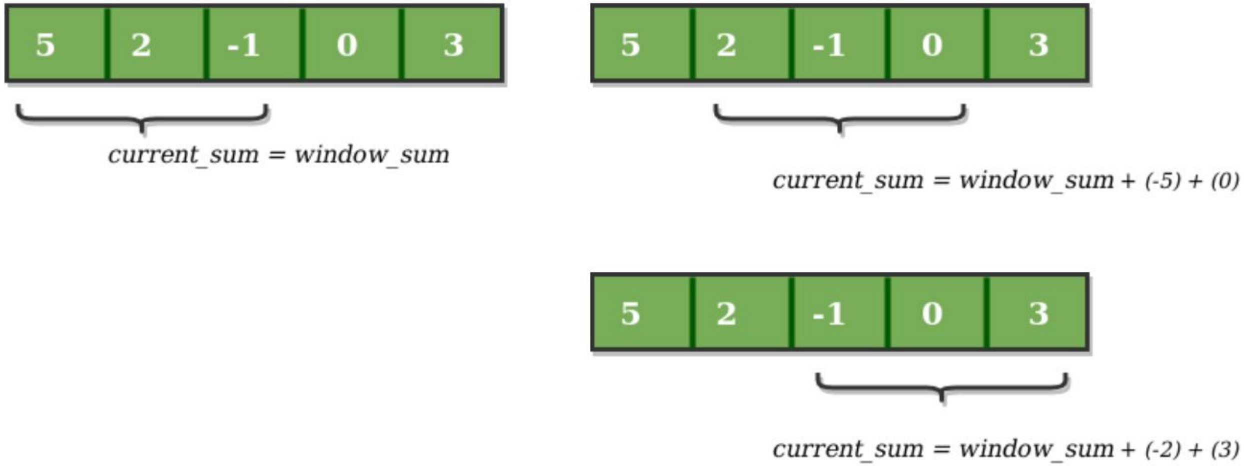
Exact Nearest Neighbor Algorithms

Sliding sums

- Suppose we want to "smooth" a histogram, i.e. replace the values by the average over a window
  - How can we do this efficiently?



CORNELL TECH

Sabermetrics

Derek Jeter

- One of the best players ever
  - .310 batting average
  - 3,465 hits
  - 260 home runs
  - 1,311 RBIs
  - 14x All-star
  - 5x World Series winner
- Who is the next Derek Jeter?

Source: Wikipedia

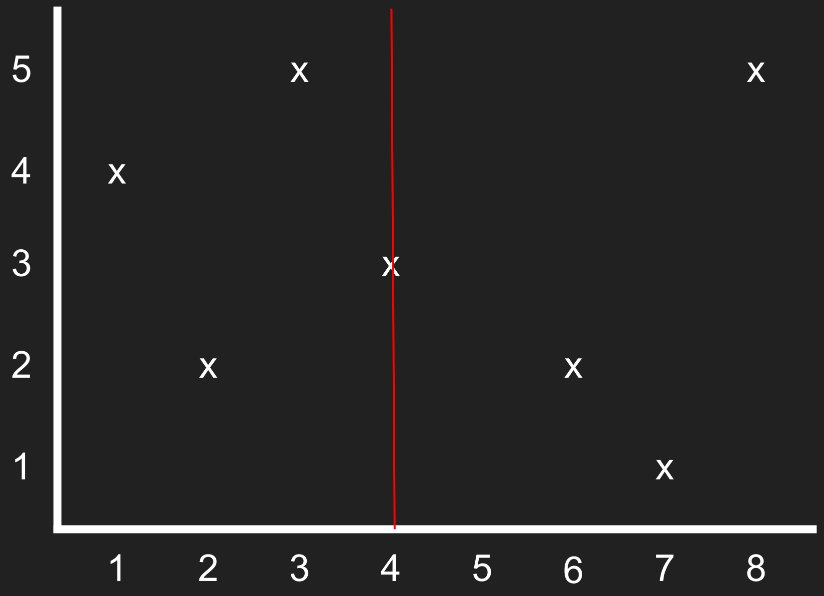
Sabermetrics

- Classic example of nearest neighbor application
- Hits, Home runs, RBIs, etc. are dimensions in “Baseball-space”
  - Every individual player has a unique point in this space
- Problem reduces to finding closest point in this space

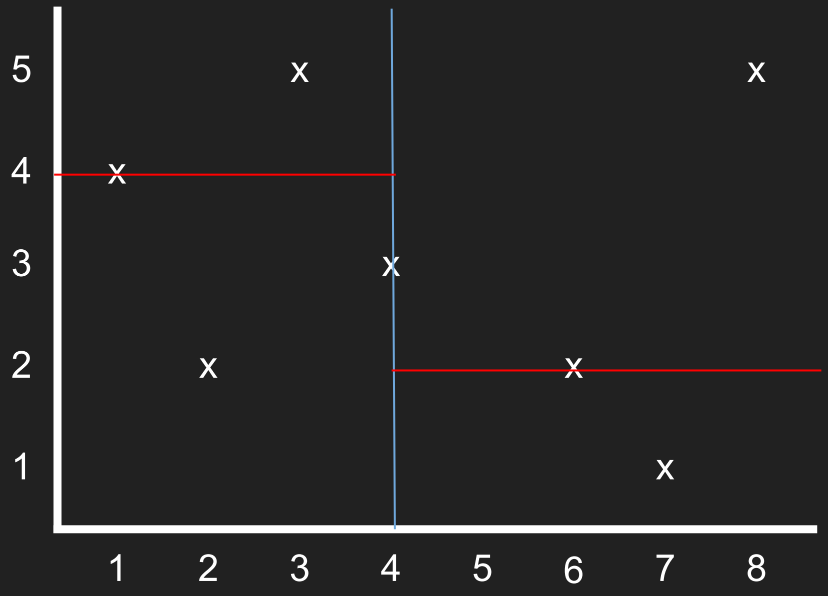
POI Suggestions

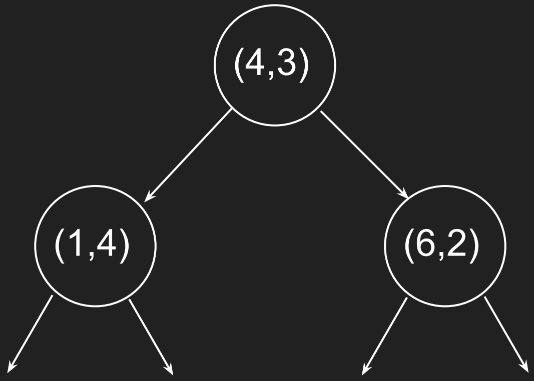
- Simpler example, only 2d space
- Known set of interest points in a map - we want to suggest the closest
  - Note: we could make this more complicated; we could add dimensions for ratings, category, newness, etc.
- How do we figure out what to suggest?
  - Brute force: just compute distance to all known points and pick lowest
    - *O(n)* in the number of points
    - Feels wasteful… why look at the distance to the Eiffel Tower when we know you're in NYC?
  - Space partitioning

2d trees

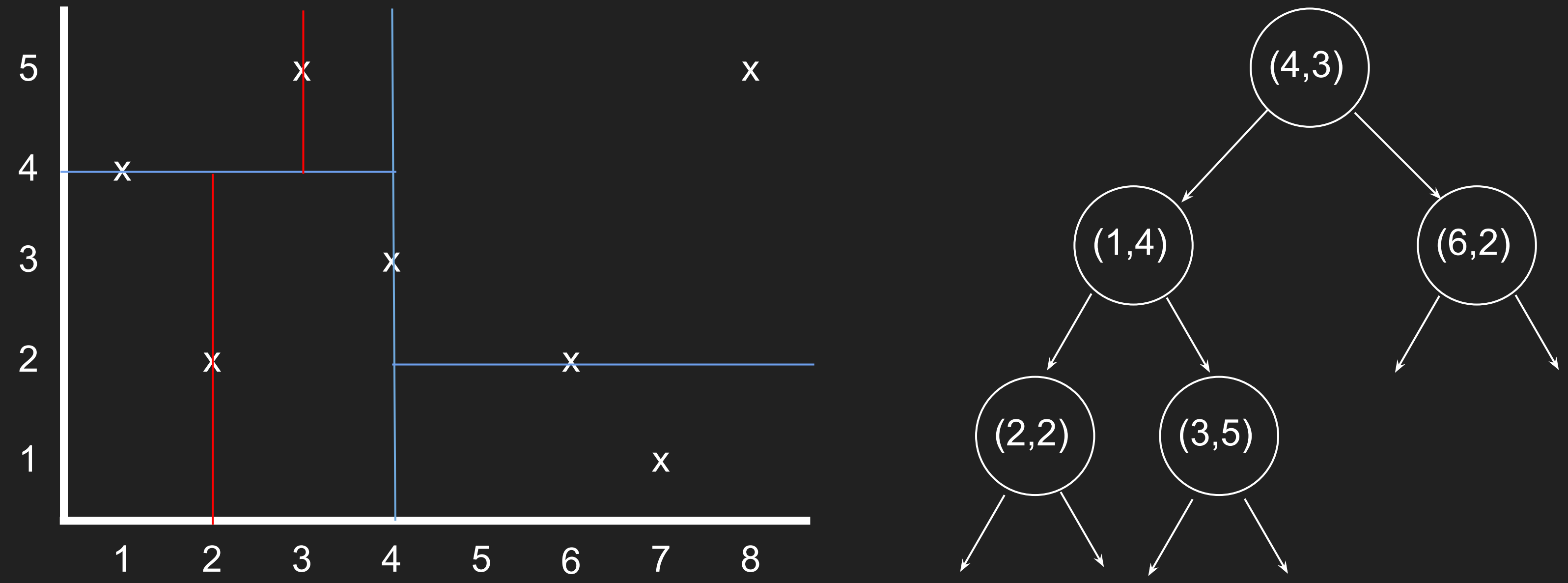


2d trees

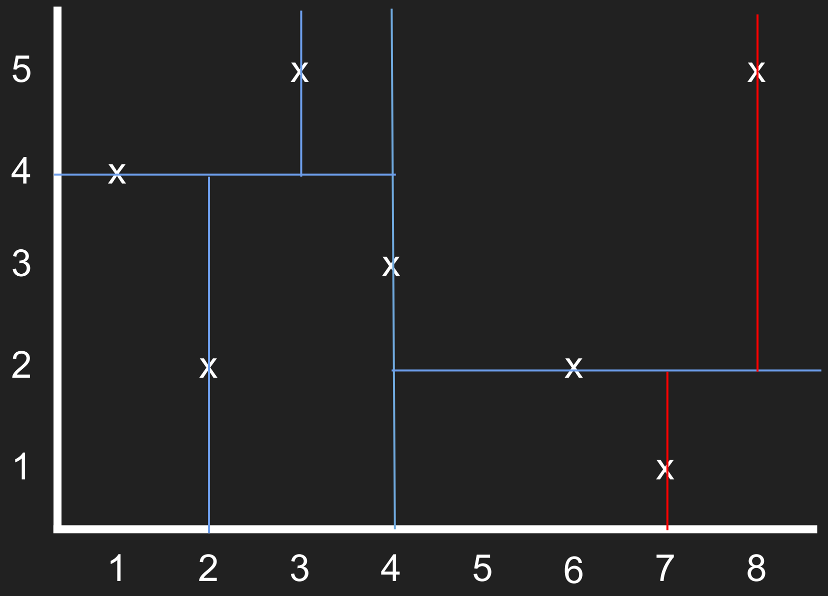


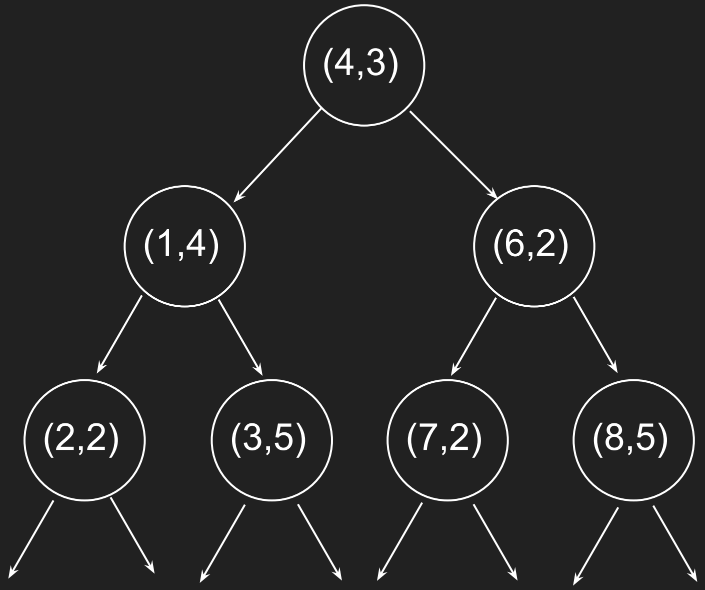


2d trees

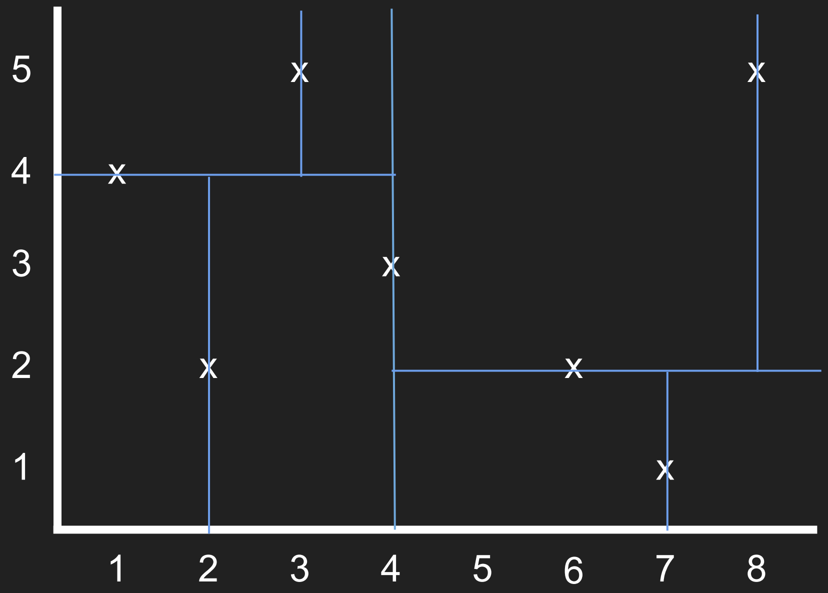


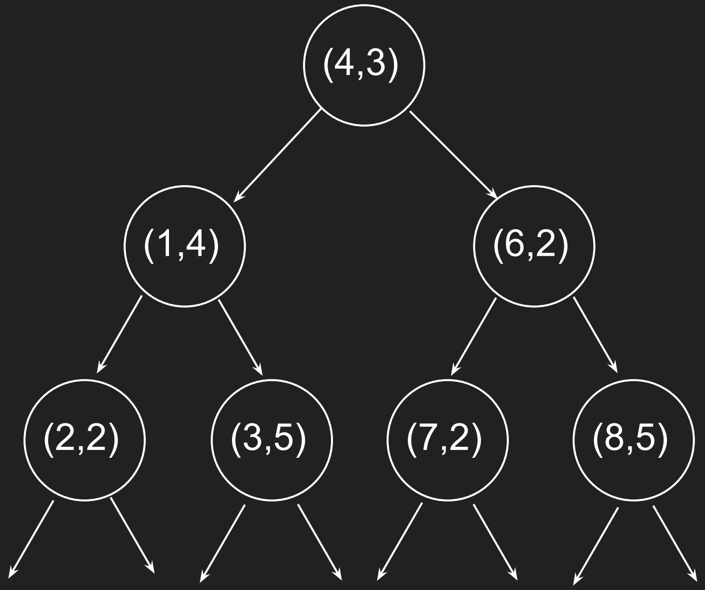
2d trees



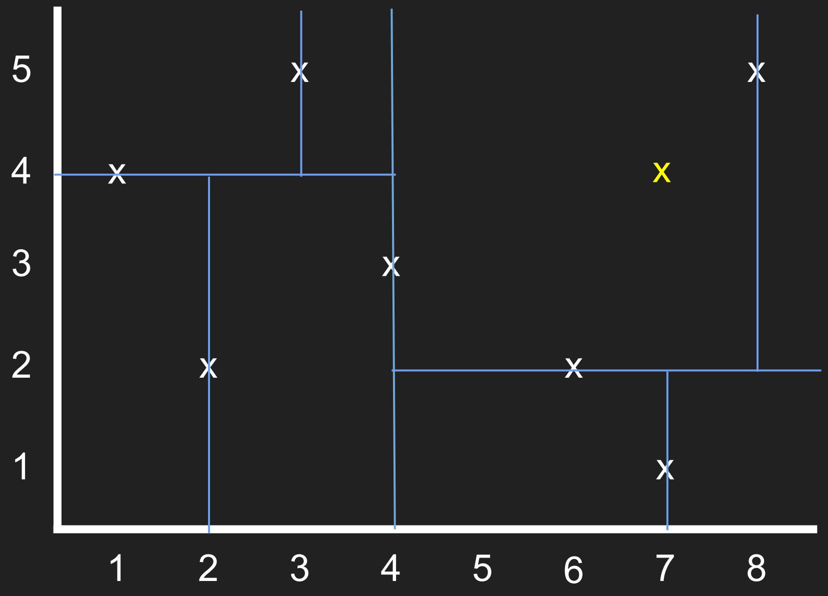


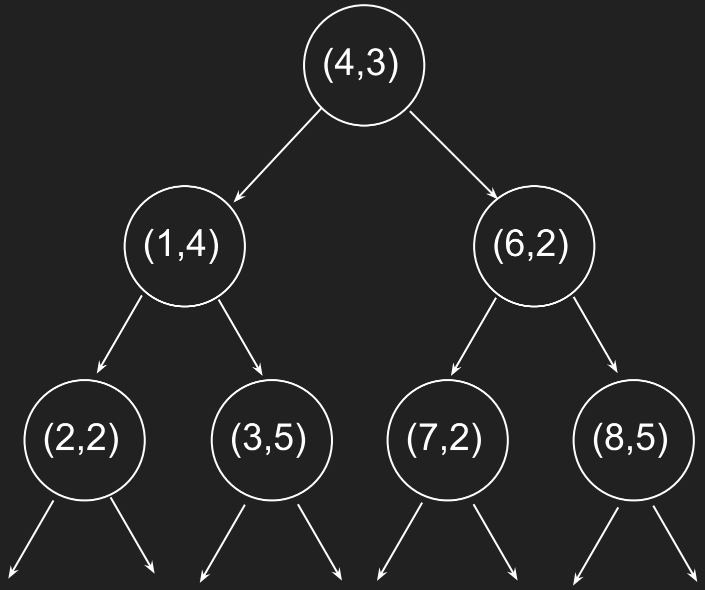
2d trees



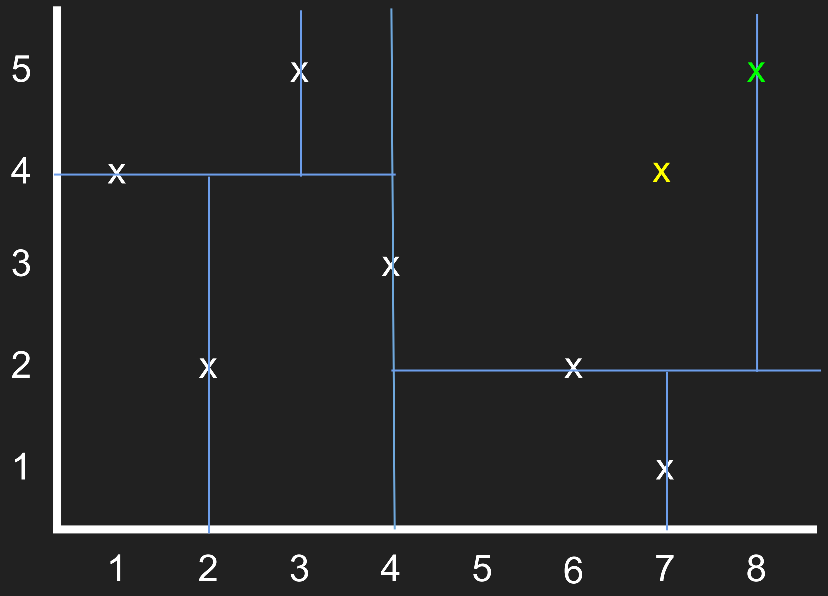


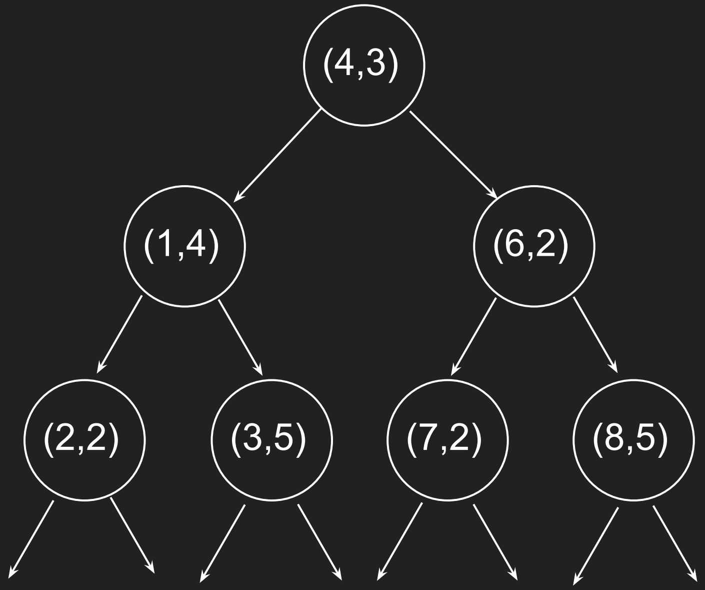
2d trees - Nearest Neighbor



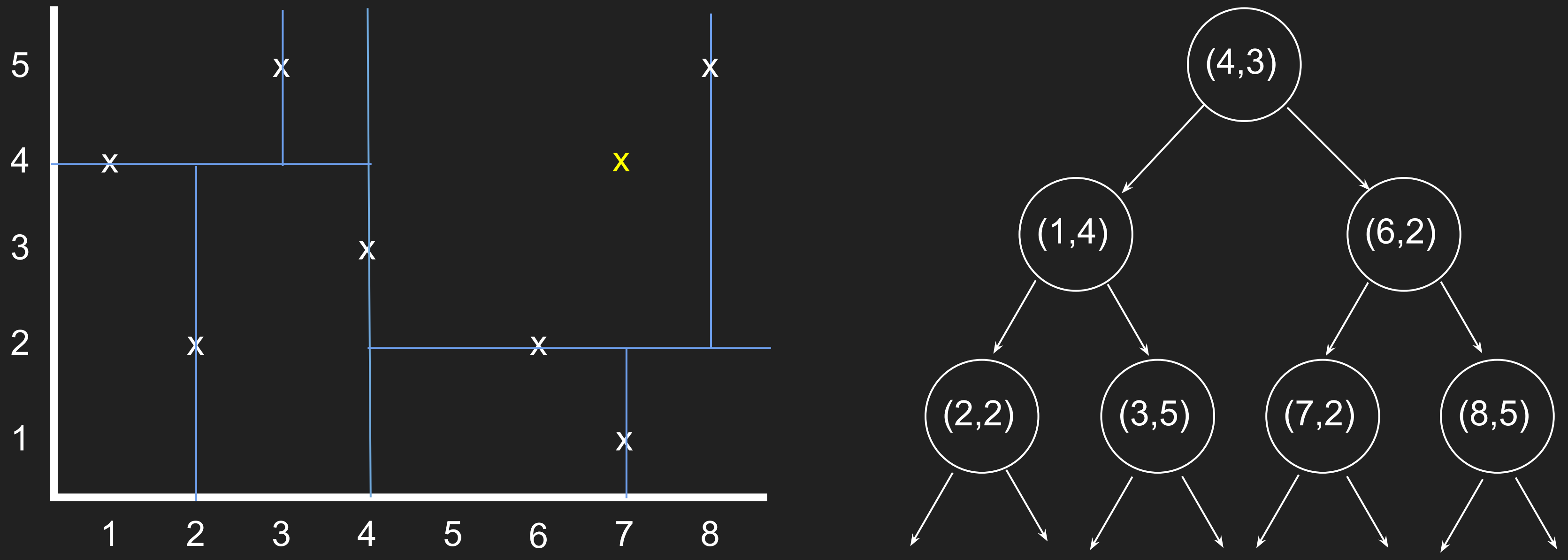


2d trees - Nearest Neighbor

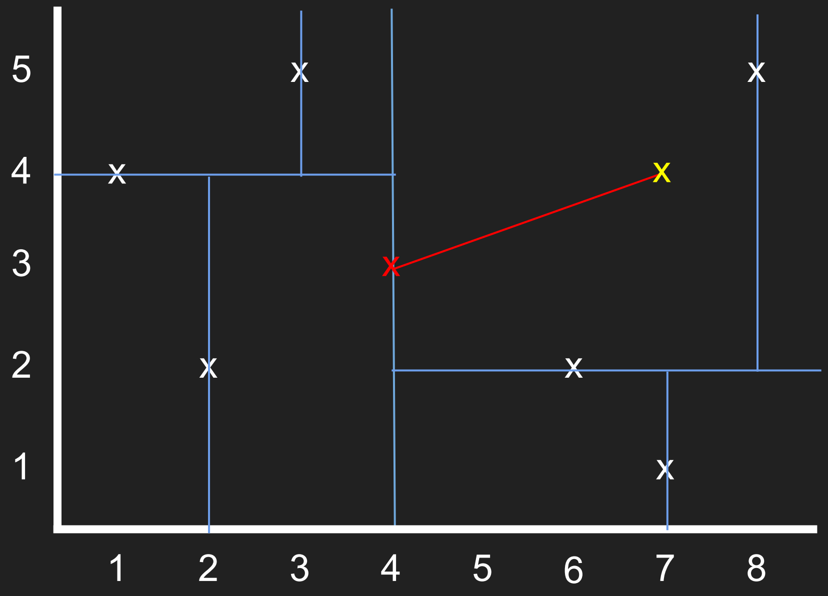


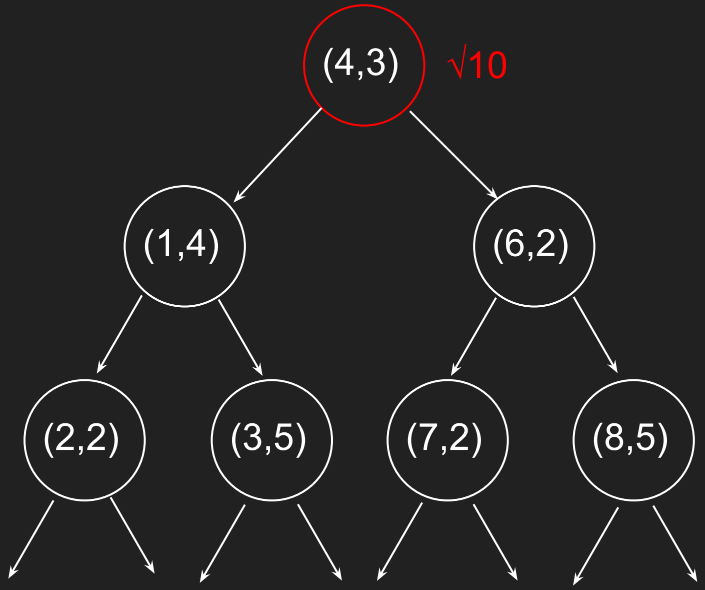


2d trees - Nearest Neighbor

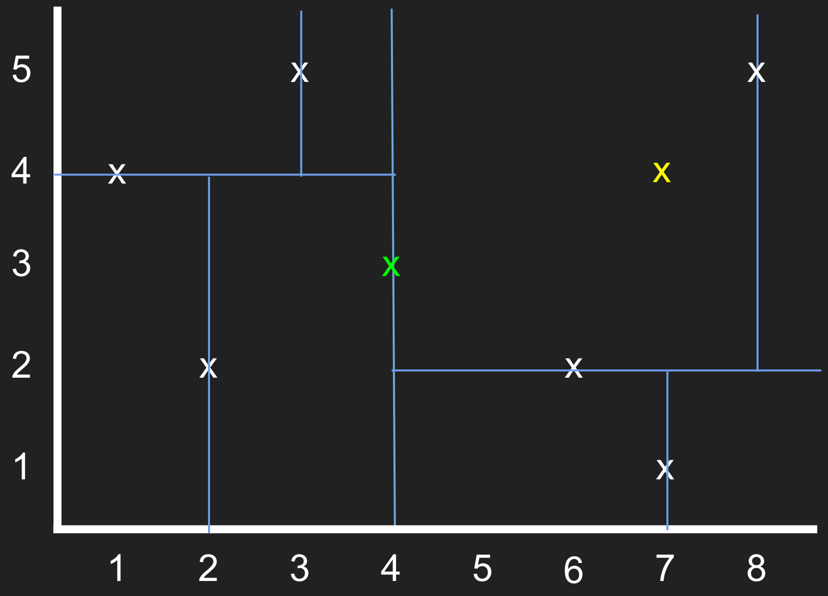


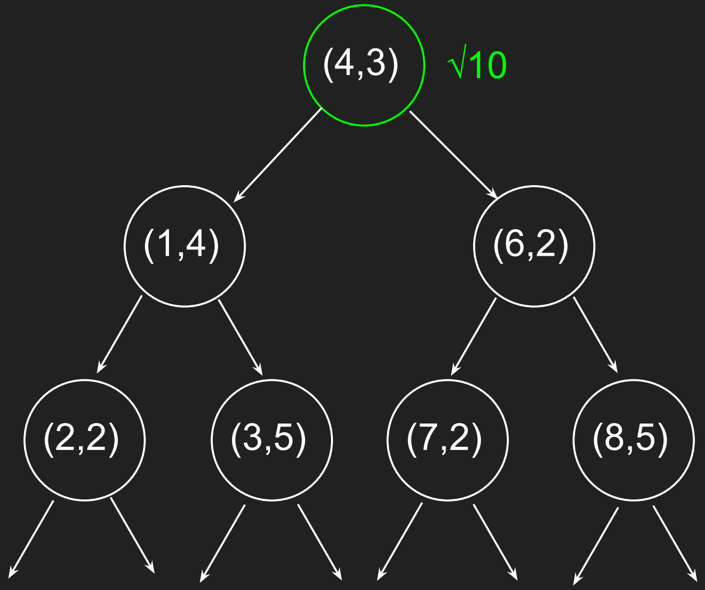
2d trees - Nearest Neighbor



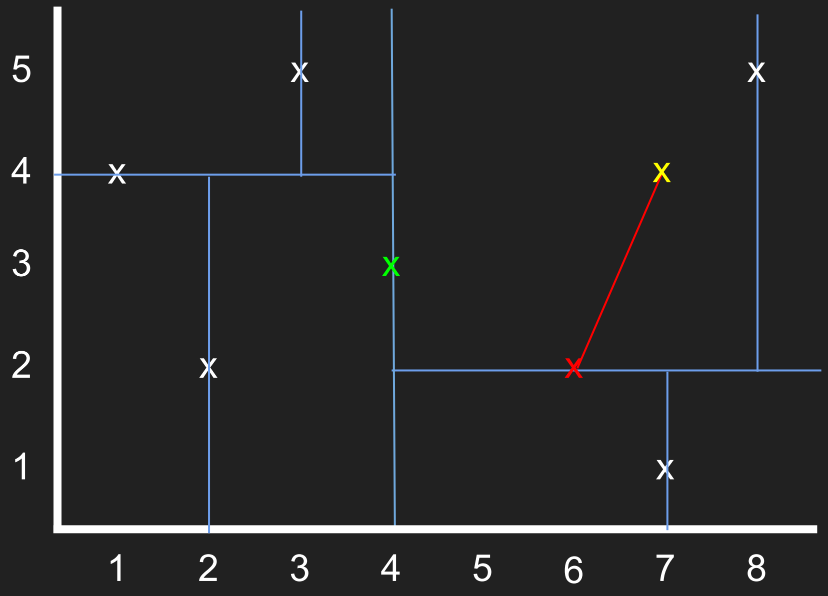


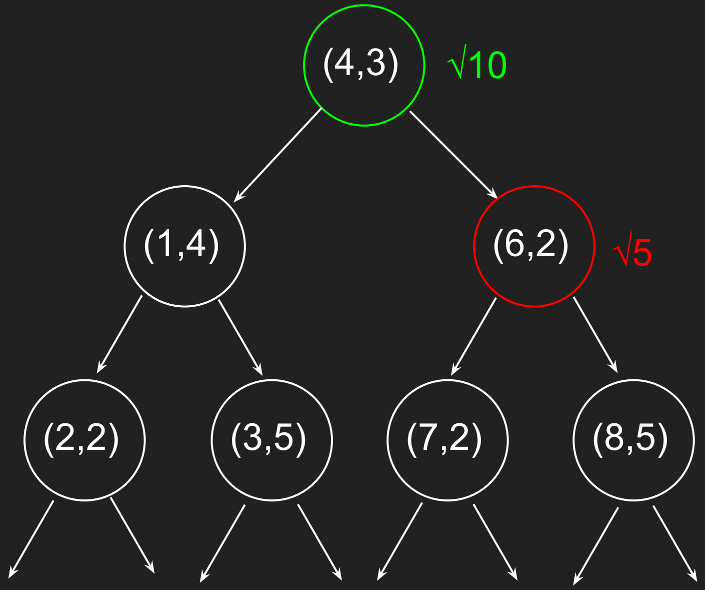
2d trees - Nearest Neighbor



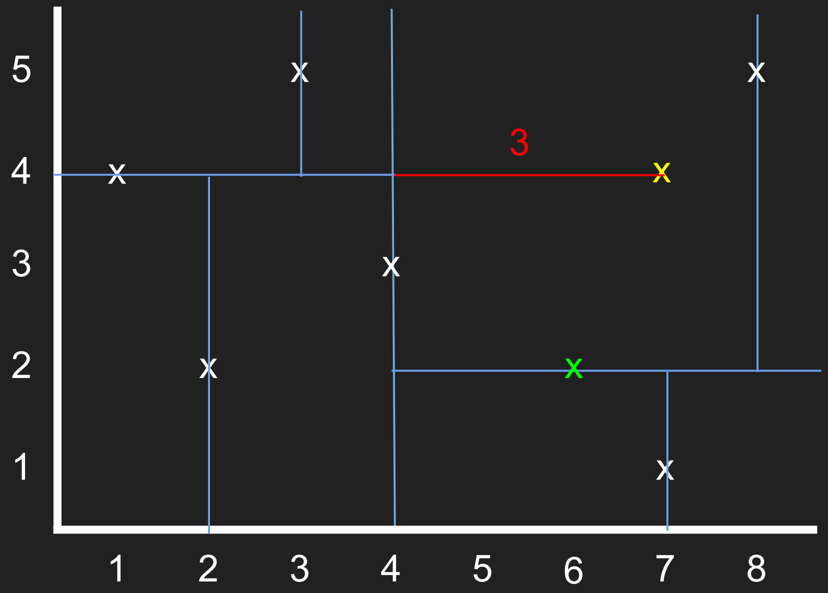


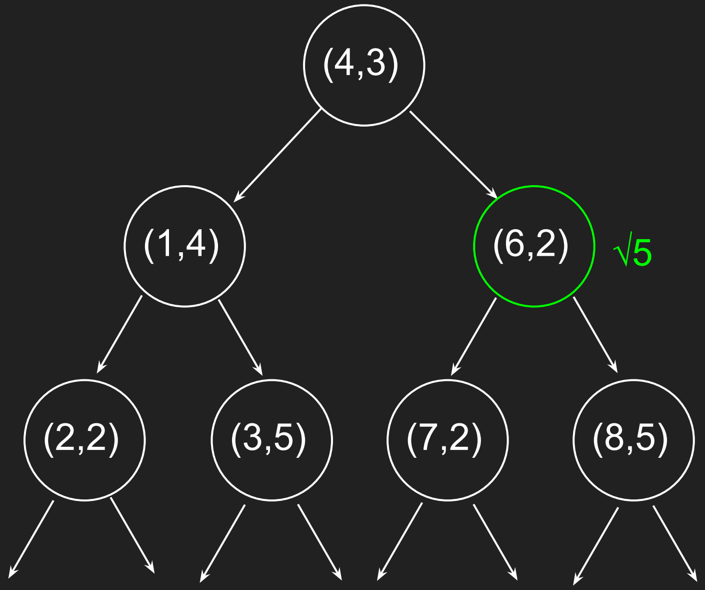
2d trees - Nearest Neighbor



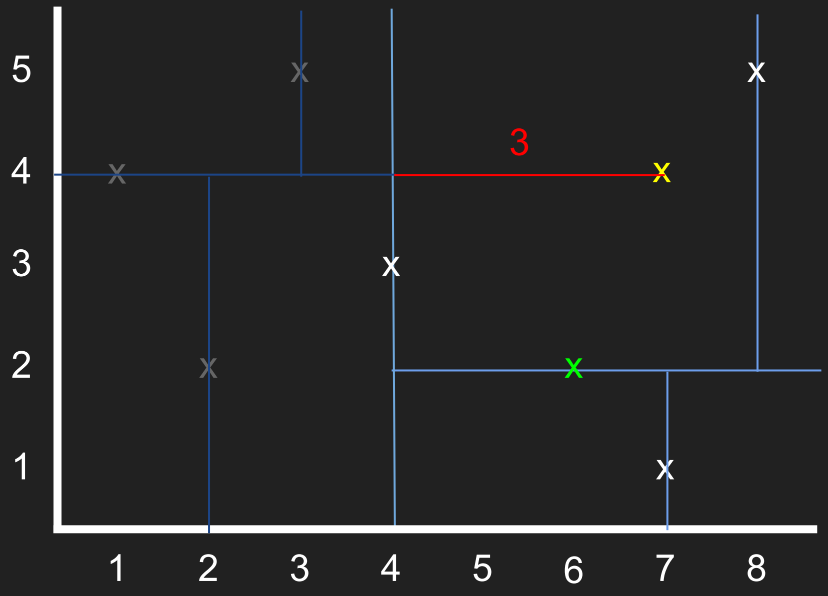


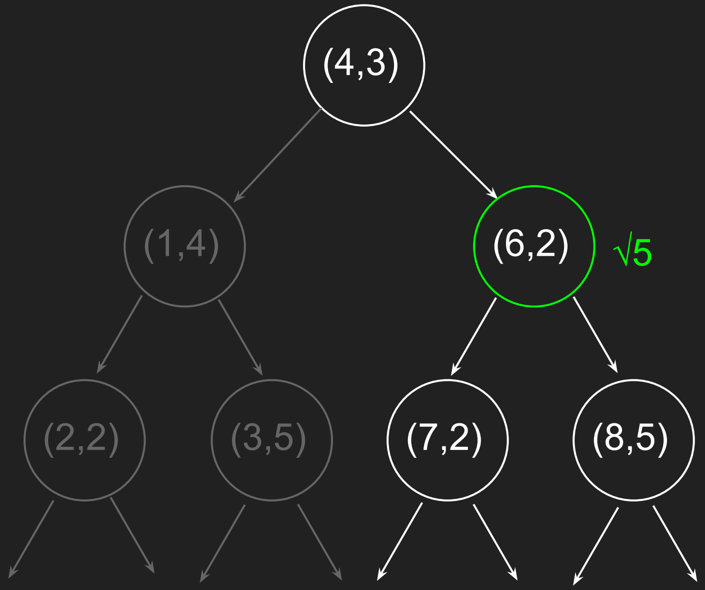
2d trees - Nearest Neighbor



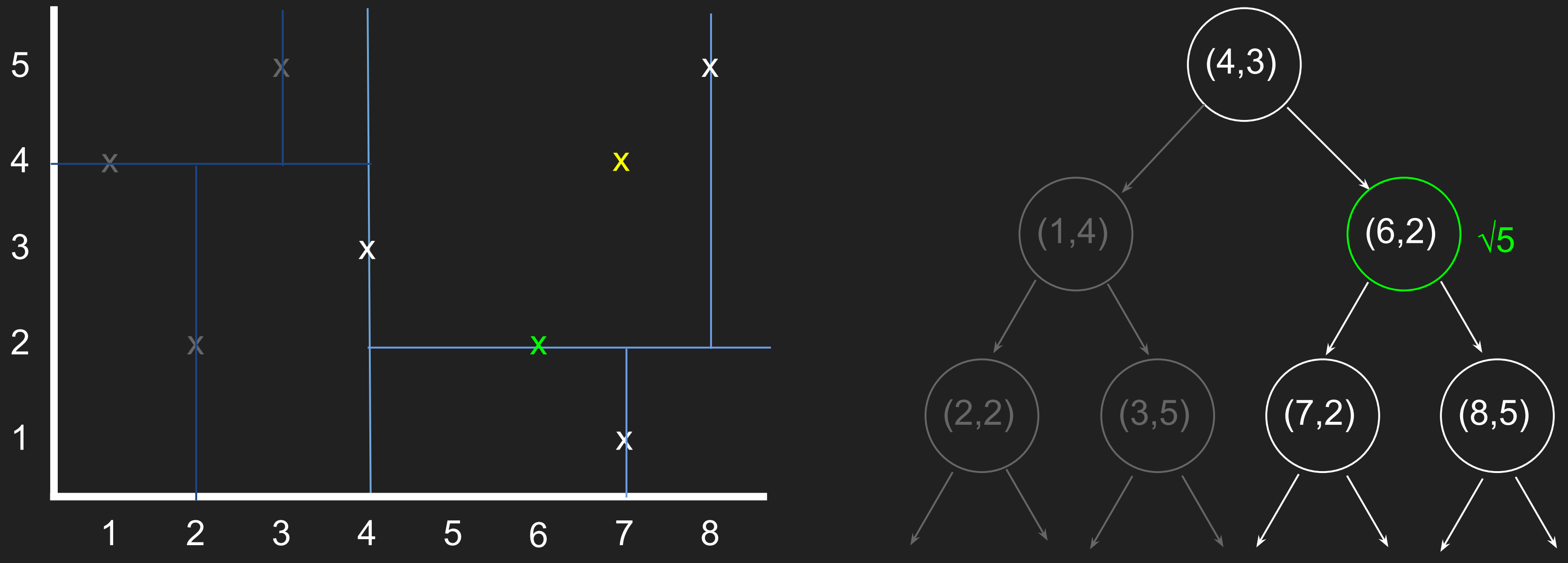


2d trees - Nearest Neighbor

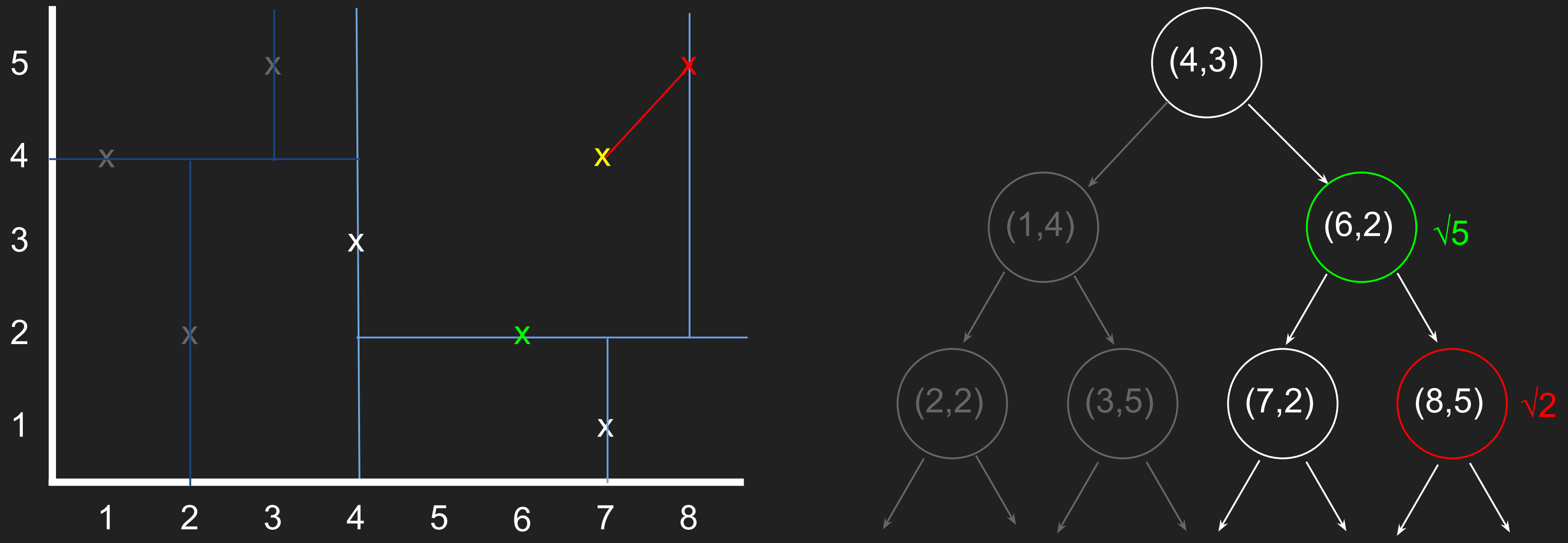




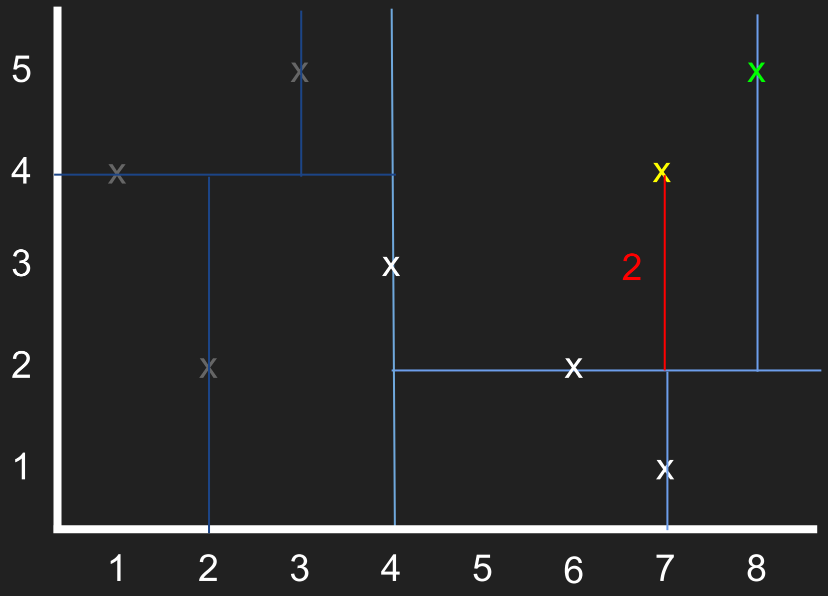
2d trees - Nearest Neighbor

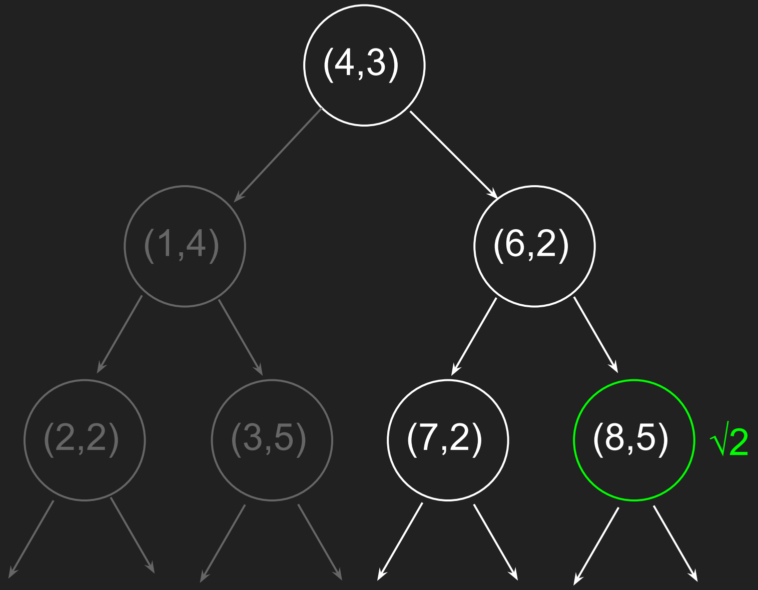


2d trees - Nearest Neighbor

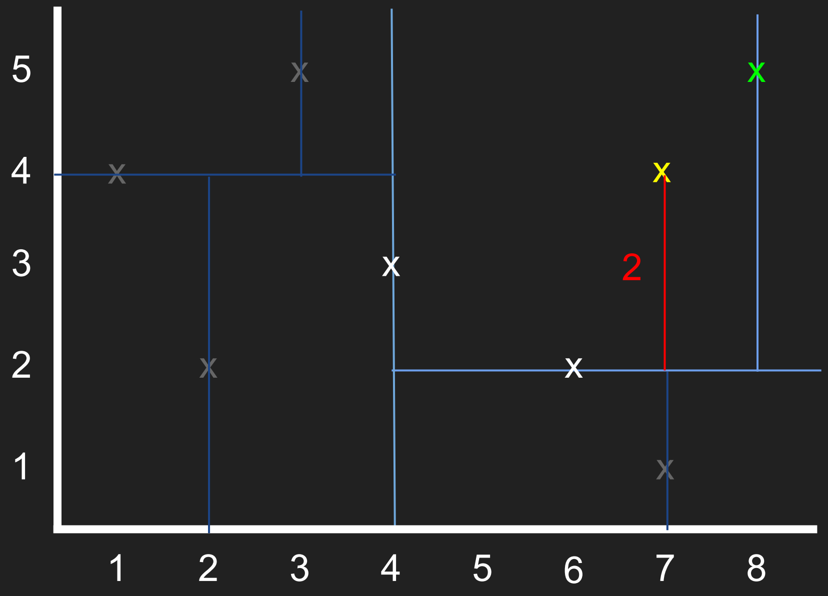


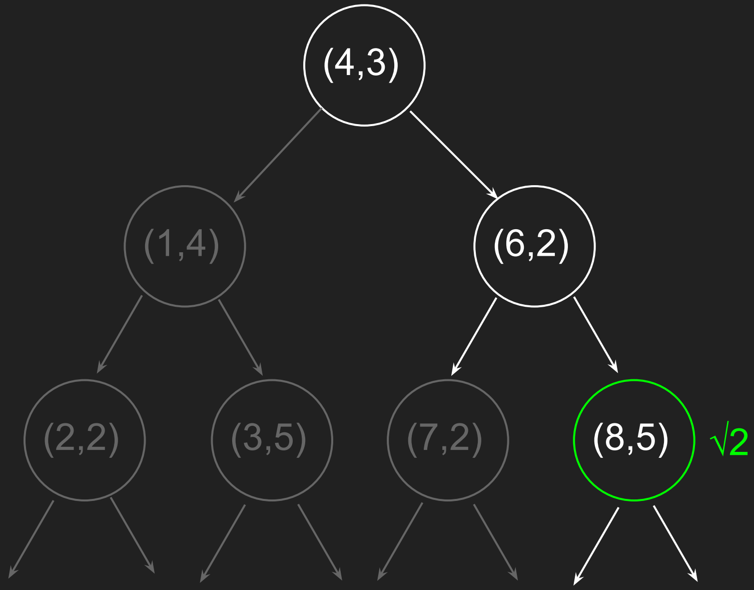
2d trees - Nearest Neighbor



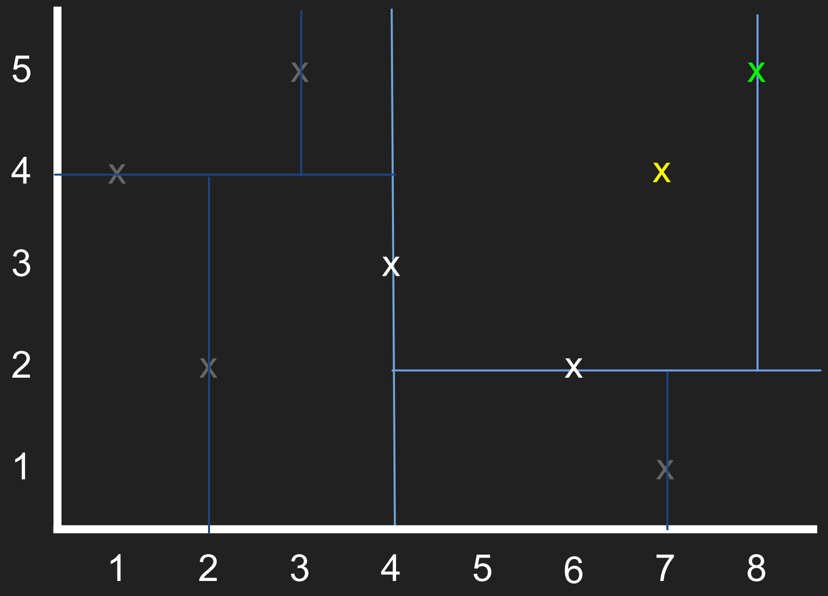


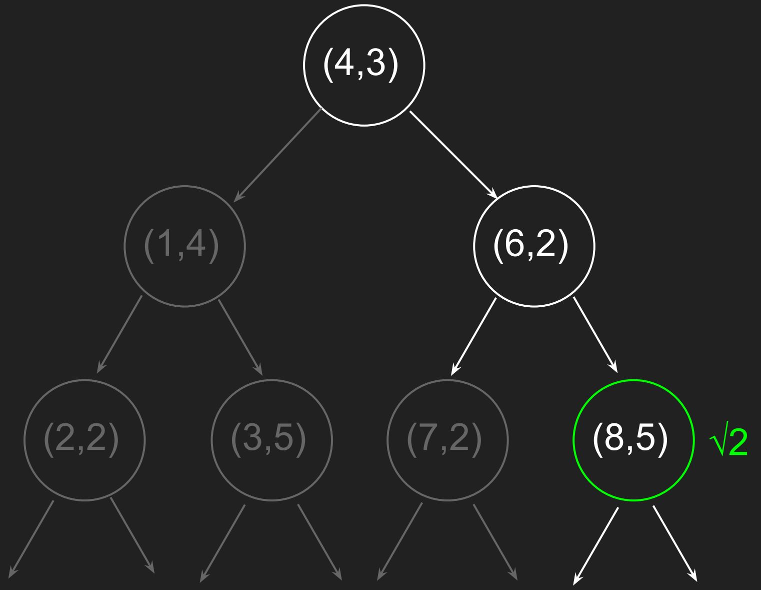
2d trees - Nearest Neighbor



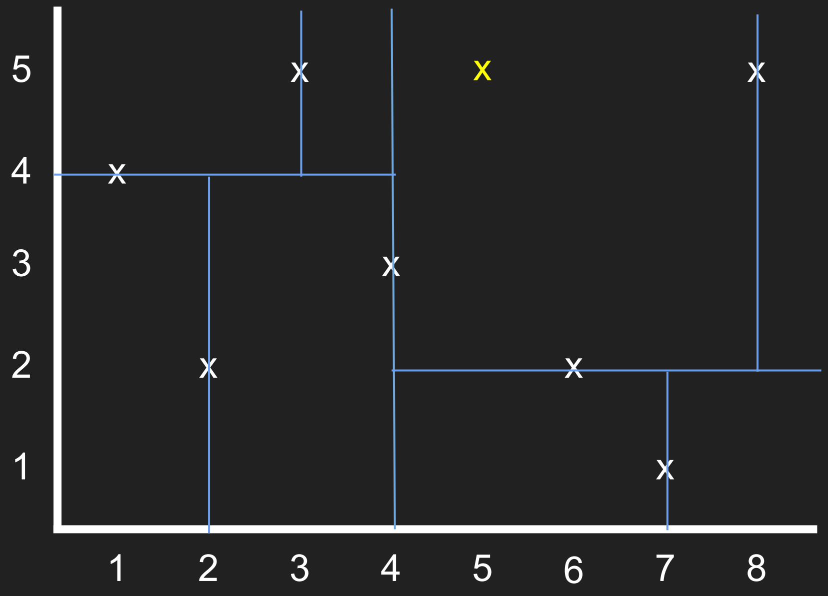


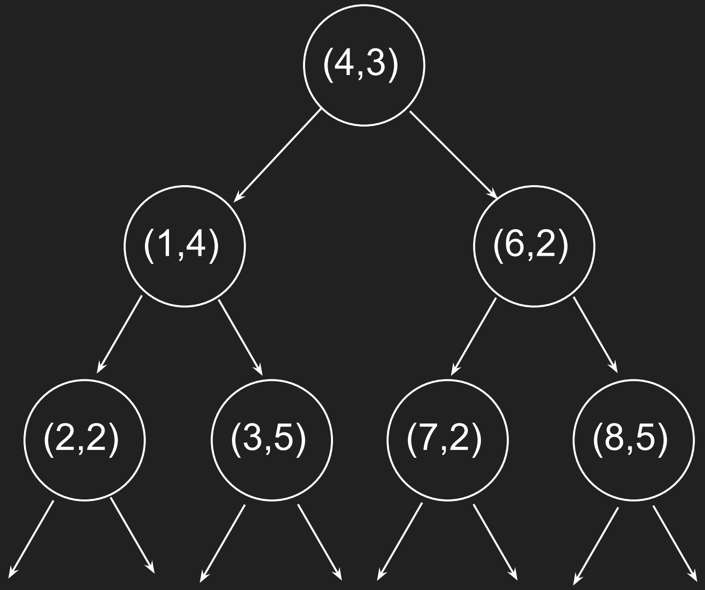
2d trees - Nearest Neighbor



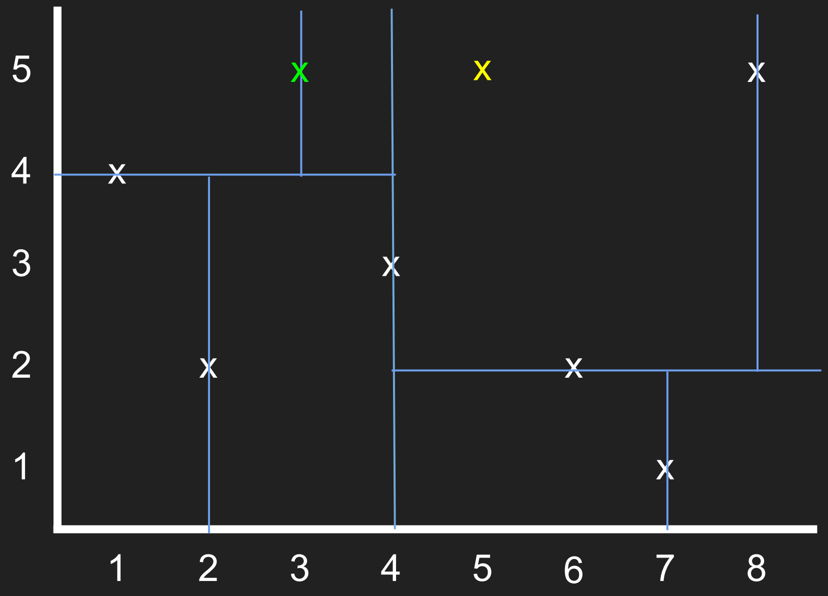


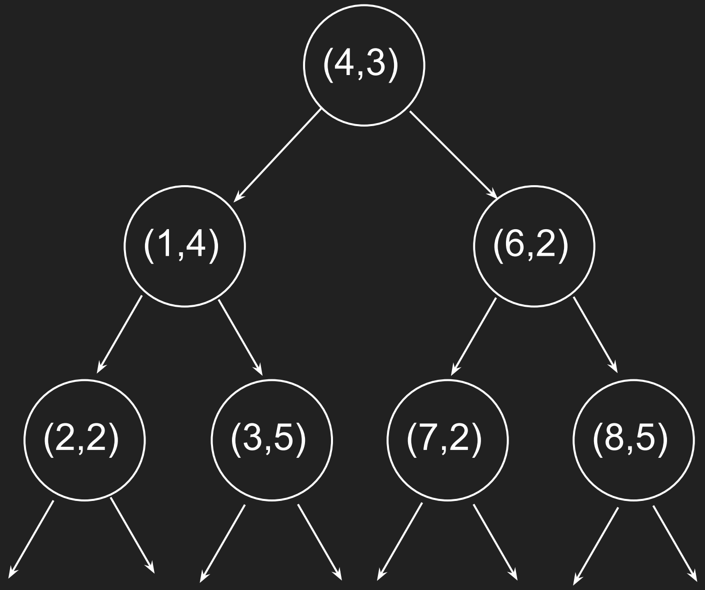
2d trees - Nearest Neighbor



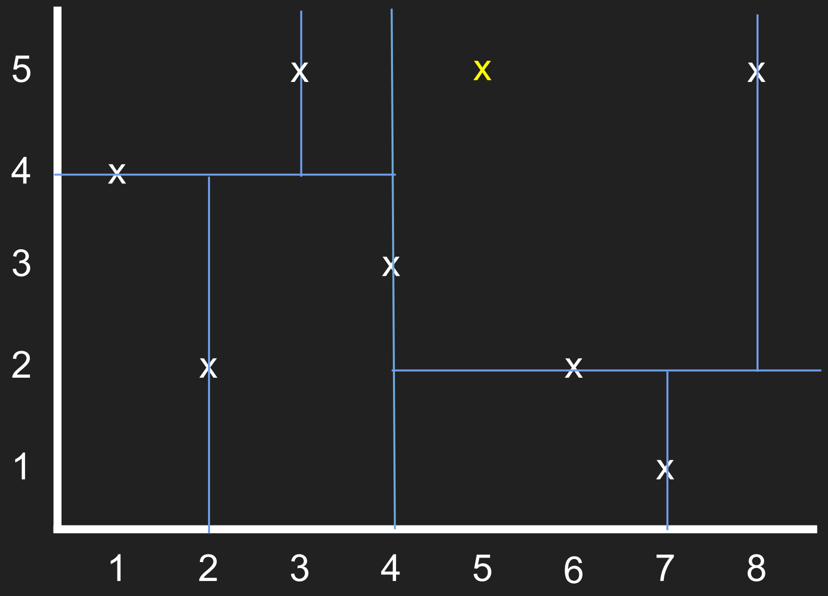


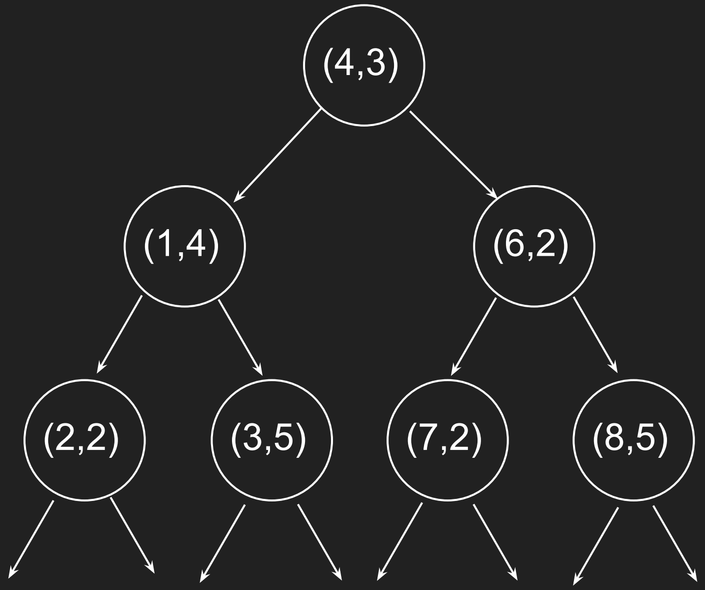
2d trees - Nearest Neighbor



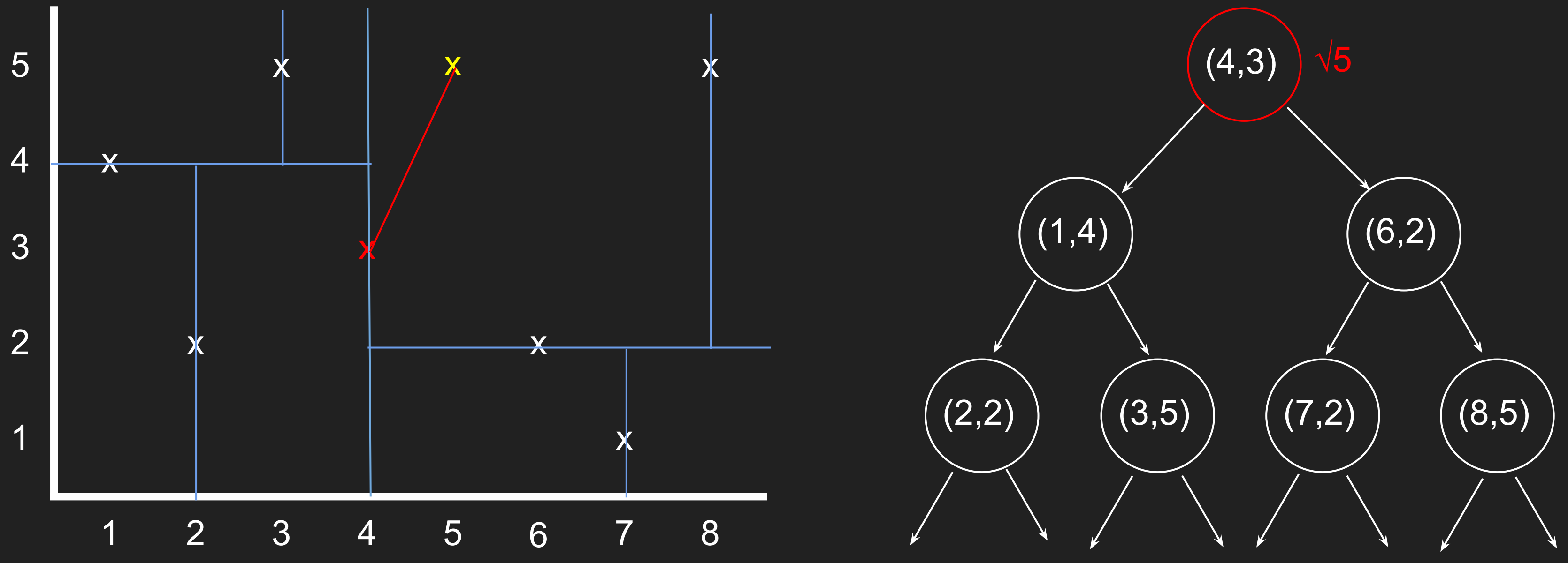


2d trees - Nearest Neighbor

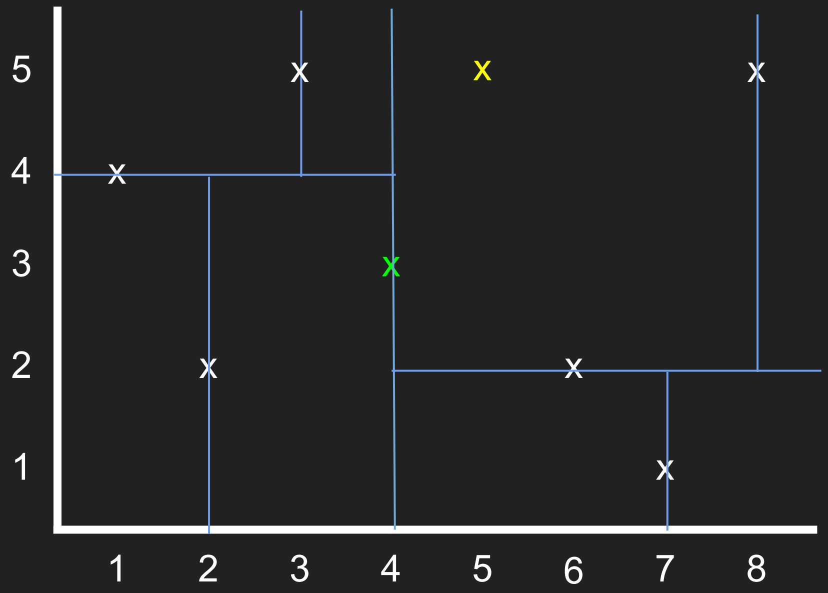


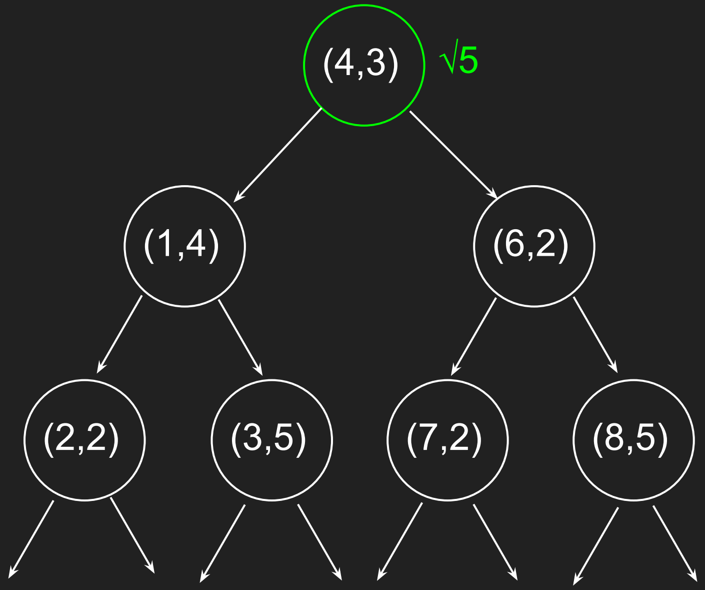


2d trees - Nearest Neighbor

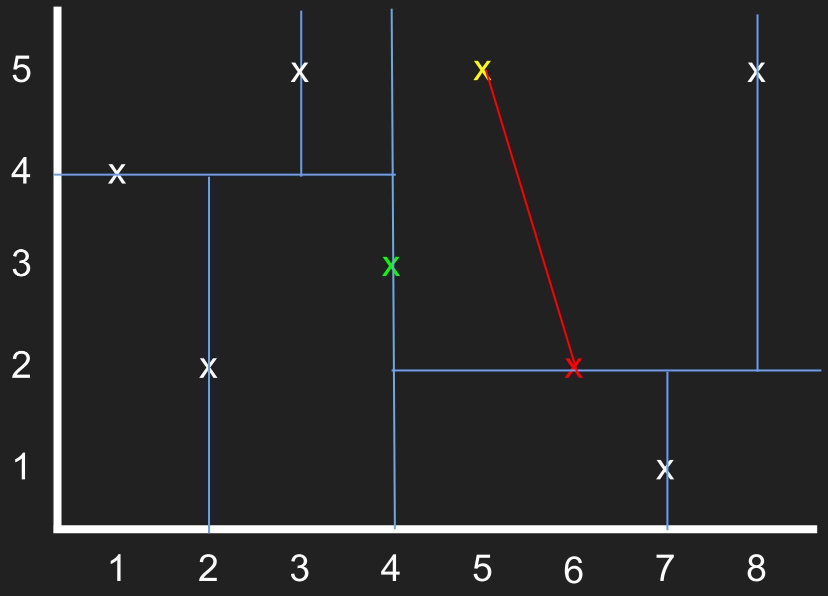


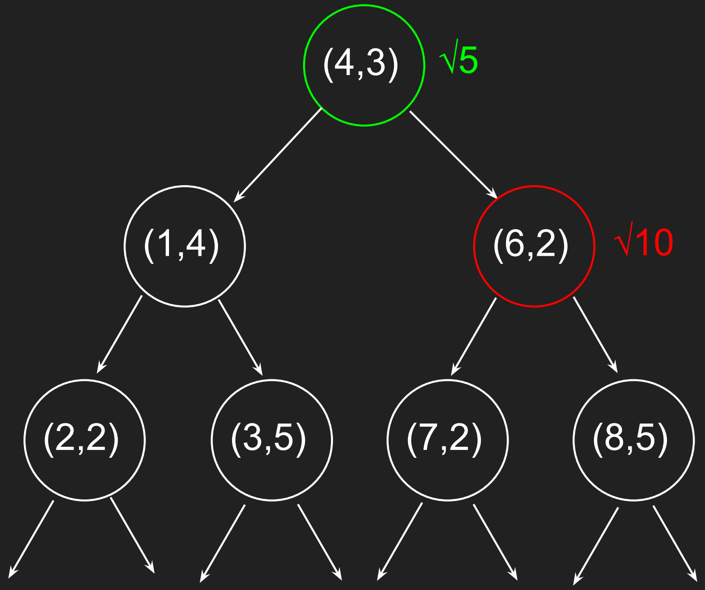
2d trees - Nearest Neighbor



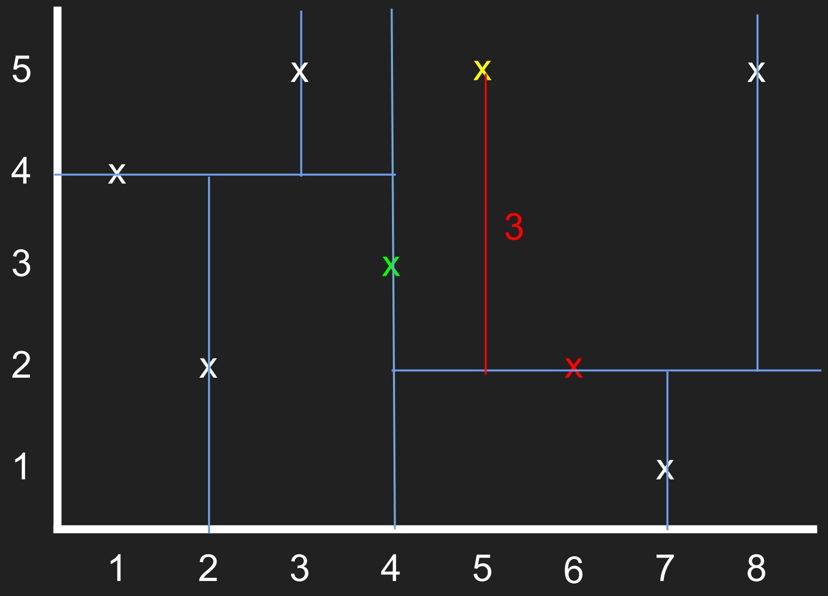


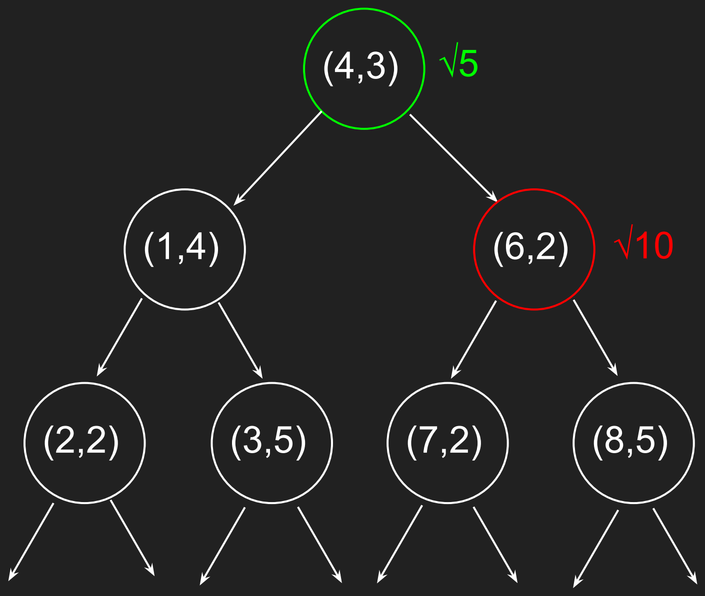
2d trees - Nearest Neighbor



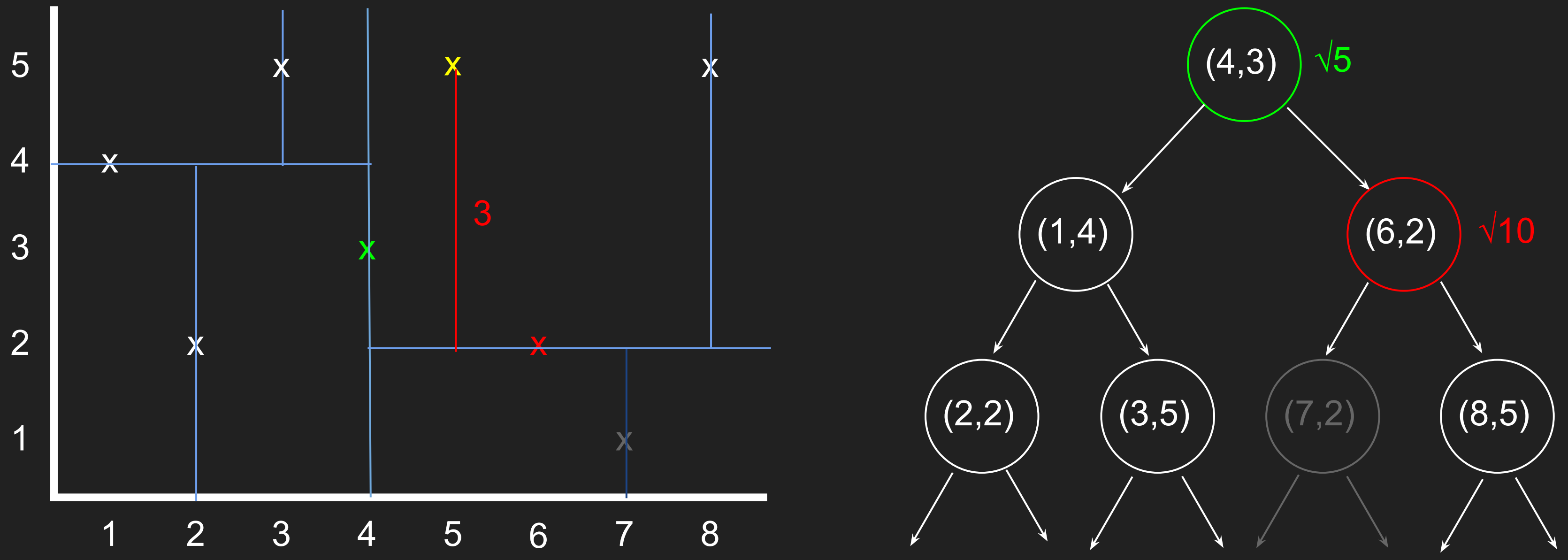


2d trees - Nearest Neighbor

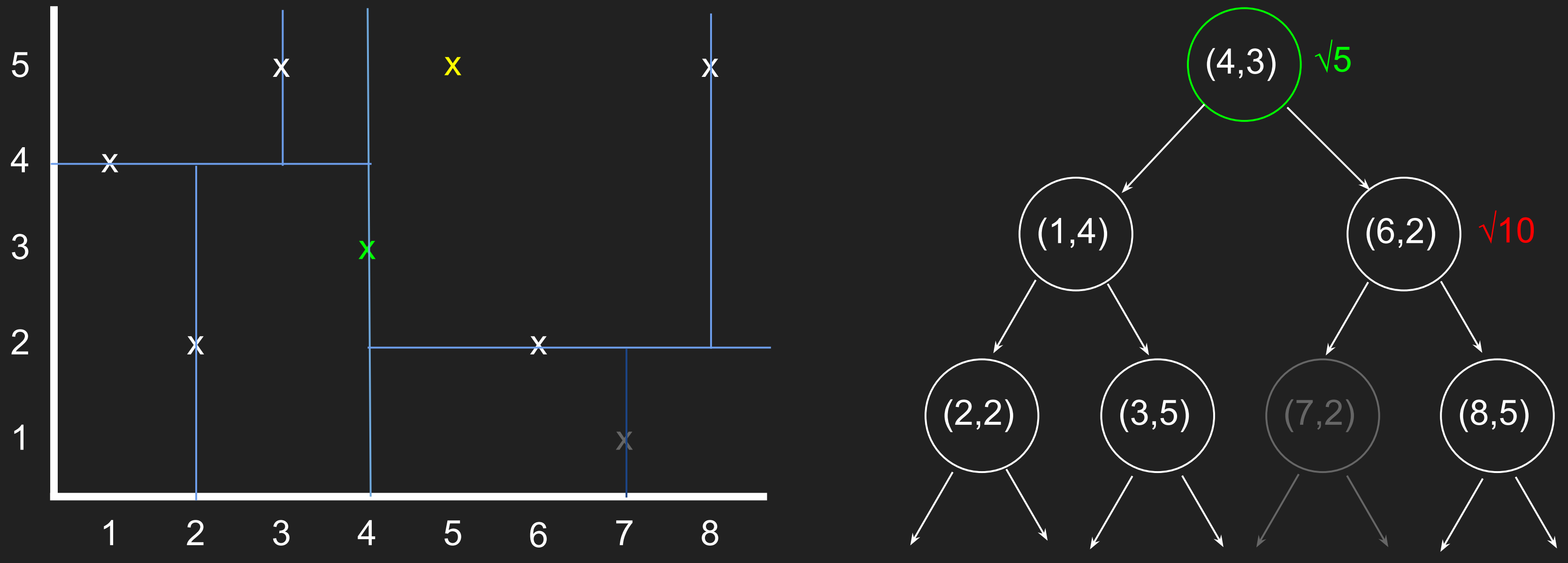




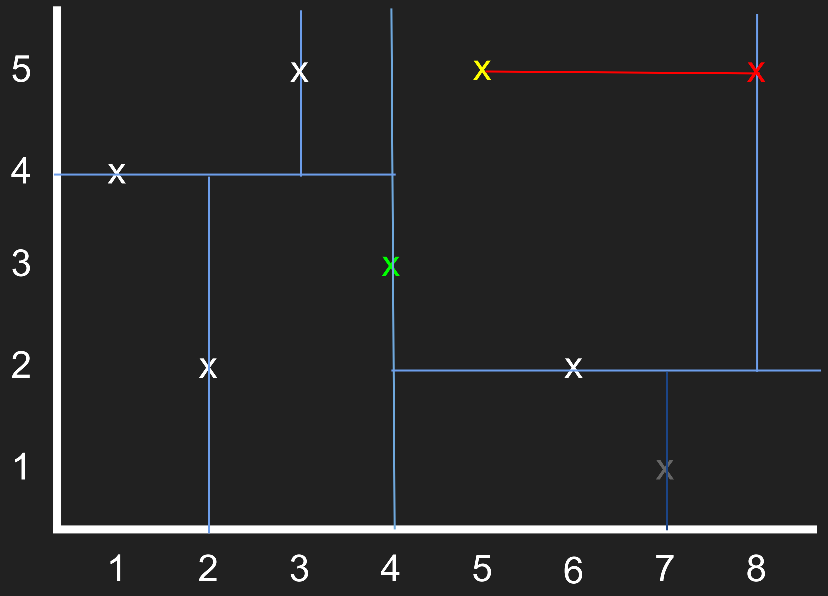
2d trees - Nearest Neighbor

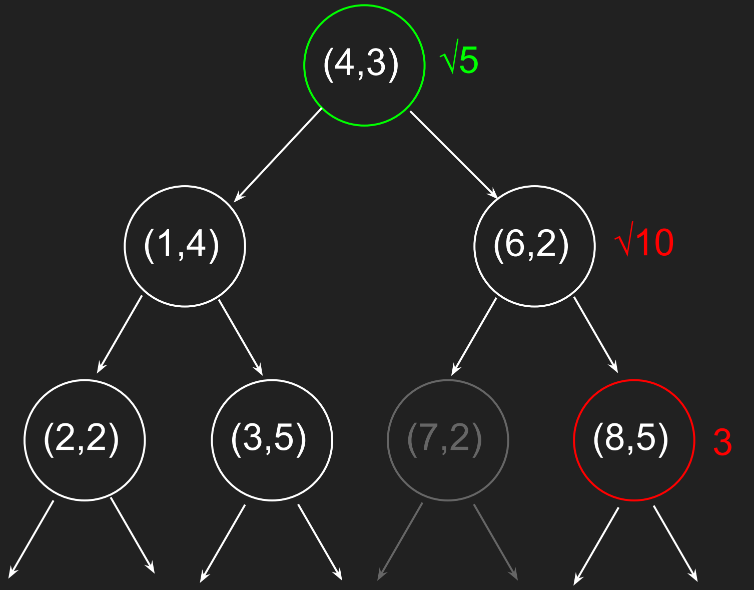


2d trees - Nearest Neighbor

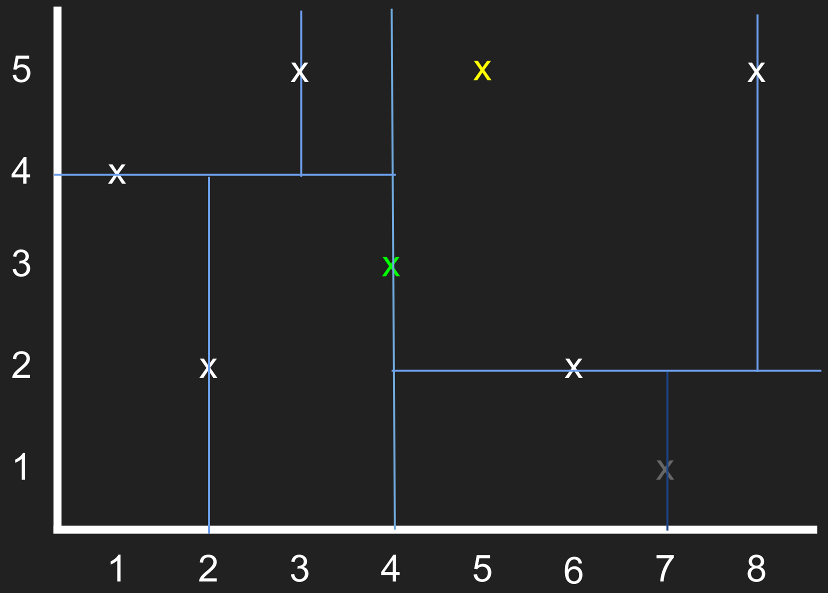


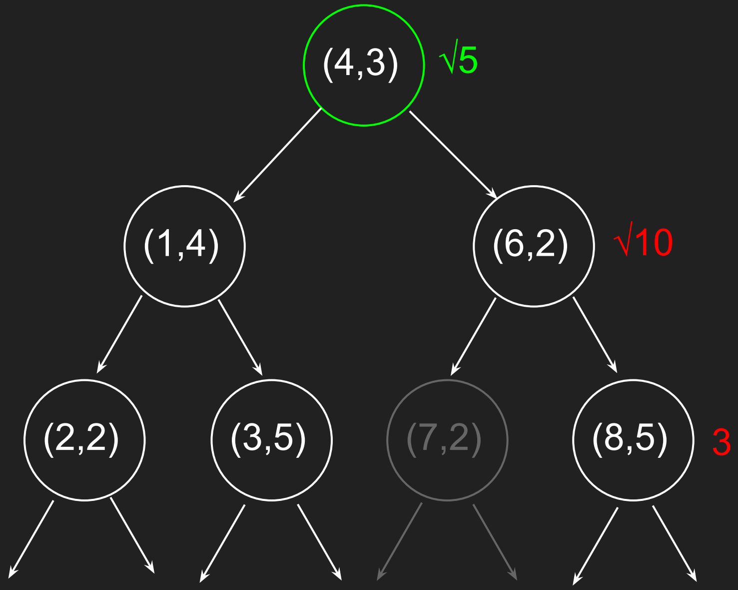
2d trees - Nearest Neighbor



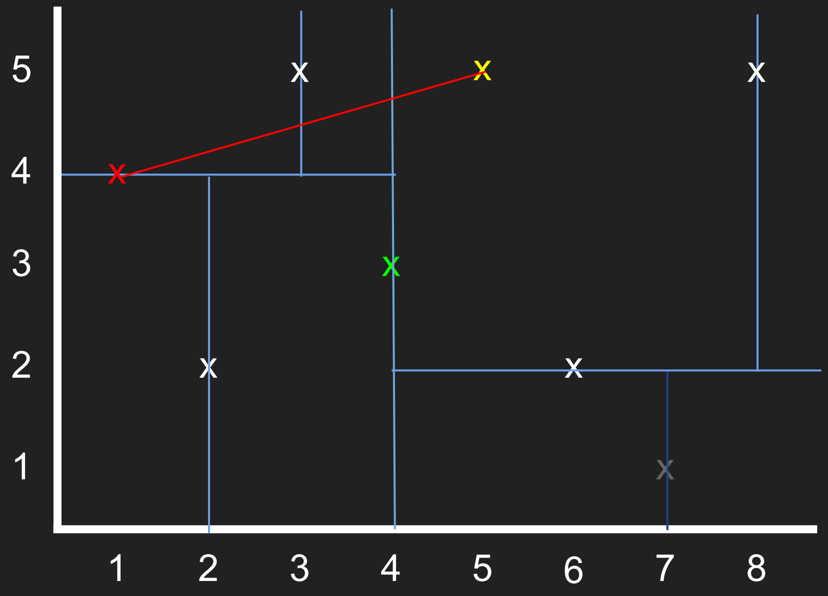


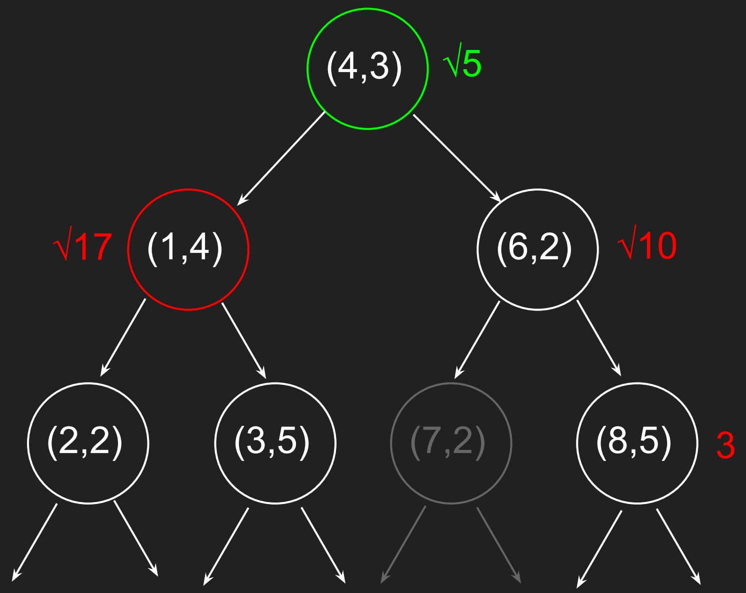
2d trees - Nearest Neighbor



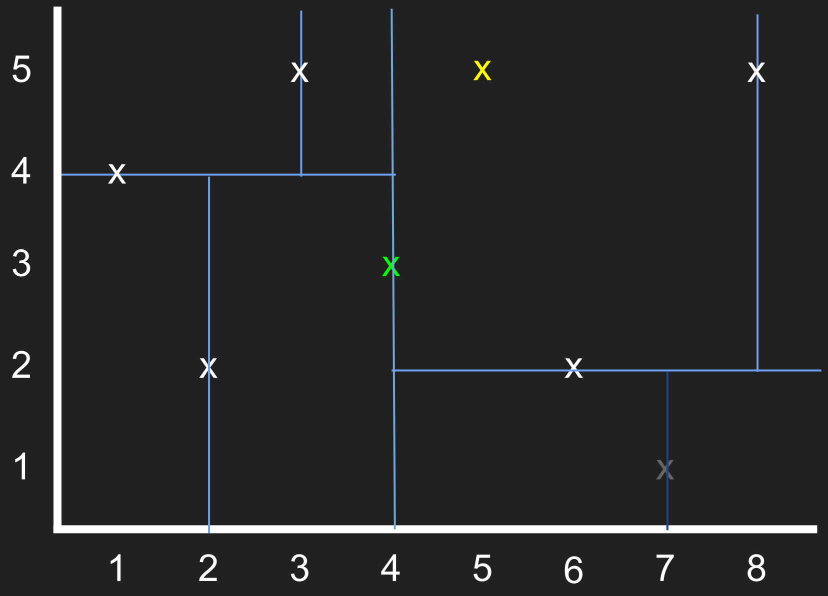


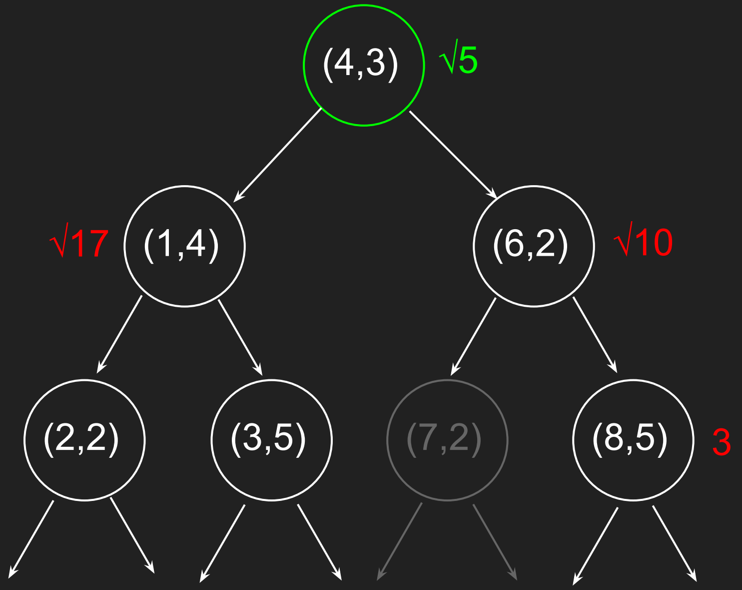
2d trees - Nearest Neighbor



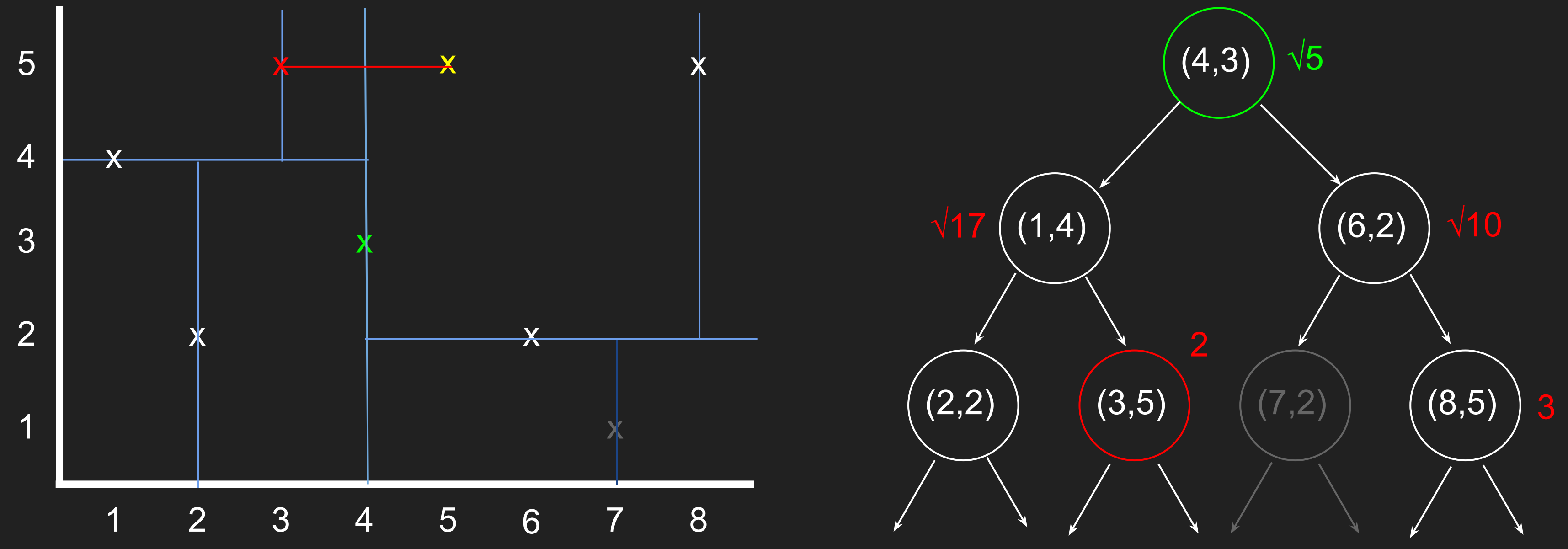


2d trees - Nearest Neighbor

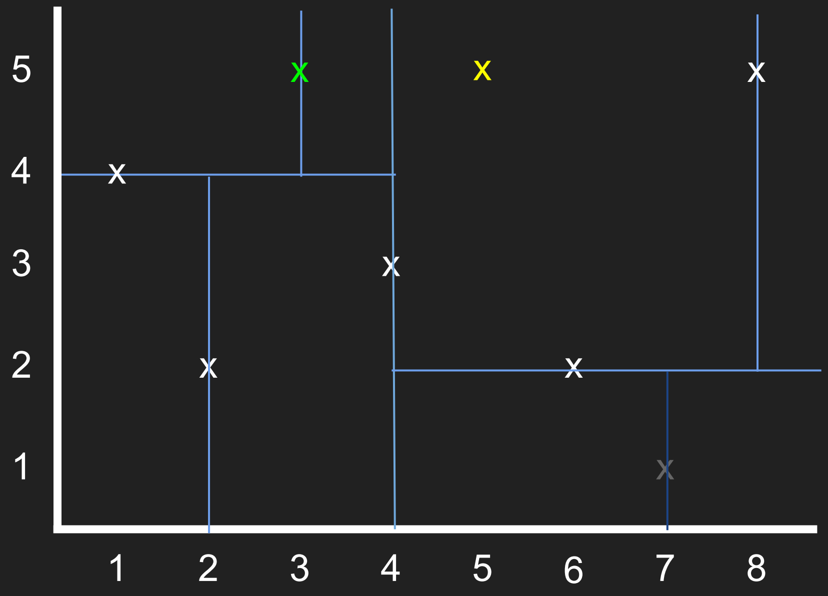


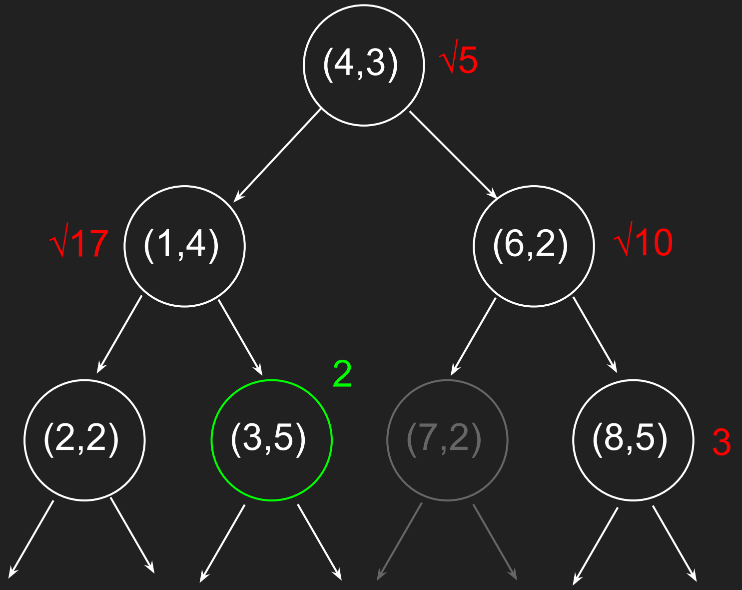


2d trees - Nearest Neighbor

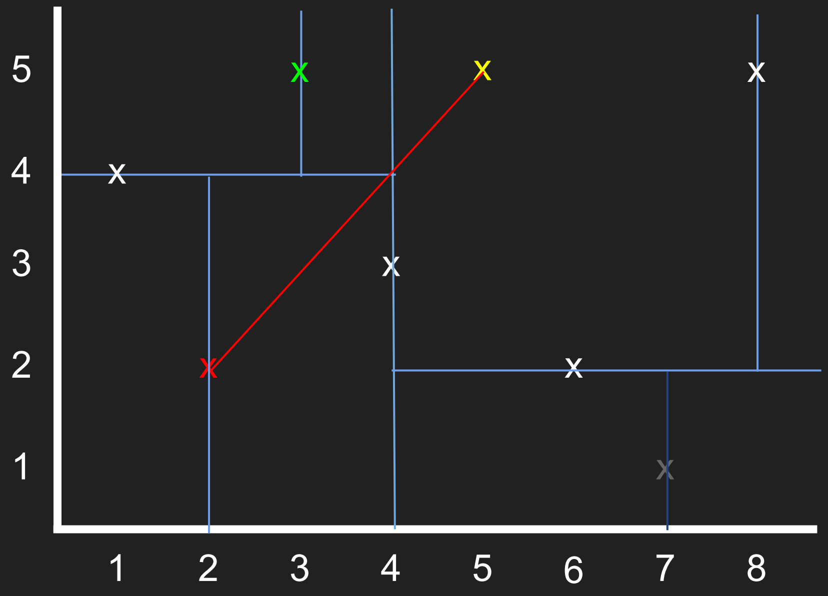


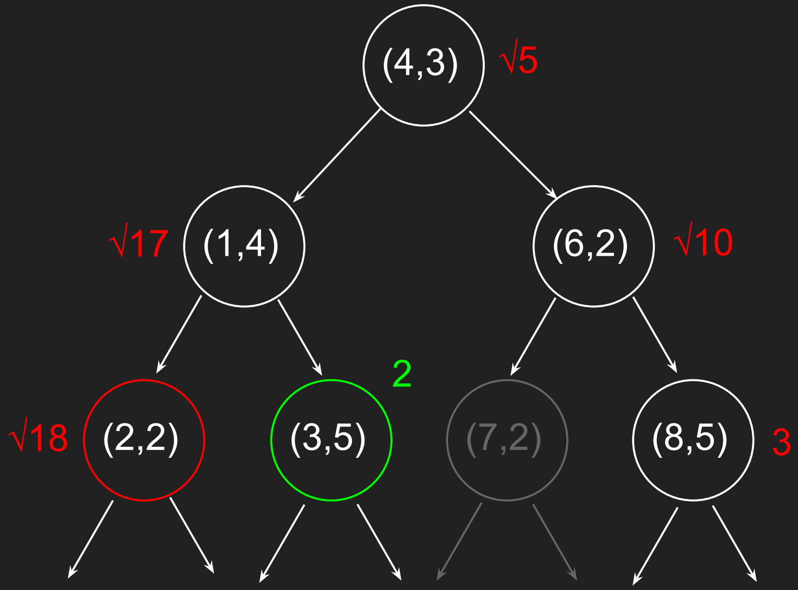
2d trees - Nearest Neighbor



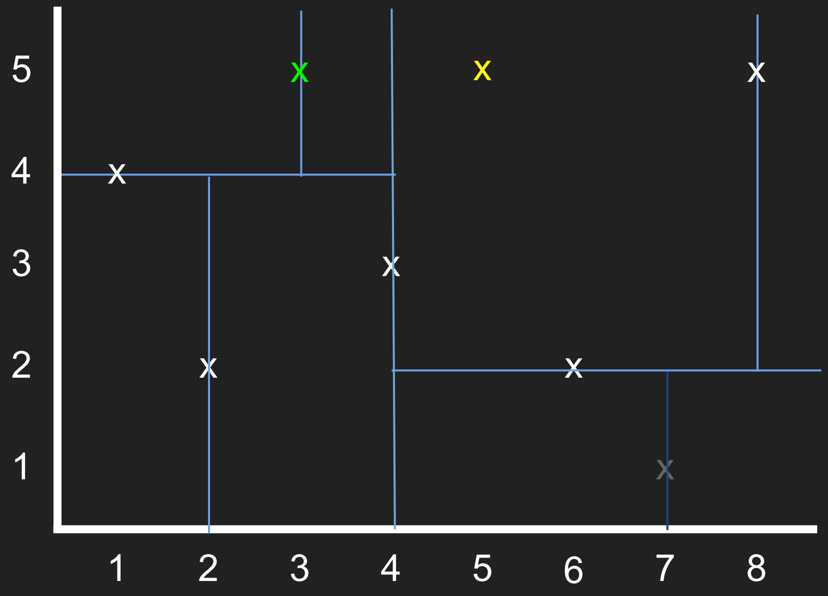


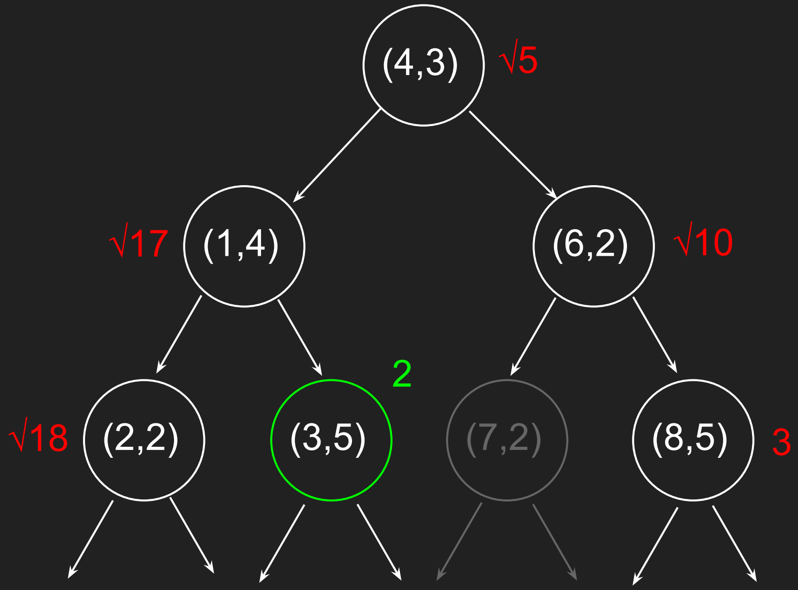
2d trees - Nearest Neighbor





2d trees - Nearest Neighbor



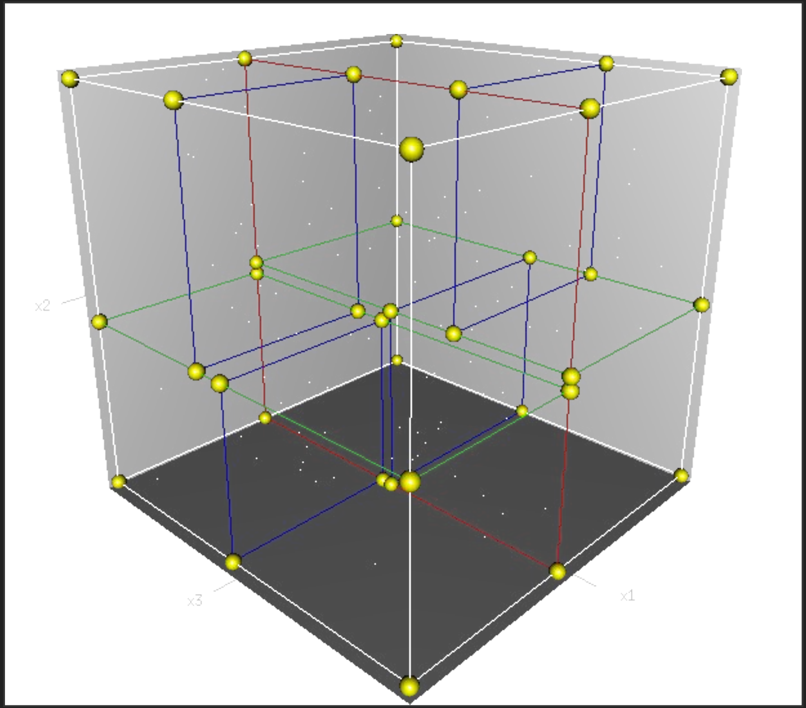


2d trees

- Construction complexity: *O(n log n)*
  - Complication: how do you decide how to partition?
  - Requires you be smart about picking pivots
- Adding/Removing element: *O(log n)*
  - This is because we know it's balanced from the median selection
  - ...except adding/removing might make it unbalanced -- there are variants that handle this
- Nearest Neighbor
  - Average case: *O(log n)*
  - Worst case: *O(n)*
    - Not great… but not worse than brute force
- Can also be used effectively for range finding

k-d trees

- 2d trees can be extended to *k* dimensions



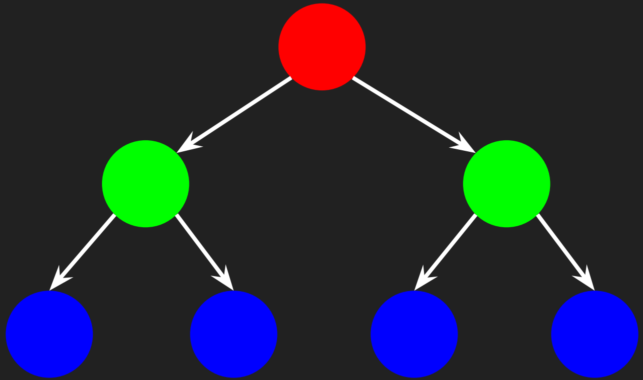


Image Source: Wikipedia

k-d trees

- Same algorithm for nearest neighbor!
    - Remember sabermetrics!
    - ...except there's a catch
- Curse of Dimensionality
    - The higher the dimensions, the "sparser" the data gets in the space
    - Harder to rule out portions of the tree, so many searches end up being fancy brute forces
    - In general, *k*-d trees are useful when $N >> 2^k$

Sabermetrics (reprise)

- Finding single neighbor could be noise-prone
  - Are we sure this year’s “Derek Jeter” will be next year’s too?
  - What if there are lots of close points… are we sure that the relative distance matters?
  - Could ask for set of most likely players

- Alternate question: will a player make it to the Hall of Fame?
  - Still k-dimensional space, but we’re not comparing with individual point
  - Am I in the “neighborhood” of Hall of Famers?
  - Classic example of “classification problem”

k-Nearest Neighbors (kNN)

- New plan: find the *k* closest points
  - Each can “vote” for a classification
  - …or you can do some other kind of averaging
- Can we modify our k-d tree NN algorithm to do this?
  - Track *k* closest points in max-heap (priority queue)
    - Keep heap at size *k*
  - Only need to consider *k*’th closest point for tree pruning

Voronoi Diagrams

- Useful visualization of nearest neighbors
  - Good when you have a known set of comparison points
- Wide ranging applications
  - Epidemiology
    - Cholera victims all near one water pump
  - Aviation
    - Nearest airport for flight diversion
  - Networking
    - Capacity derivation
  - Robotics
    - Points are obstacles, edges are safest paths

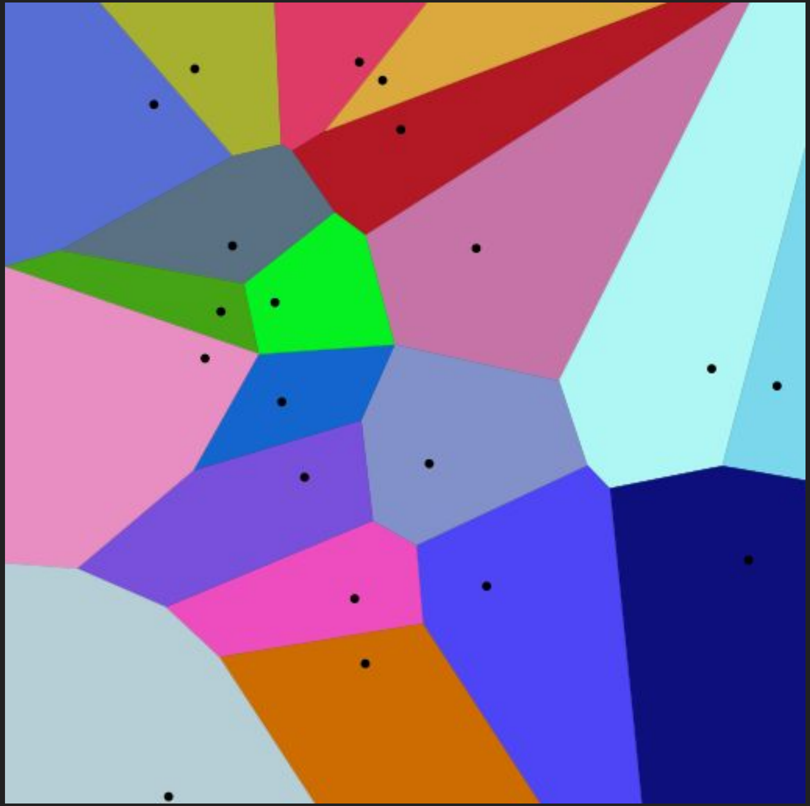
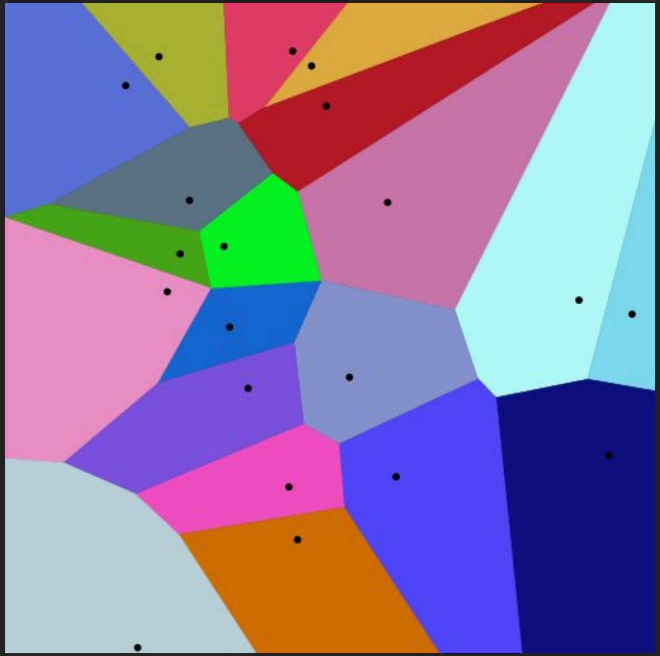


Image Source: Wikipedia

Voronoi Diagrams

- Also helpful for visualizing effects of different distance metrics

Euclidean distance



Manhattan distance

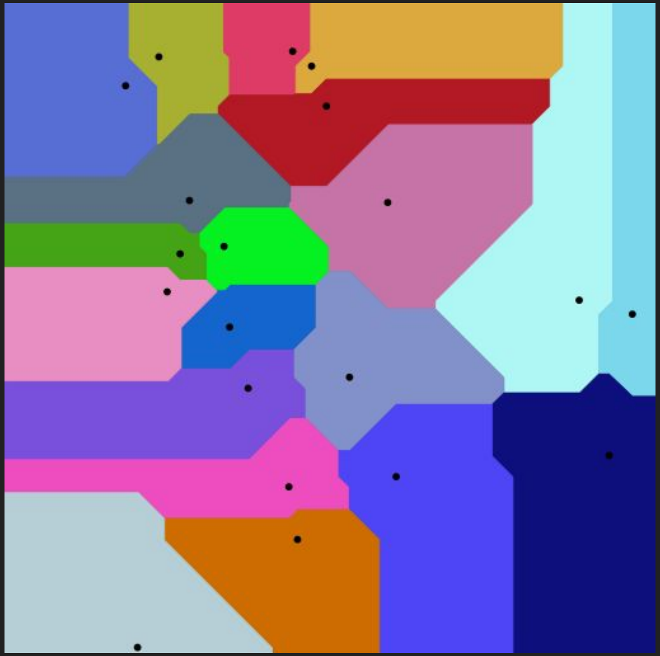


Image Source: Wikipedia

Voronoi Diagrams

- Polygon construction algorithm is a little tricky, but conceptually you can think of expanding balls around the points

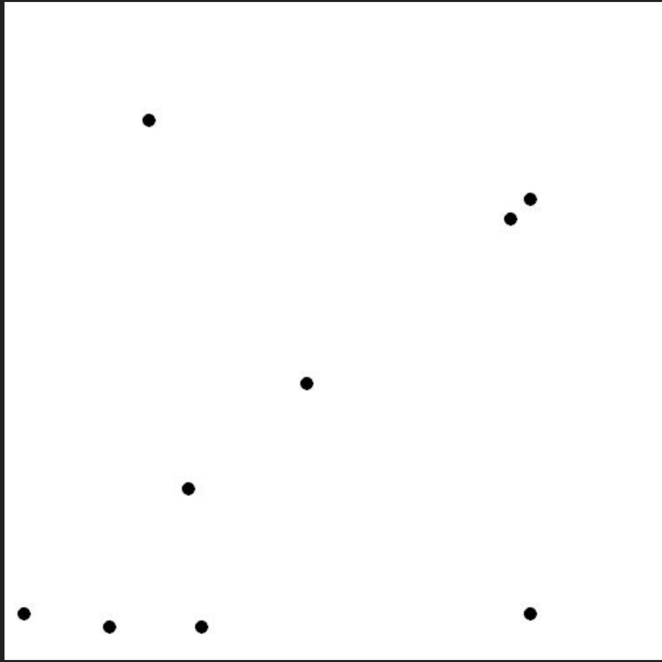


Image Source: Wikipedia

k-means Clustering

- Goal: Group *n* data points into *k* groups based on nearest neighbor

Algorithm:

1. Pick *k* data points at random to be starting “centers,” call each center c_i
2. For each node *n*, calculate which of the *k* centers is the nearest neighbor and add it to set S_i
3. Compute the mean of all points in S_i to generate a new c_i
4. Go back to (2) and repeat with the new centers, until the centers converge

k-means clustering

Notice: the algorithm basically creates Voronoi diagrams for the centers!

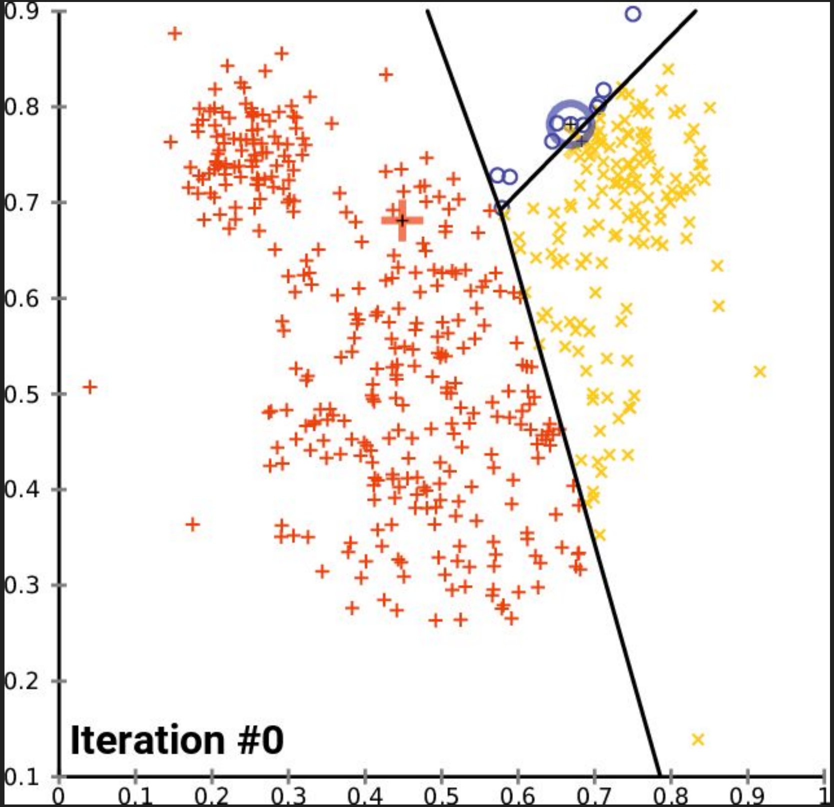


Image Source: Wikipedia

k-means clustering

- Does this always converge?
  - Depends on distance function. Generally yes for Euclidean
  - Converges quickly in practice, but worst case can take an exponential number of iterations
- Does it give the optimal clustering?
  - NO! Well, at least not always.

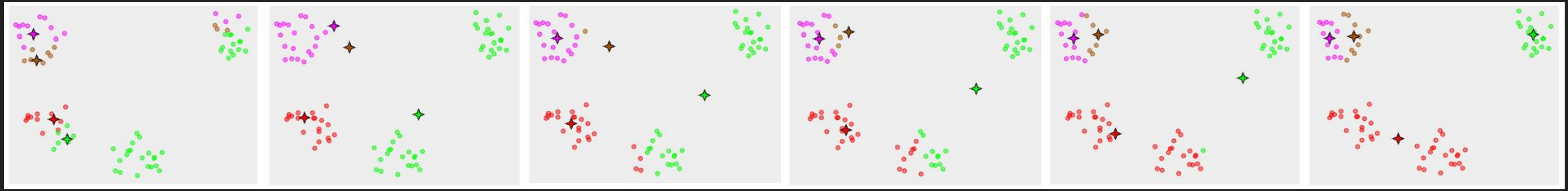


Image Source: Wikipedia

Other space partitioning data structures

- Leaf point *k*-d trees
  - Only stores points in leaves, but leaves can store more than one point
  - Split space at the middle of longest axis
  - Effectively “buckets” points - can be used for approximate nearest neighbor
- Quadtrees
  - Split space into quadrants (i.e. every tree node has four children)
  - Quadrant can only contain at most q nodes
    - If there are more than q, split that quadrant again into quadrants
  - Applications
    - Collision detection (video games)
    - Image representation/processing (transforming/comparing/etc. nearby pixels)
    - Sparse data storage (spreadsheets)
  - Octrees are extension to 3d