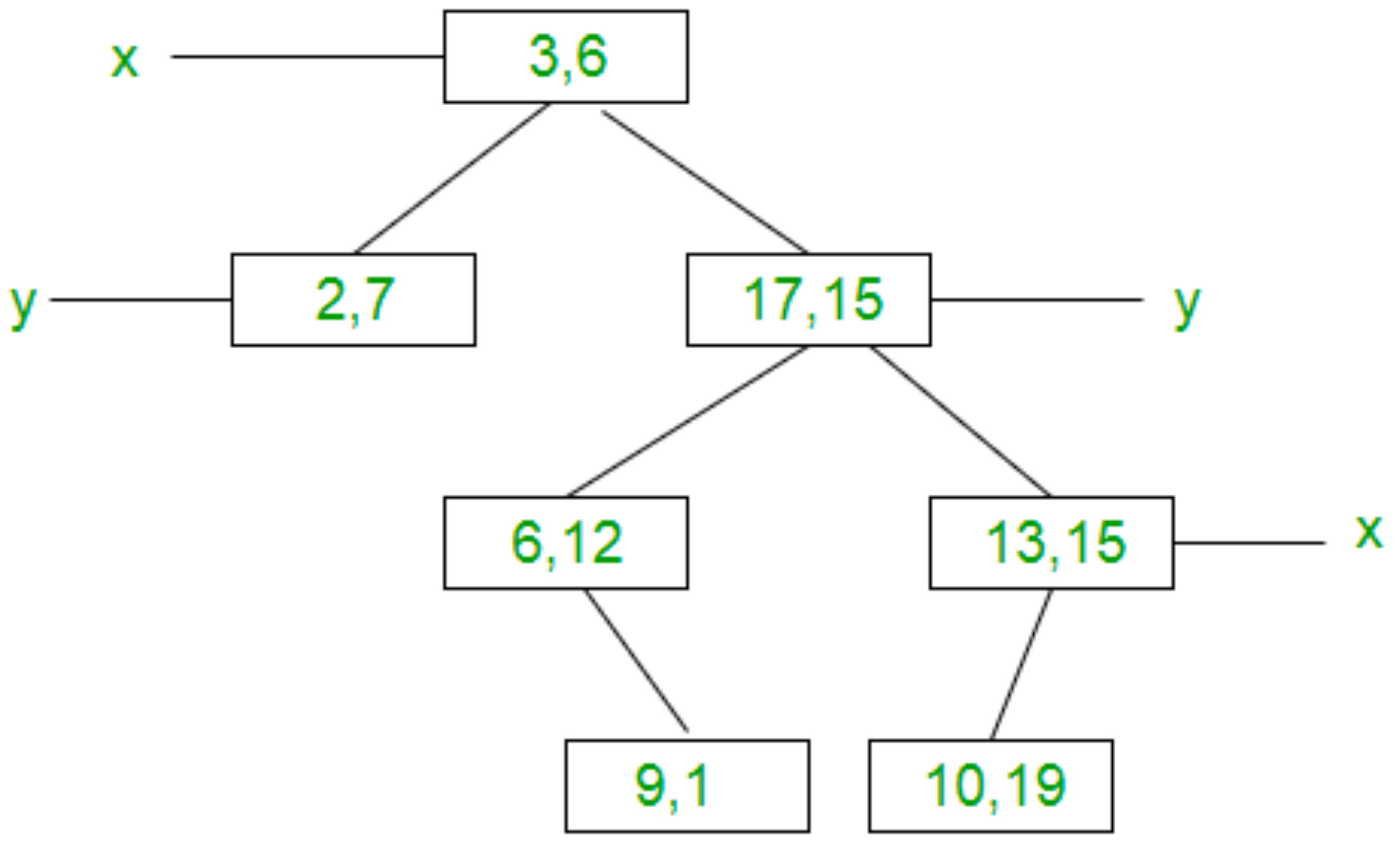
- Imagine 2D points over a plane
- The root of 2D tree would have an x-aligned plane,
- the root’s children would both have y-aligned planes,
- the root’s grandchildren would all have x-aligned planes,
- the root’s great-grandchildren would all have y-aligned planes and so on

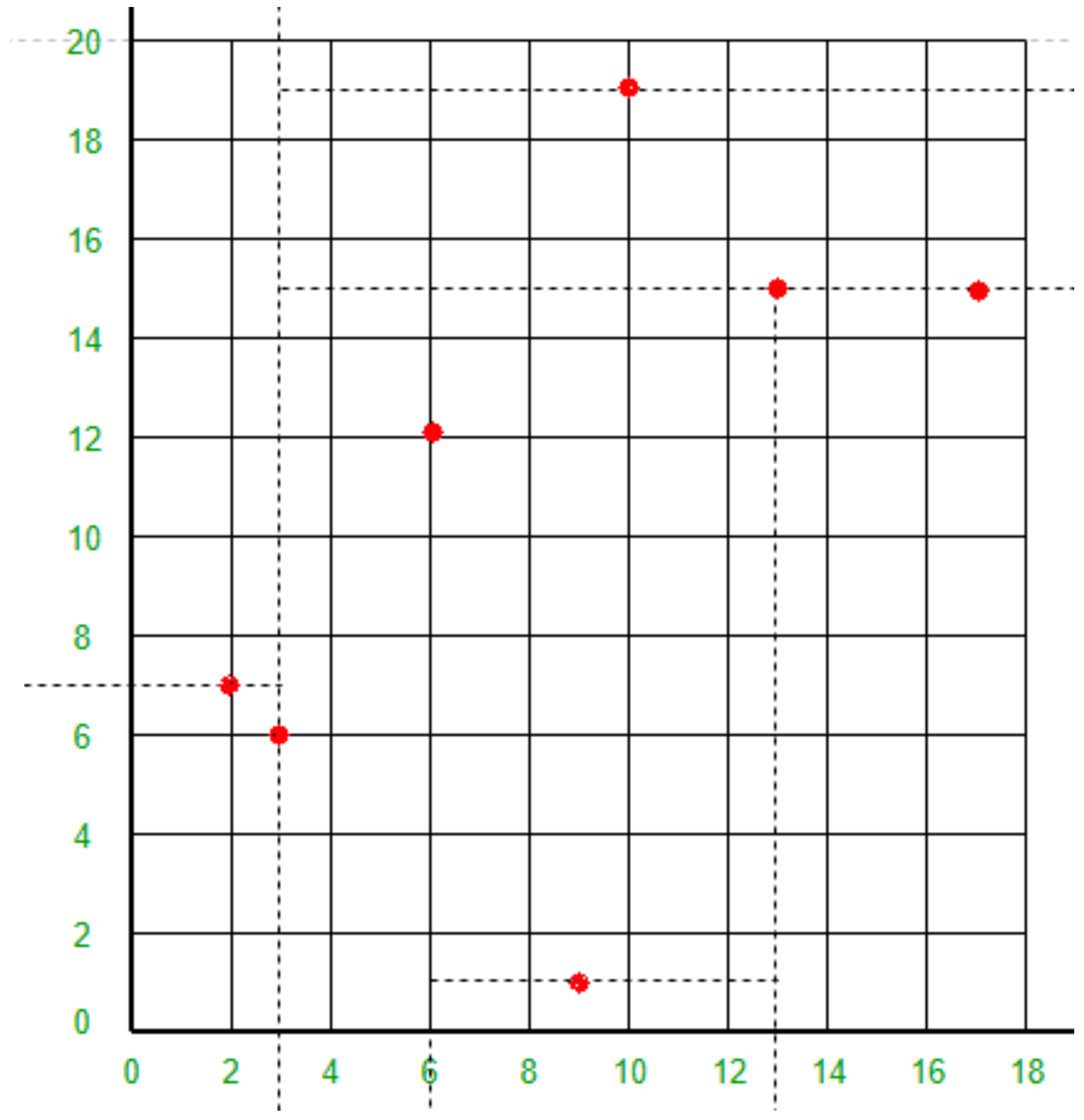
2-D Tree

- How to determine if a point will lie in the left subtree or in right subtree?
- If the root node is aligned in planeA, then the left subtree will contain all points whose coordinates in that plane are smaller than that of root node
- Similarly, the right subtree will contain all points whose coordinates in that plane are greater-equal to that of root node

2-D Tree

- Consider following points in a 2-D plane: (3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)


x
3,6
y
2,7
17,15
y
6,12
13,15
x
9,1
10,19


20
18
16
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
16
18