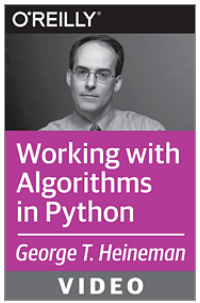
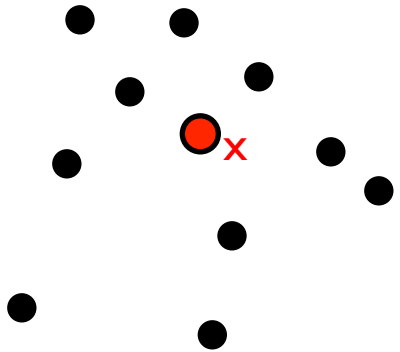

O'REILLY®

K-Dimensional Trees



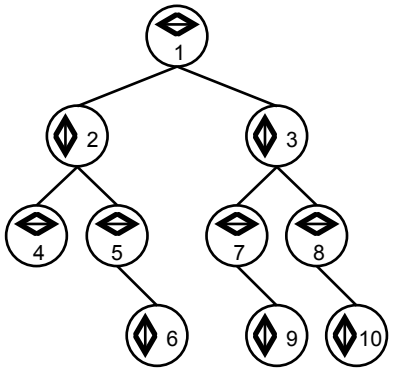
Two-Dimensional Graphical Problem

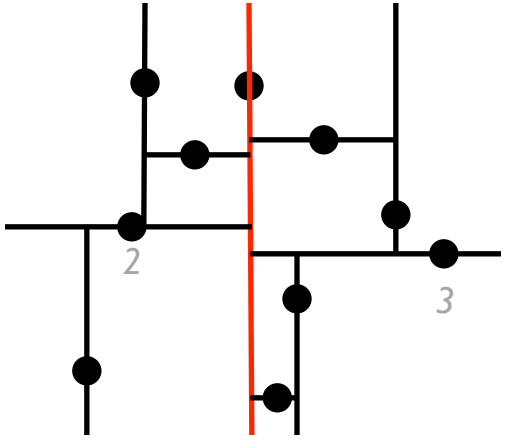
- Given a collection of 2-dimensional points
  - Which point is closest to query point x?
- Applications include
  - Evaluate coverage of fire stations
  - Big Data clustering analysis
- Obvious solution is O(n)
  - Can we do better?



Two-Dimensional Graphical Problem

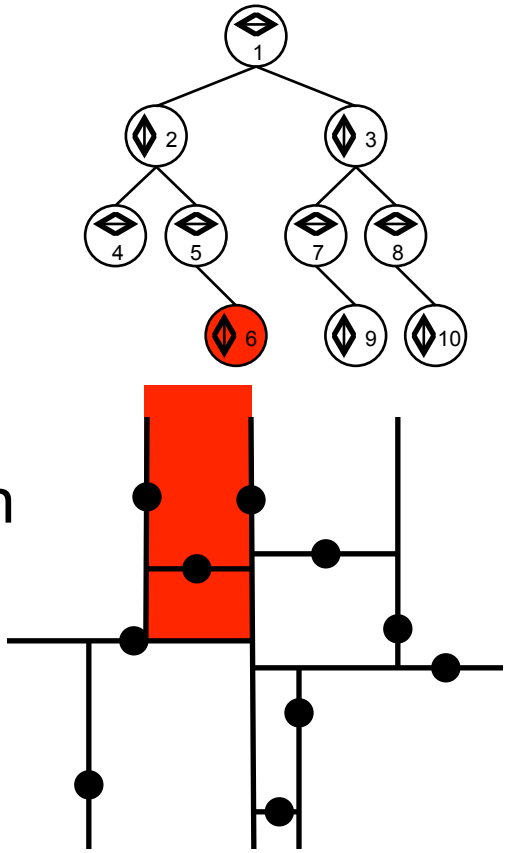
- To achieve O(log n) performance
  - Must find some way to divide points
  - Inspiration from Binary Search Tree
- KD-tree data structure
  - Nodes represents partitions of two-dimensional space
  - Easily extends to *k* dimensions





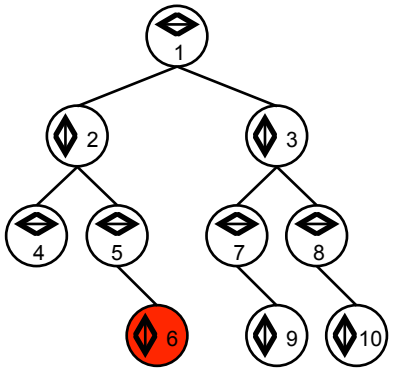
Two-Dimensional Node Structure

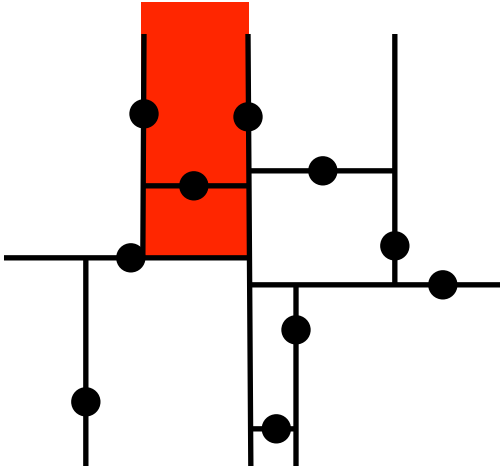
- Partition is rectangular region
- Root node “covers” infinite region
  - Child node covers half of parent’s region
  - Alternating levels “flip” orientation
- Hilighted region associated with P_6
  - Contains point
  - Prepared to subdivide further



Two-Dimensional Node Structure

- KD-tree has Binary Tree Structure
  - HORIZONTAL or VERTICAL orientation
  - Root arbitrarily VERTICAL





```
class KDNode {
    2D_Point        point
    int             orient
    Region          region
    KDNode          above
    KDNode          below
}
```

```
class KDTree {
    KDNode          root
}
```

```
class Region {
    int  xmin, ymin
    int  xmax, ymax
}
```

NearestNeighbor Algorithm

Locate region where **X** would have been inserted

- Good place to start
- Confirm by traversing from root to see whether another point was actually closer

Nearest Neighbor

Best	Average	Worst
O(log n)	O(log n)	O(n)

KD-tree

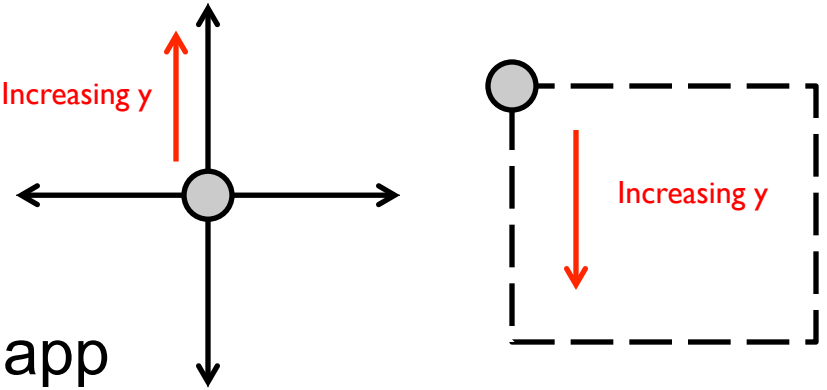
Recursion

```
def nearest (T, x)
  n = find parent node where x would have been inserted
  min = distance from x to n.point
  better = nearest (T.root, min, x)
  if (better found) { return better } else { return n }

def nearest (node, min, x)
  d = distance from x to node.point
  if (d < min) then
      min = d; result = node
  if “too close to call” then
     result = closer of nearest (node.above, min, x) and nearest (node.below, min, x)
  else
     if (node is above x) then
        pt = nearest (node.above, min, x)
     else
        pt = nearest (node.below, min, x)
     if (pt exists) then return pt
return result
```

Applications with KD-Trees

- Understand graphical coordinate systems
  - Cartesian coordinates different from computer coordinates
  - Origin in different location
  - Concept of up and down is different
- Code must reflect this
  - Let's write nearest neighbor app



KD-Tree Summary

- Derives efficiency from recursion inspired by Binary Trees
  - Other applications abound
  - “What points are contained in query rectangle?”
- Binary Space Partitioning structures
  - Foundation for efficient 3D-graphics