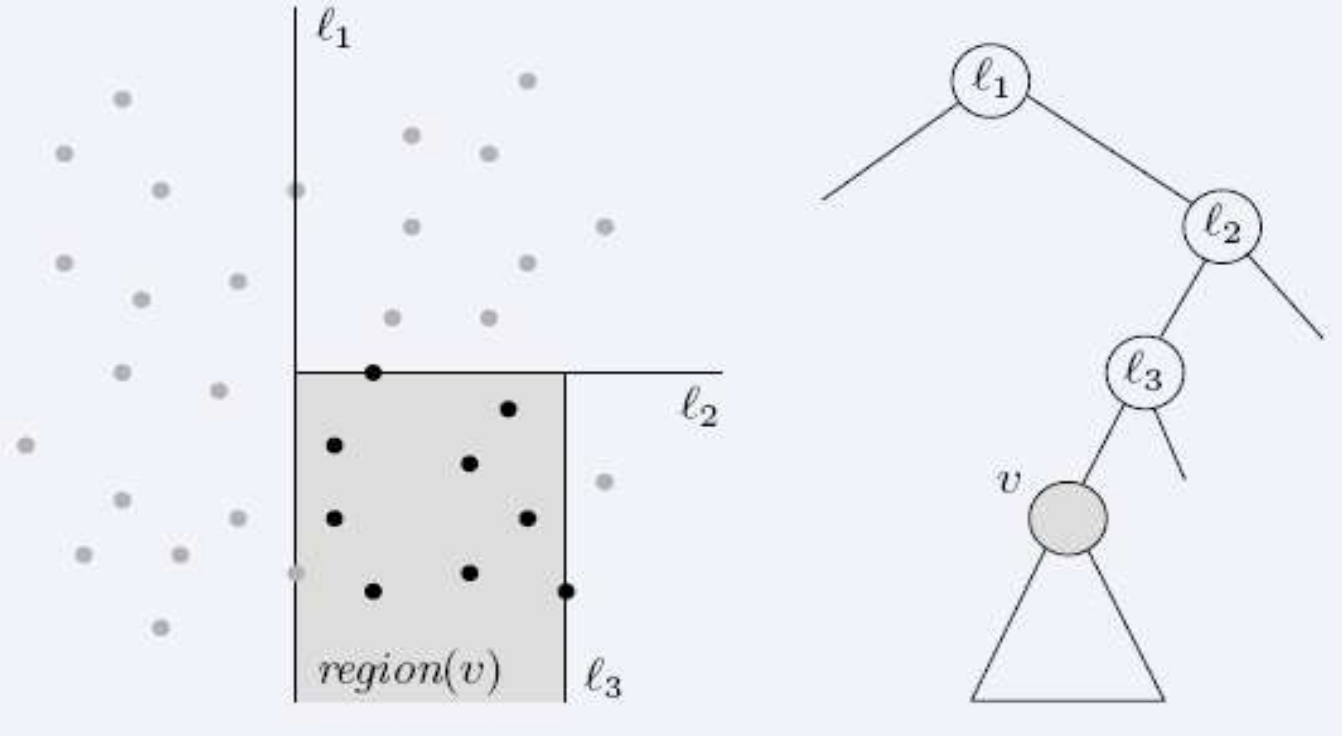
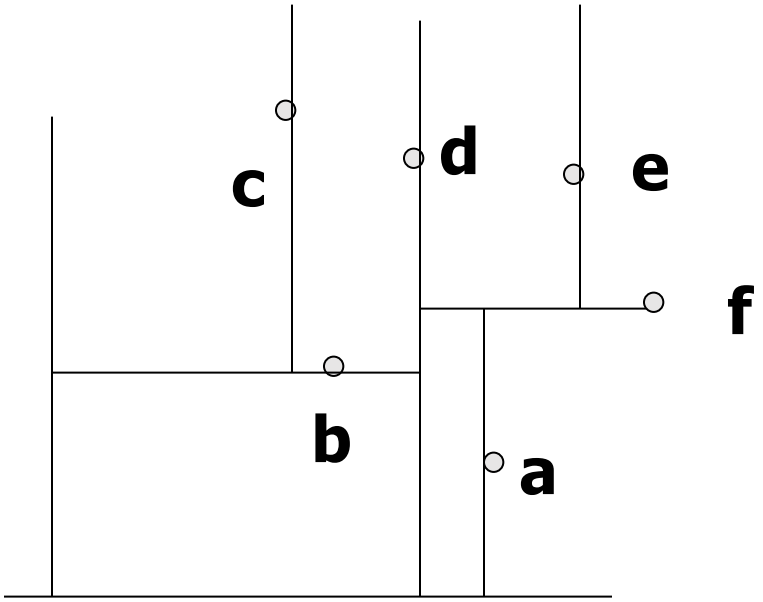
Range Query

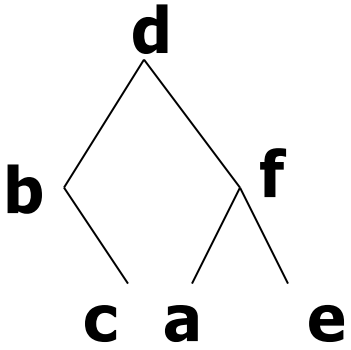
- Range-searching in **2-d**
  - Given a set of **n** points, build a data structure that for any query rectangle **R** reports all point in **R**



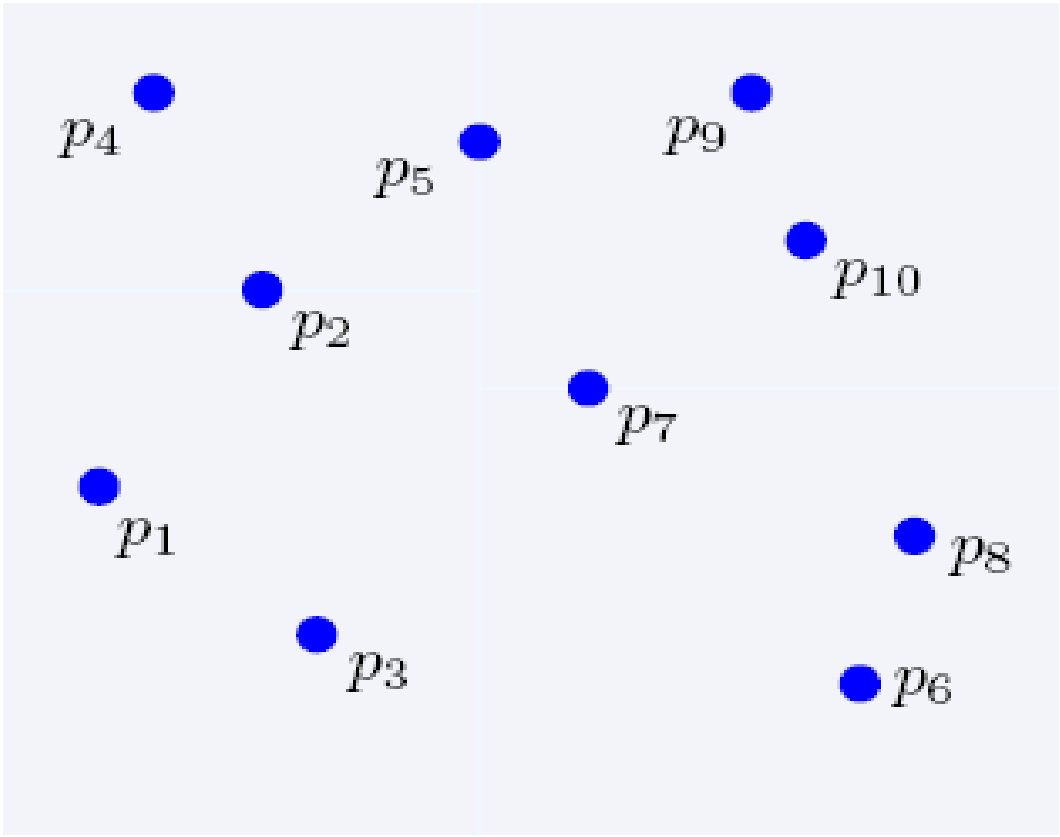
K-d Tree

- A recursive space partitioning tree to perform search in 2D space
  - Partition along x and y axis in an alternating fashion.
  - Each internal node stores the splitting node along x (or y).

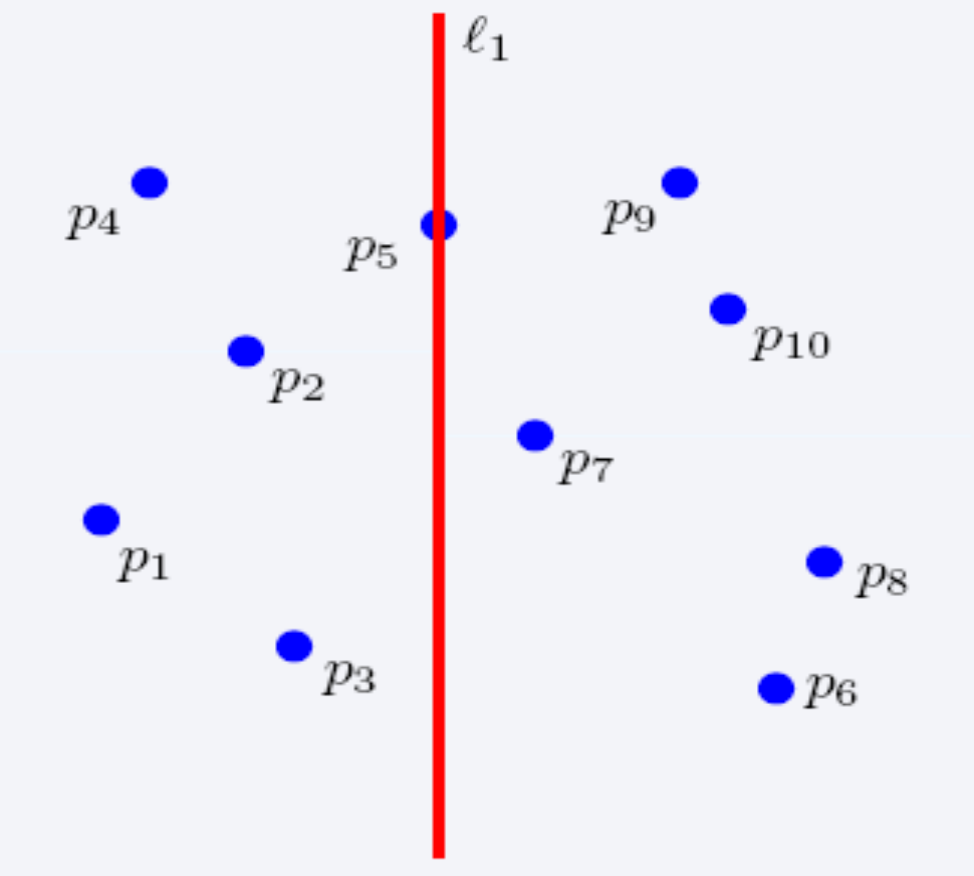




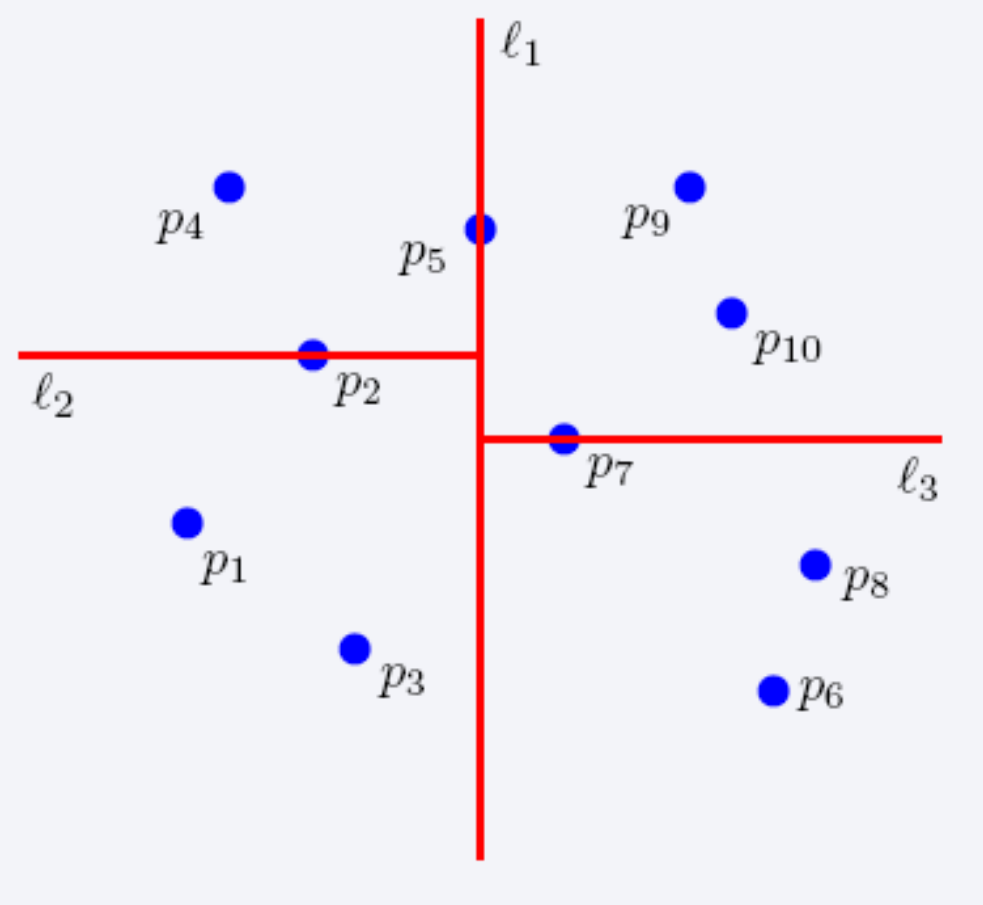
Construction of KD-Trees



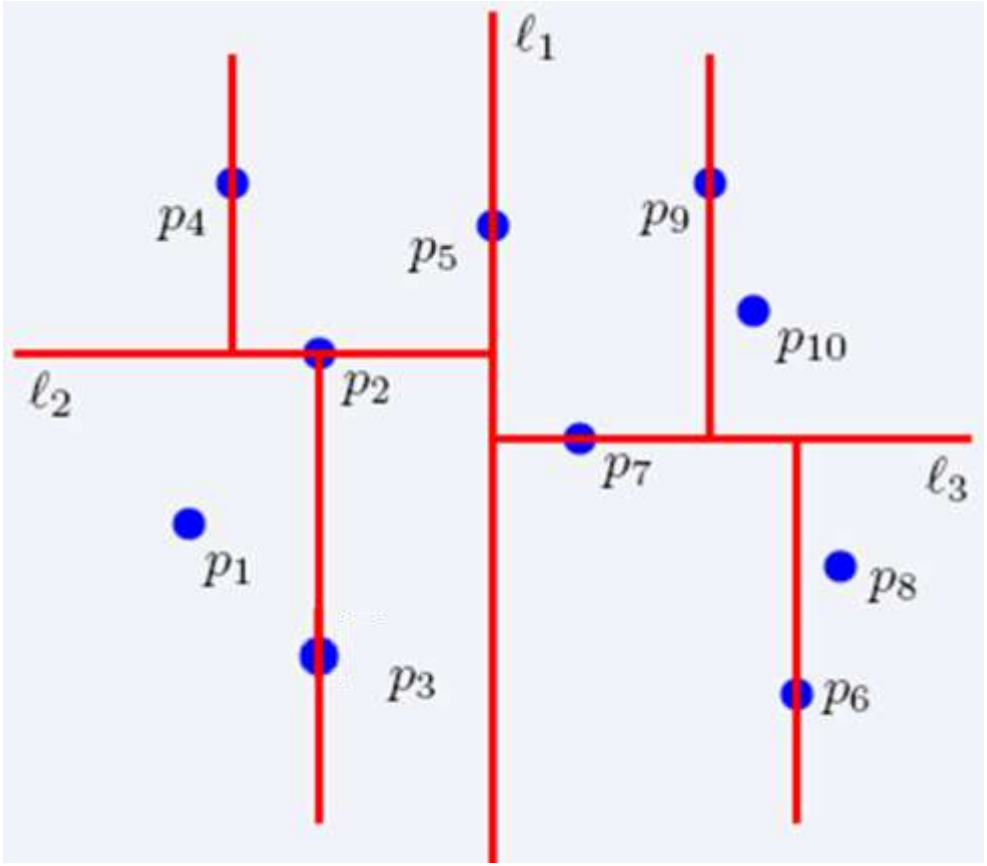
Construction of kd-trees



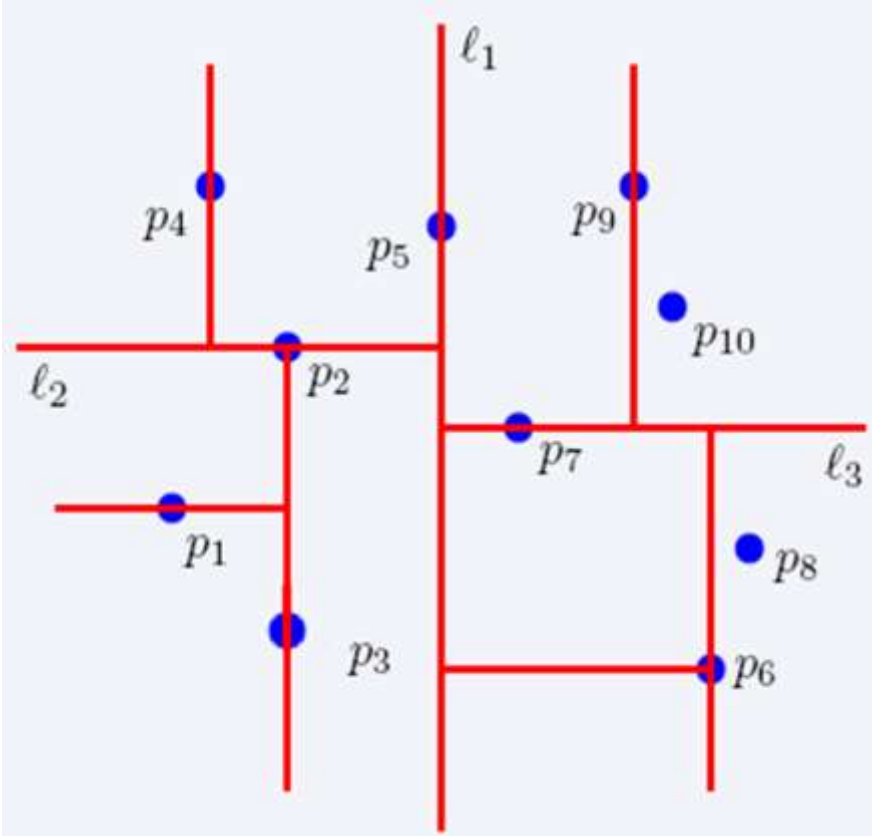
Construction of kd-trees



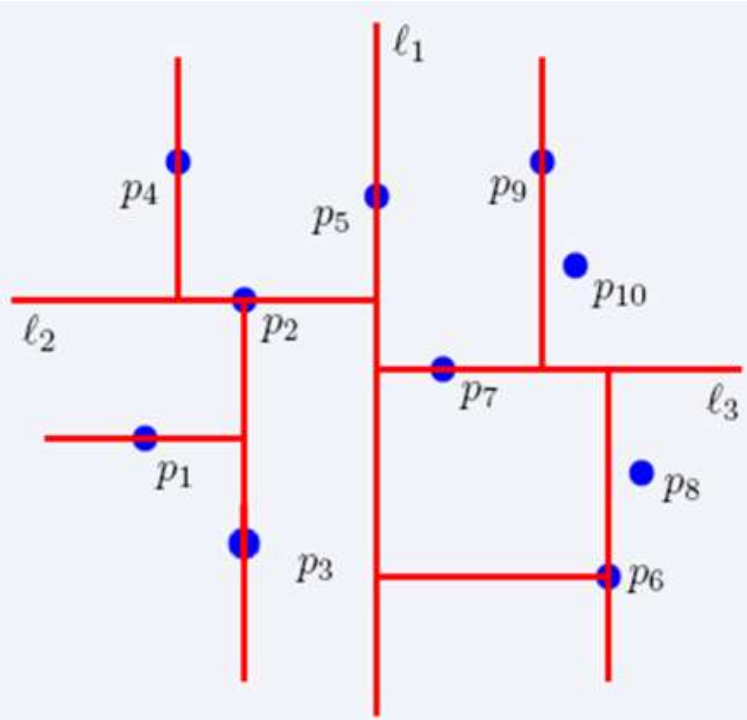
Construction of kd-trees

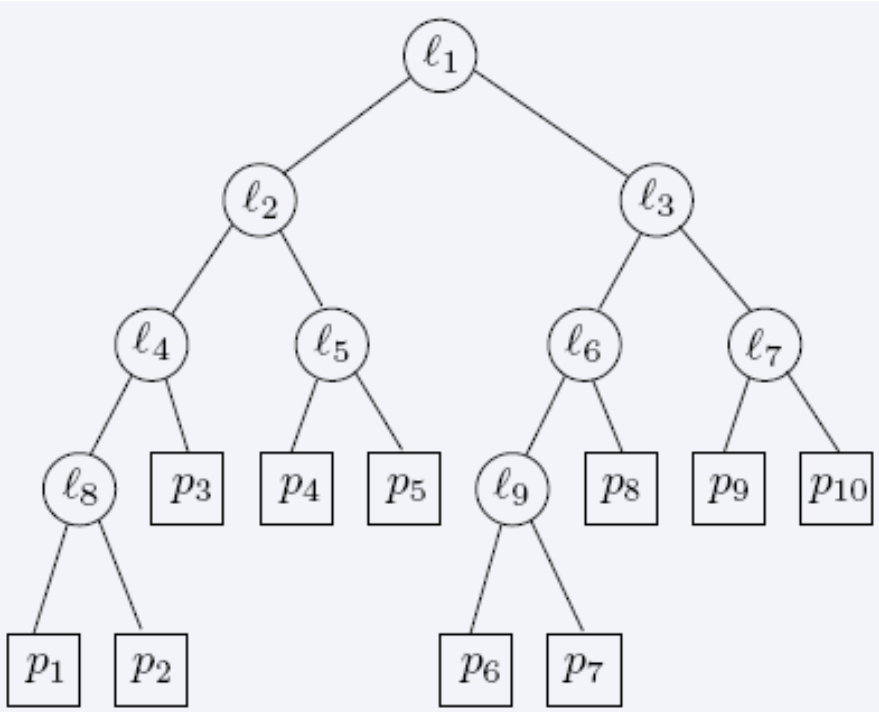


Construction of kd-trees



The complete kd-tree

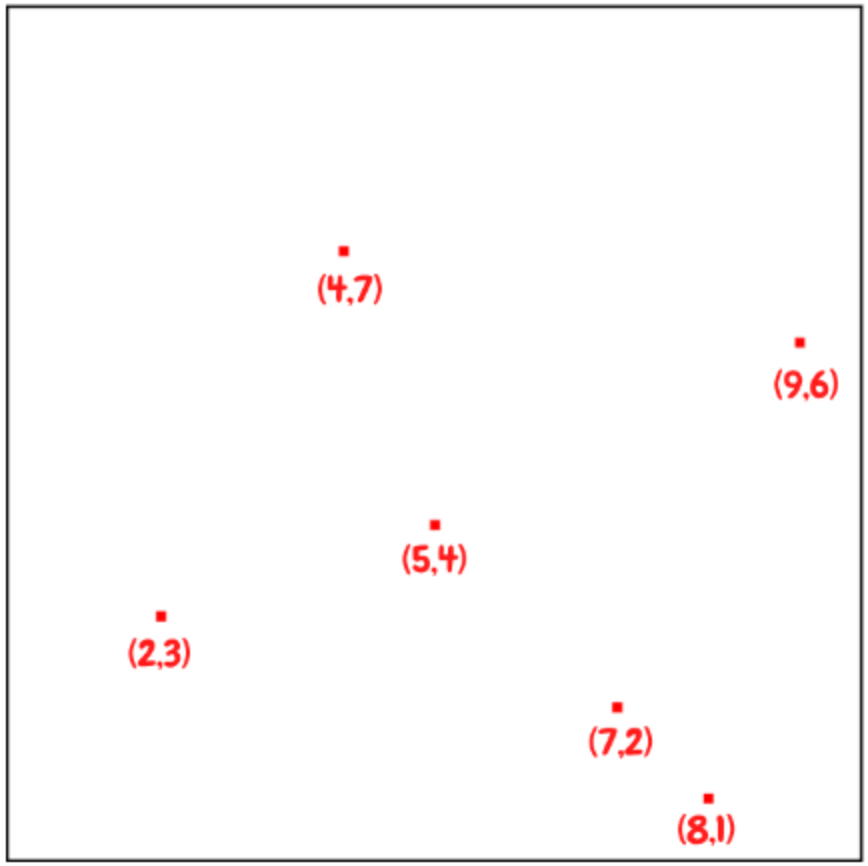




KD TREE

(7,2), (5,4), (2,3), (4,7), (9,6), (8,1),

(2,3), (5,4), (4,7), (7,2), (8,1), (9,6)
First, insert (7,2) since it is the median along the x

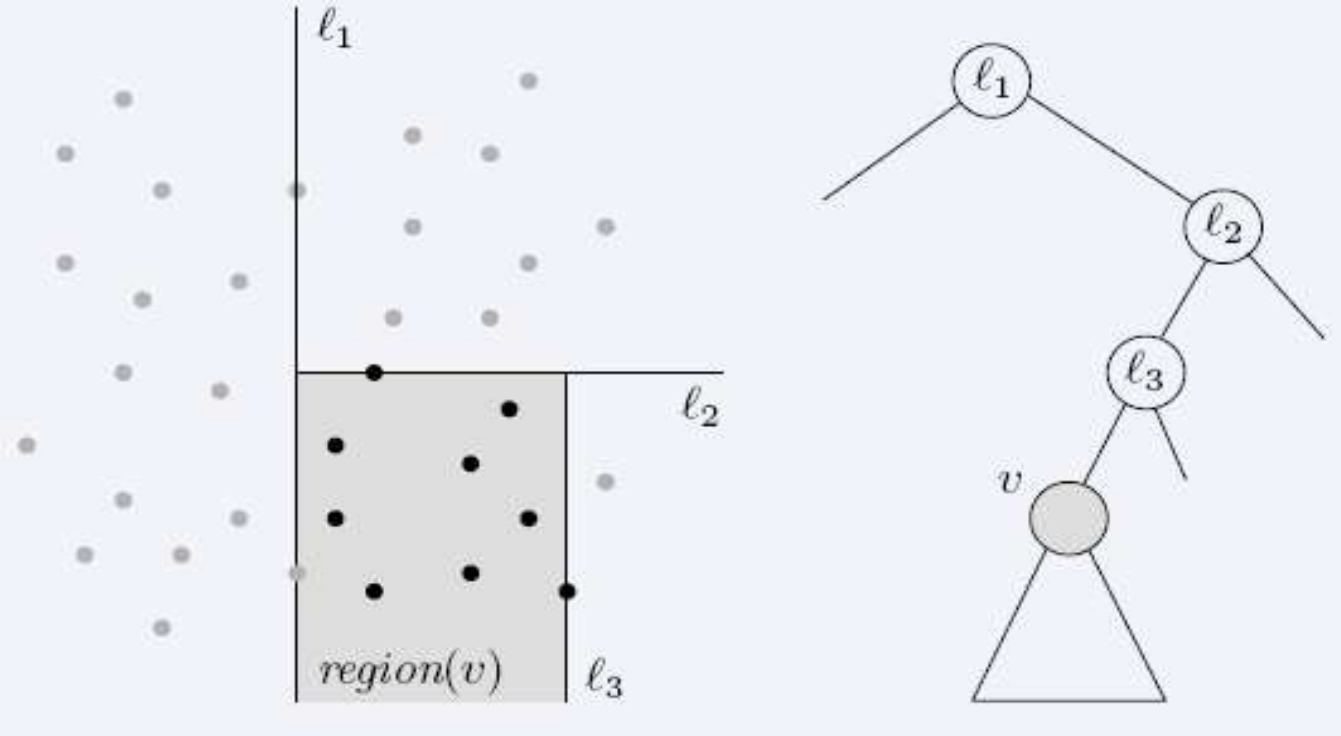


2D KD-Trees

- **Algorithm:**
  - Choose **x** or **y** coordinate (alternate)
  - Choose the median of the coordinate; this defines a horizontal or vertical line
  - Recurse on both sides

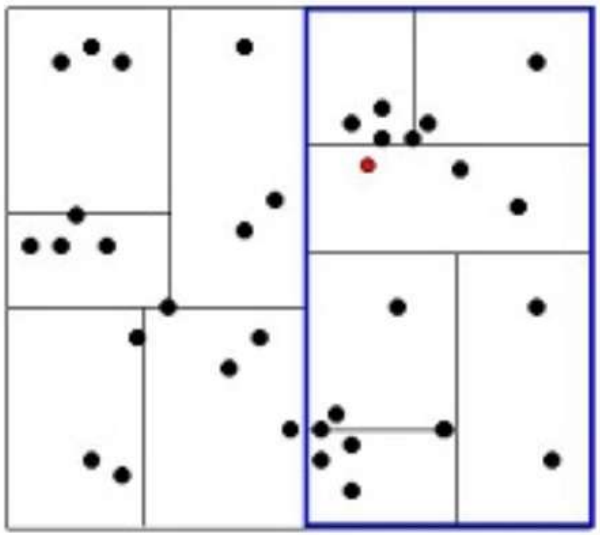
- We get a binary tree:
  - Size **O(n)**
  - Depth **O(logn)**
  - Construction time **O(nlogn)**

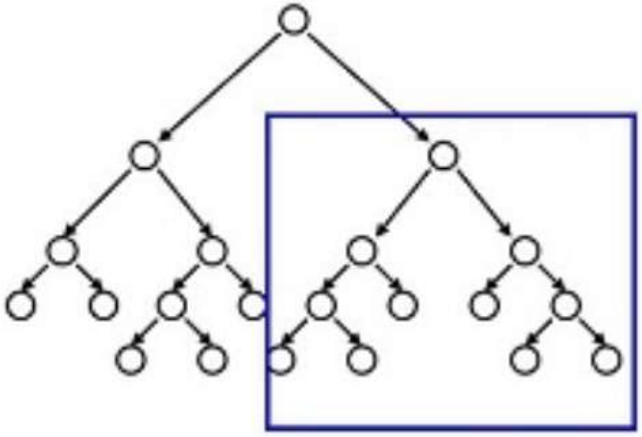
Region of node v



Region(v) : the subtree rooted at v stores the points in black dots

Nearest Neighbor with KD Trees





Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

Insert Node in KD-tree

- One adds a new point to a *k*d-tree in the same way as one adds an element to any other search tree.

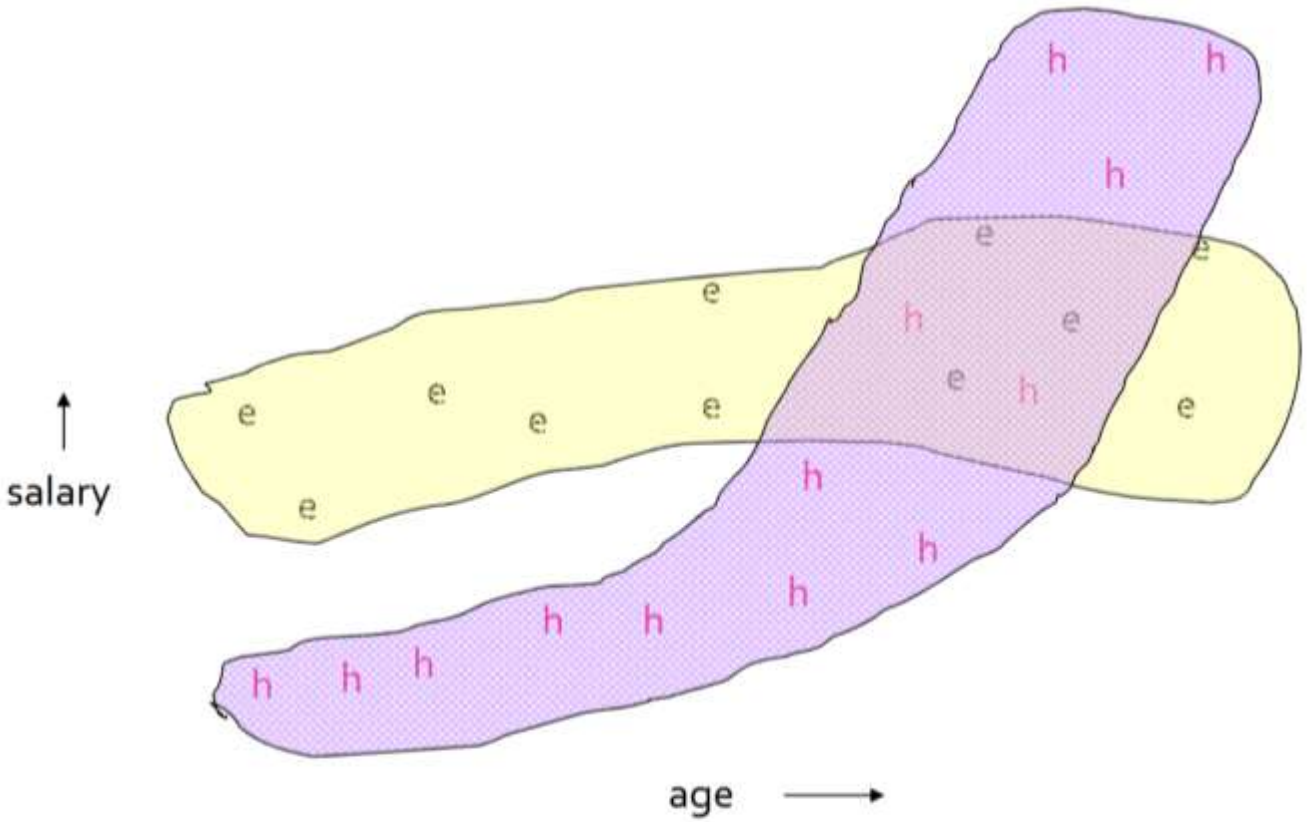
- Traverse the tree, starting from the root
  - Move either to the left or the right child depending on whether the point to be inserted is on the "left" or "right"

CURE (Clustering Using Representatives)

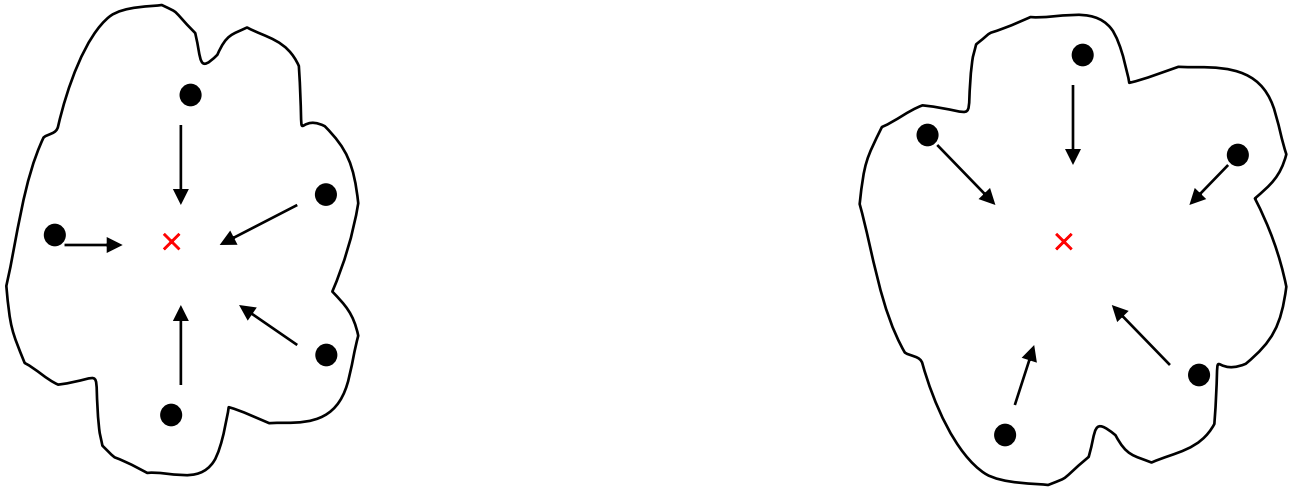
Pros

- Can detect arbitrary shaped clusters
- Can detect clusters with non-uniform sizes
- Can handle large datasets
- Can identify outliers

CURE



CURE: Representative Points



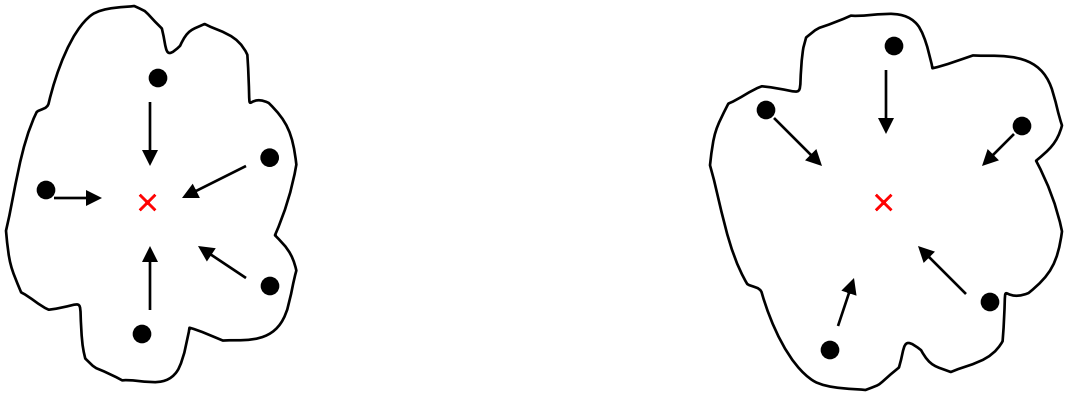
- Uses a number of points to represent a cluster

CURE: Representative Points

- Uses a number of points to represent a cluster
  - **c** well scattered points within the cluster are chosen
  - The ***representative points*** help to maintain the shape of the cluster

It is similar to **hierarchical clustering approach**. But it use sample point variant as the cluster representative rather than every point in the cluster.

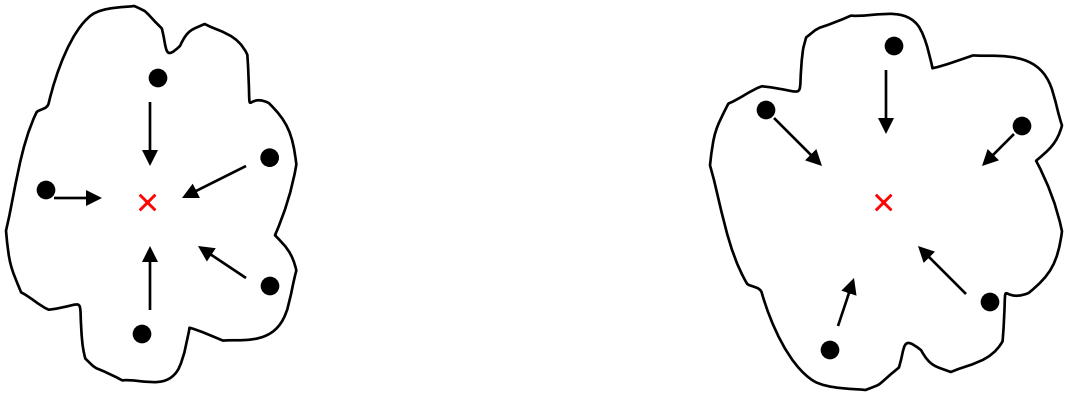
CURE: Representative Points



- Uses **c** well scattered points within the cluster to maintain the shape of the cluster
- The chosen scattered points are shrunk toward the mean (centroid) in a fraction of α where $\mathbf{0 \leq \alpha \leq 1}$
  - Shrinking them to the centroid help to eliminate the outliers

$$p + \alpha * (w.\text{mean-}p)$$

CURE: Representative Points

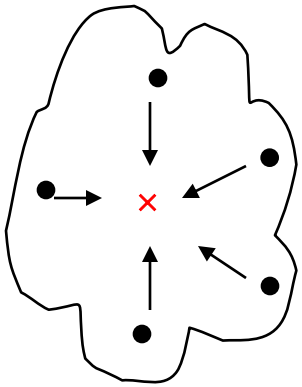


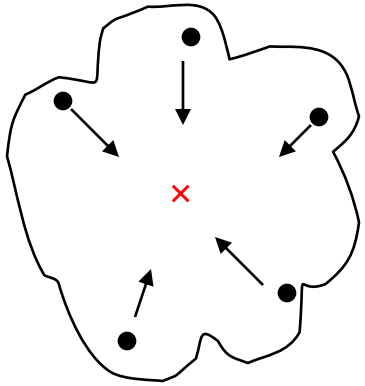
- The chosen scattered points are shrunk toward the mean (centroid) in a fraction of α where $0 \leq \alpha \leq 1$
  - Shrinking them to the centroid help to eliminate the outliers

$$p + \alpha * (w.\text{mean} - p)$$

- **A smaller value of the alpha shrink points very little and favors elongated clusters.**
- **While for the large value of alpha, the scattered points get closer to the center and cluster are more compact.**

CURE: Representative Points

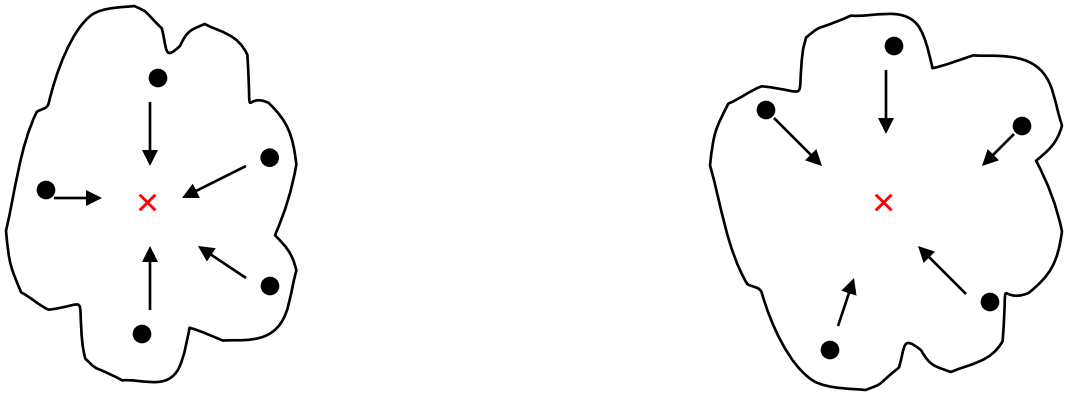




$p + \alpha * (w.\text{mean-}p)$

- Shrinking c representative points to the centroid help eliminate the outliers
- **HOW ?**
  - Outliers are farthest away from the cluster center
  - As a result the shrinking would cause the points to move more towards cluster center as compared to the other points.
  - The largest shift in outlier would reduce the ability to merge the wrong clusters

CURE: Representative Points

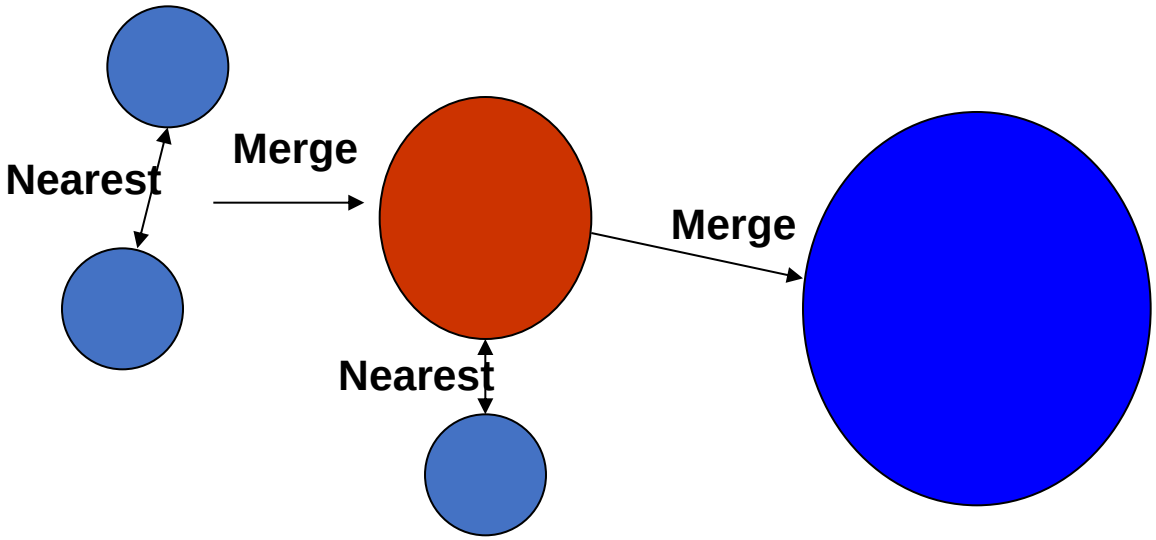


- We use **c** points to represent a cluster and maintain its shape
- We shrink c representative points to the centroid to eliminate the outliers
- **Only representative points are used to calculate cluster distance from other clusters**

Cluster similarity is the similarity of the closest pair of representative points from different clusters

General CURE clustering procedure

- Only representative points are used to calculate cluster distance from other clusters
- Cluster similarity is the similarity of the closest pair of representative points from different clusters
- After each merging, c sample points will be selected from original representative of previous clusters to represent new cluster.
- Cluster merging will be stopped until target k cluster is found



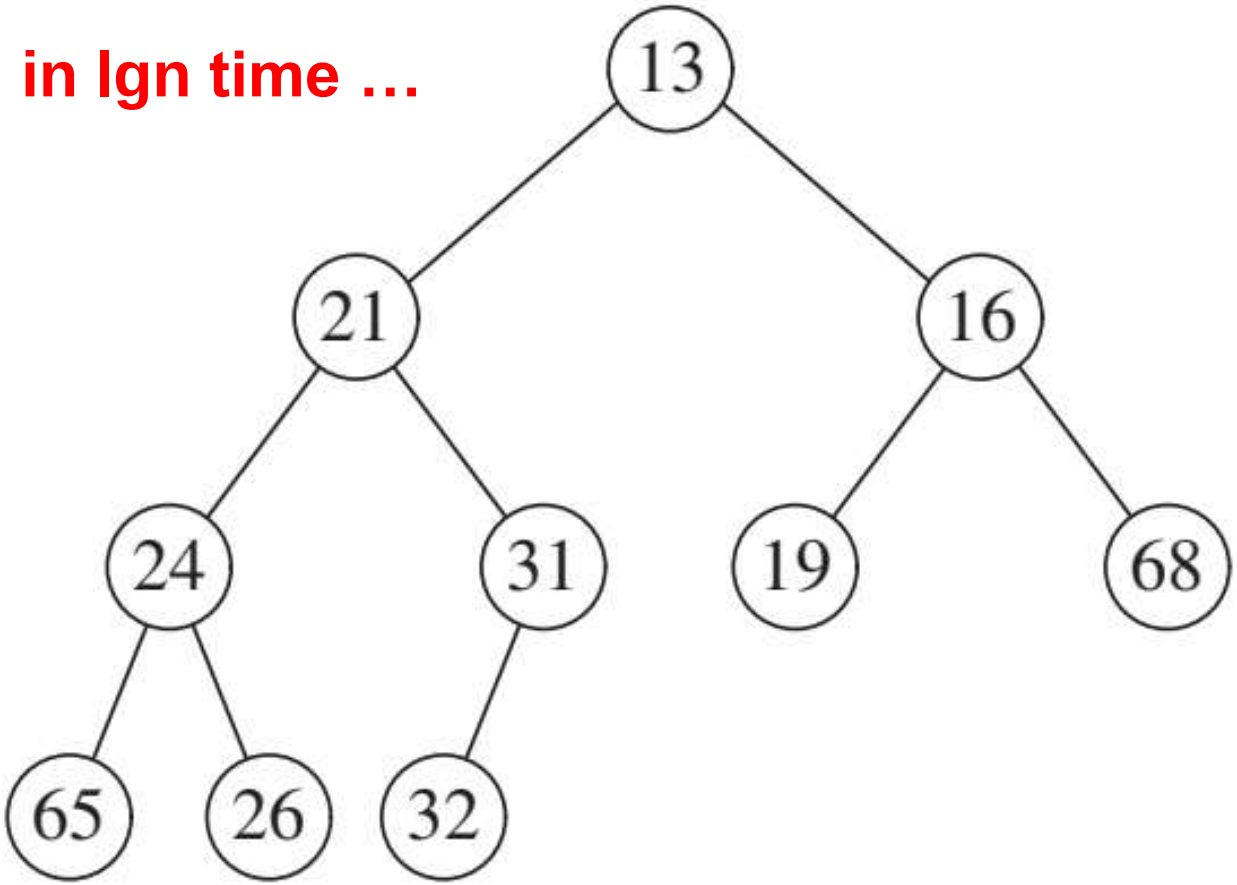
```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
```

Pseudocode of the basic idea of CURE

Min Heap

Heap is a binary tree and the value stored at a node is less or equal to the values stored at the children.

Can give min in lgn time …



```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
3.  while size(Q) > k do {
4.      u := extract_min(Q)
5.      v := u.closest
6.      delete(Q, v)
7.      w := merge(u, v)
```

Pseudocode of the basic idea of CURE

```
procedure merge(u, v)
begin
1.  w := u ∪ v
2.  w.mean := (|u|u.mean+|v|v.mean) / (|u|+|v|)
3.  tmpSet := ∅
4.  for i := 1 to c do {
5.      maxDist := 0
6.      foreach point p in cluster w do {
7.          if i = 1
8.              minDist := dist(p, w.mean)
9.          else
10.             minDist := min{dist(p, q) : q ∈ tmpSet}
11.         if (minDist ≥ maxDist){
12.             maxDist := minDist
13.             maxPoint := p
14.         }
15.     }
16.     tmpSet := tmpSet ∪ {maxPoint}
17. }
18. foreach point p in tmpSet do
19.     w.rep := w.rep ∪ {p + α*(w.mean-p) }
20. return w
end
```

Pseudocode of the Merge Procedure

$\bar{p} + \alpha*(w.\text{mean}-p)$

```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
3.  while size(Q) > k do {
4.      u := extract_min(Q)
5.      v := u.closest
6.      delete(Q, v)
7.      w := merge(u, v)
8.      delete_rep(T, u); delete_rep(T, v); insert_rep(T, w)
```

Pseudocode of the basic idea of CURE

The worst-case time complexity is *O(n²logn)*
The space complexity is *O(n)*

```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
3.  while size(Q) > k do {
4.      u := extract_min(Q)
5.      v := u.closest
6.      delete(Q, v)
7.      w := merge(u, v)
8.      delete_rep(T, u); delete_rep(T, v); insert_rep(T, w)
9.      w.closest := x /* x is an arbitrary cluster in Q */
10.     for each x ∈ Q do {
11.         if dist(w, x) < dist(w, w.closest)
12.             w.closest := x
```

Pseudocode of the basic idea of CURE

```
procedure cluster(S, k)
begin
1.  T := build_kd_tree(S)
2.  Q := build_heap(S)
3.  while size(Q) > k do {
4.      u := extract_min(Q)
5.      v := u.closest
6.      delete(Q, v)
7.      w := merge(u, v)
8.      delete_rep(T, u); delete_rep(T, v); insert_rep(T, w)
9.      w.closest := x /* x is an arbitrary cluster in Q */
10.     for each x ∈ Q do {
11.         if dist(w, x) < dist(w, w.closest)
12.             w.closest := x
13.         if x.closest is either u or v {
14.             if dist(x, x.closest) < dist(x, w)
15.                 x.closest := closest_cluster(T, x, dist(x, w))
16.             else
17.                 x.closest := w
18.             relocate(Q, x)
19.         }
20.         else if dist(x, x.closest) > dist(x, w) {
21.             x.closest := w
22.             relocate(Q, x)
23.         }
24.     }
25.     insert(Q, w)
26. }
end
```

Pseudocode of the basic idea of CURE

The worst-case time complexity is $O(n^2 \log n)$
The space complexity is $O(n)$

Improved CURE --Random Sampling

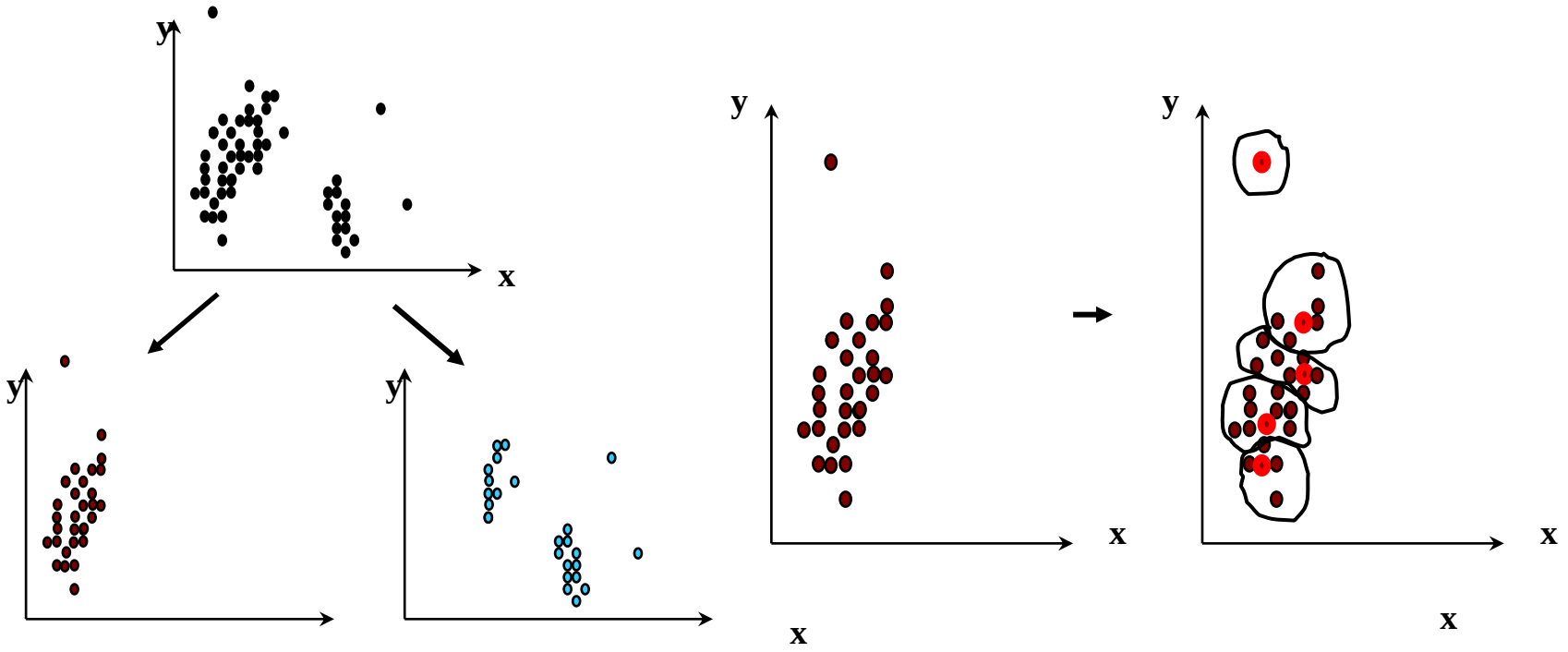
- For large data sets,
  - we can’t store every data point to the memory.
  - Data merging also requires a very long time.
- We use random sampling to reduce the time complexity and memory usage.

Improved Cure: use partitioning

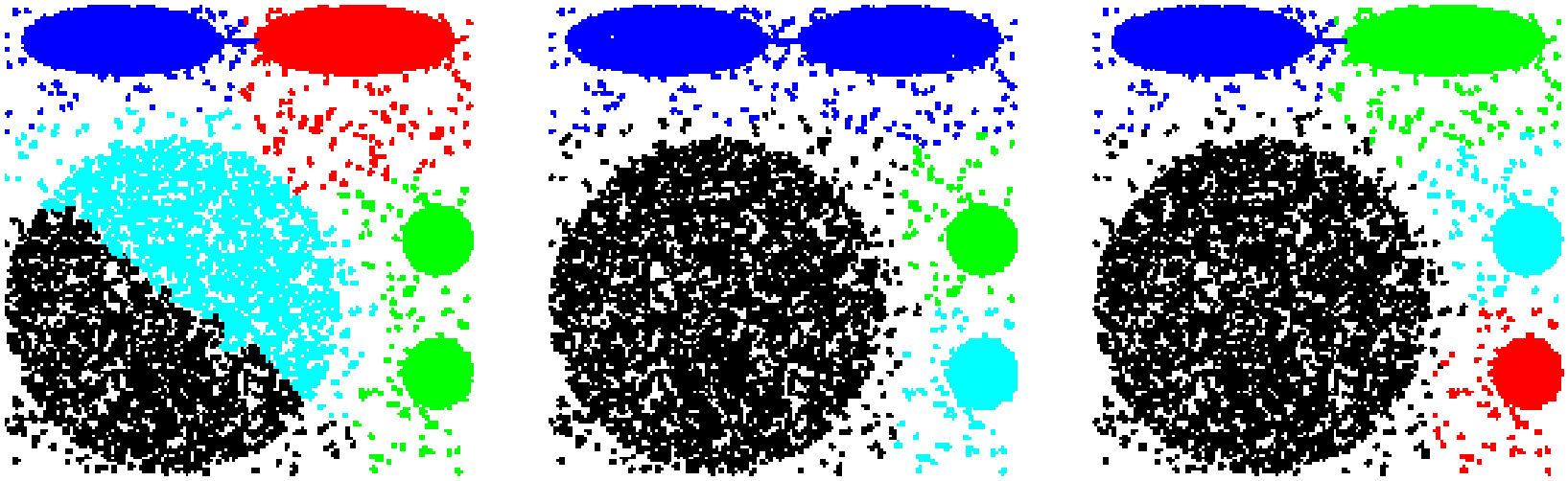
- Draw random sample *s*.
- Partition sample to *p* partitions with size *s/p*
- Partially cluster partitions into *s/pq* clusters
- Use CURE to cluster until K cluster remains
- Eliminate outliers
  - By random sampling
  - If a cluster grows too slow, eliminate it.
- **Cluster partial clusters**.
- Label data in disk

Data Partitioning and Clustering

- s = 50
- p = 2
- s/p = 25



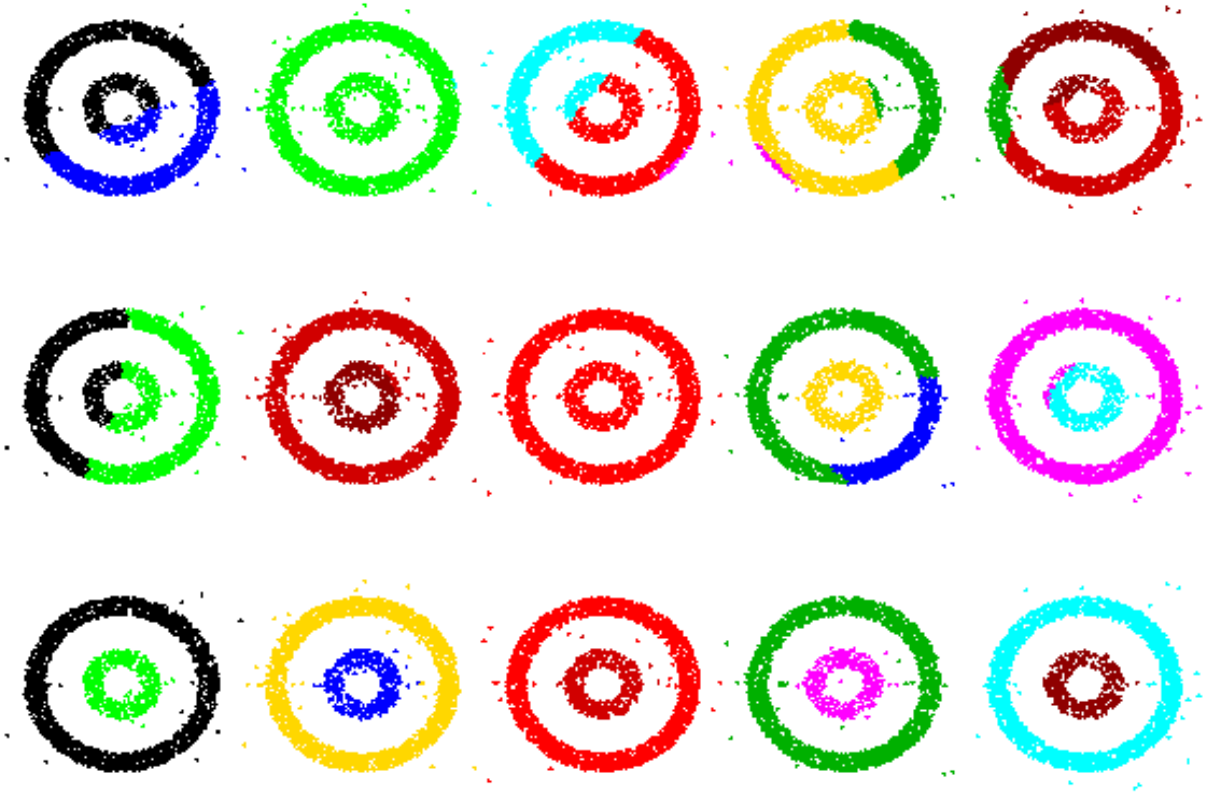
Experimental Results: CURE



a) BIRCH b) MST METHOD c) CURE

Picture from *CURE*, Guha, Rastogi, Shim.

Experimental Results: CURE



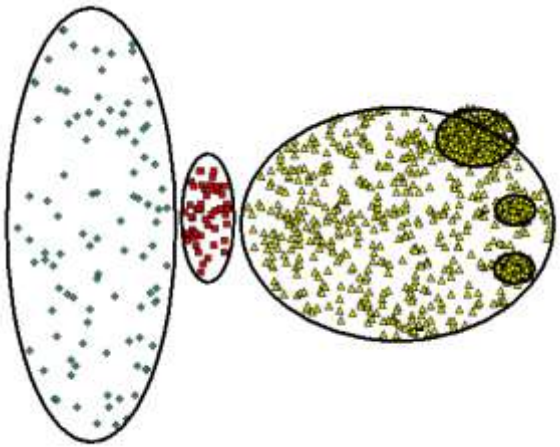
a) BIRCH
(centroid)

b) MST METHOD

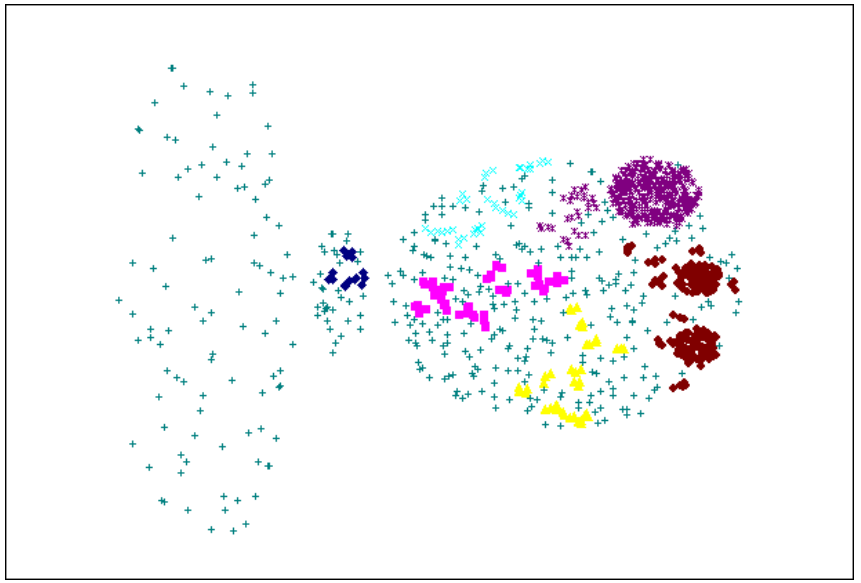
c) CURE

Picture from *CURE*, Guha, Rastogi, Shim.

CURE Cannot Handle Differing Densities



Original Points



CURE

Problems and Challenges

- Considerable progress has been made in scalable clustering methods
  - Partitioning: k-means, k-medoids, CLARANS
  - Hierarchical: BIRCH, CURE
  - Density-based: DBSCAN, CLIQUE, OPTICS
  - Grid-based: STING, WaveCluster
  - **Model-based: Autoclass, Denclue, Cobweb**
- Current clustering techniques do not address all the requirements adequately
- Constraint-based clustering analysis: Constraints exist in data space (bridges and highways) or in user queries