Clustering
(Chapter 7)

INF 553

Wensheng Wu

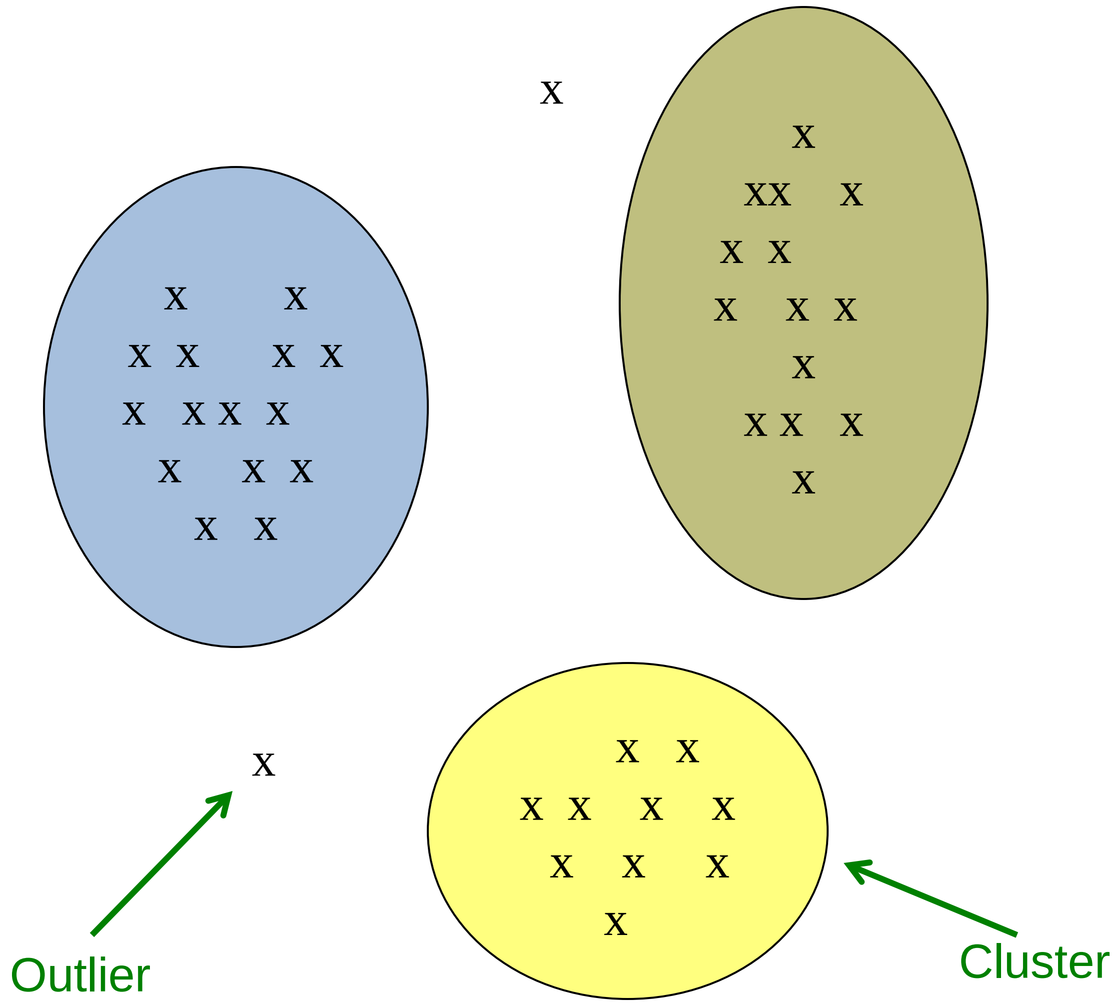
Roadmap

- Problem, types of algorithms, and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR: extend k-means to handle large data set
  - CURE
- Curse of dimensionality

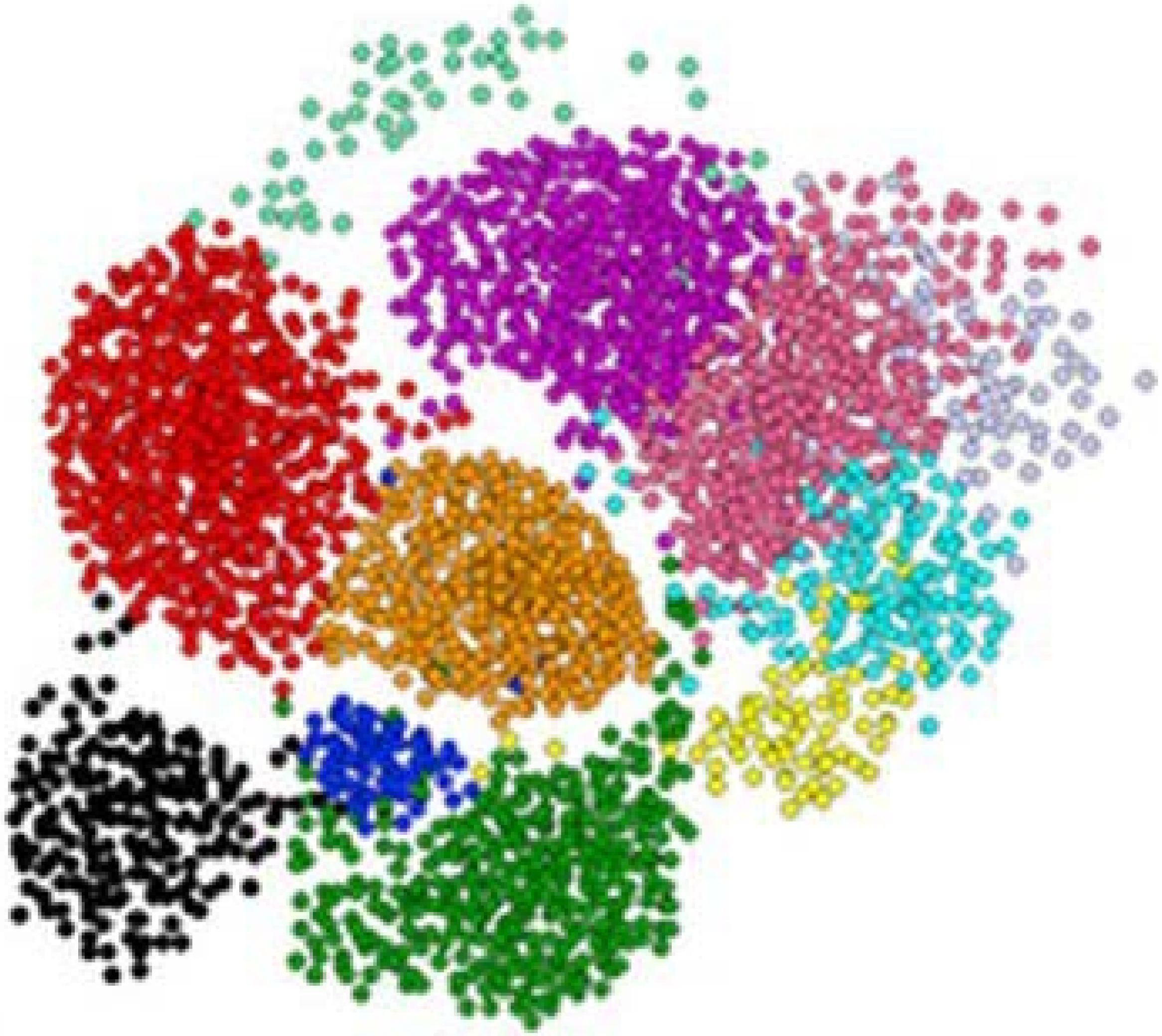
Clustering Problem

- Given a set of objects and a distance function
- Find groups/clusters of objects
- Desired properties:
  - Objects in the same group are close to each other
  - Objects in different groups are far away from each other

Example



Clustering can be hard



Clustering Stars

- Each represented by 7-dimensional point
  - Dimension = frequency band
  - Point = radiation signature

Clustering Music CDs

- CD represented by a set of its buyers
  - Similar CD’s have similar buyers
- Use LSH in clustering large # of sets
  - Use LSH to efficiently find similar sets
  - Compute pairwise similarities of sets
  - Use the similarities in clustering (e.g., hierarchical)
- Advantage:
  - avoid computing similarity of dissimilar sets

Clustering Documents

- Document D represented as a word vector
  - $(w_1, w_2, \ldots, w_k)$, where w_i = 1 (or tf or tf*idf) if it appears in D
- Measure similarity of document D_1 and D_2
  - Cosine(D_1, D_2)
- Similar documents likely on same topic

Types of Algorithms

- Hierarchical vs. point assignment
  - Hierarchical: Bottom-up iterative merging of clusters to form a multi-level clustering
  - Point assignment or partitional: one-level
- Euclidean or non-Euclidean
  - Cluster center/centroid makes sense only in Euclidean
- In-memory or not
  - In-memory: entire data can fit in main memory

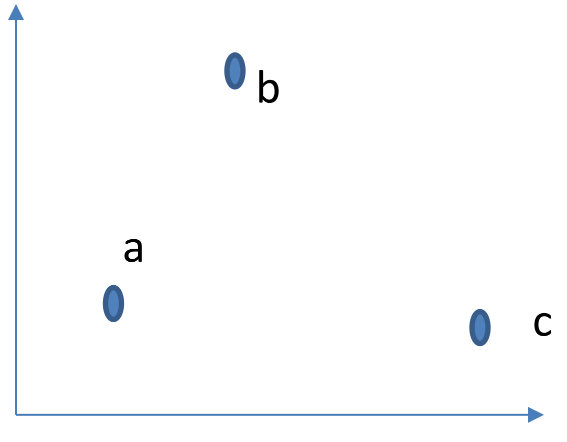
Varied Distance Functions

Distance function	Type of objects
Euclidean	Points in Euclidean space
Cosine	Vectors
Jaccard	Sets
Edit distance	Strings
Hamming distance	Bit vectors

Euclidean Distance

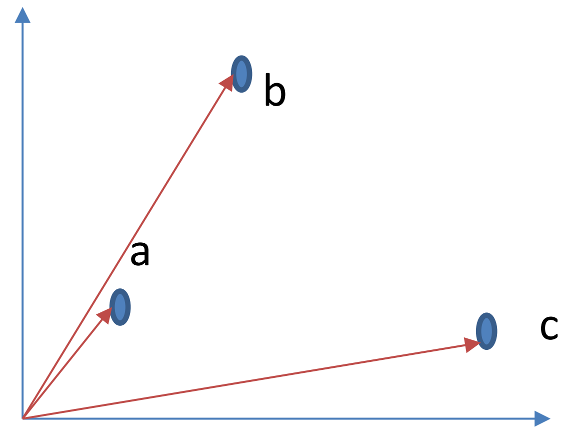
- Measures distance of two points in Euclidean space

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$



Cosine Distance

- Similarity = Cosine of angle btw vectors: A & B
- distance = 1- Cosine(A, B)



$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Edit Distance

- For string data, distance btw x and y =
  - Minimum number of insertions or deletions of characters that turn x into y
- x = abcde, y = acfdeg
  - Edit(x, y) = 3
    - Delete b
    - Insert f after c
    - Insert g after e

Hamming Distance

- For two bit vectors, distance btw x and y =
  - # of corresponding bits that differ

- x = 10101, y = 11110
  - Hamming(x, y) = 3

Roadmap

- Problem, types and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR
  - CURE
- Curse of dimensionality

Hierarchical Clustering

- Initially, a point is in a cluster by itself

How to pick and combine efficiently?

When to stop?

```
WHILE it is not time to stop DO
    pick the best two clusters to merge;
    combine those two clusters into one cluster;
END;
```

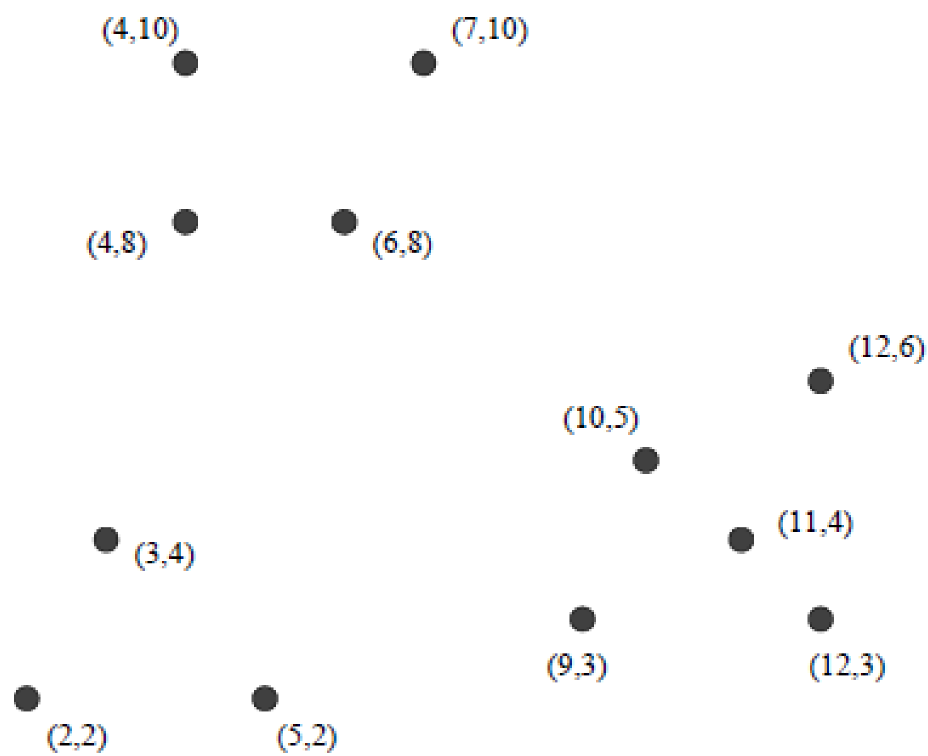
How to measure cluster distance?

Centroid-Based Distance

- Assume Euclidean space
- dist(C1, C2) = distance of their centroids
  - Coordinates of centroid = average of all points in the cluster
- C1: {(1, 2), (2, 4)}
  - Centroid = (1.5, 3)

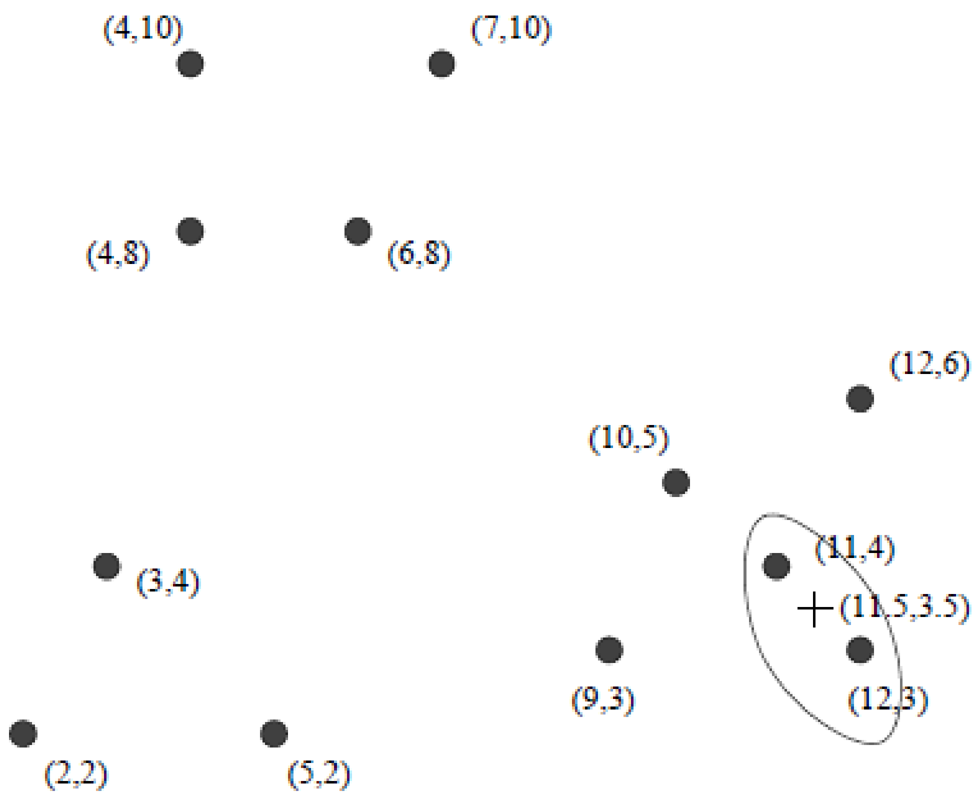
Example

- Which two points are closest?
  - What is their distance?

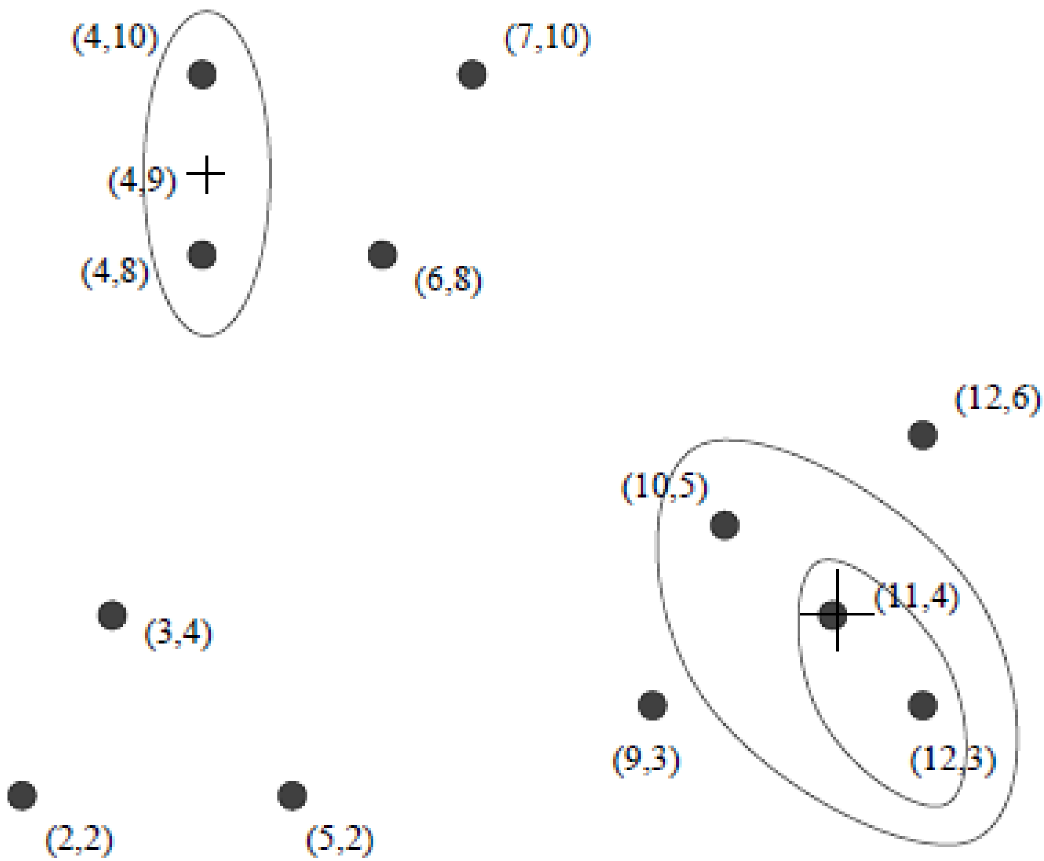


First merge, compute centroid

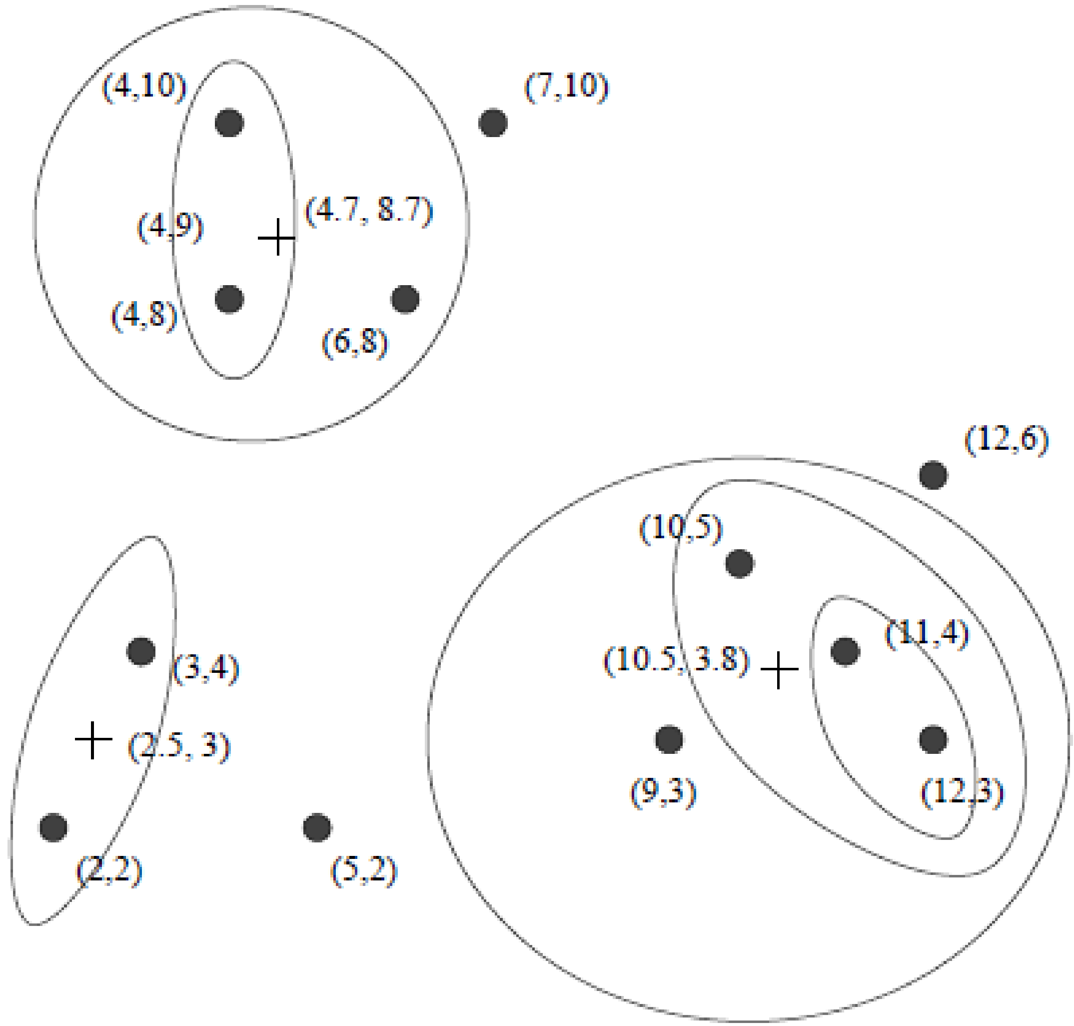
- Which two clusters to be merged next?



After two more merges

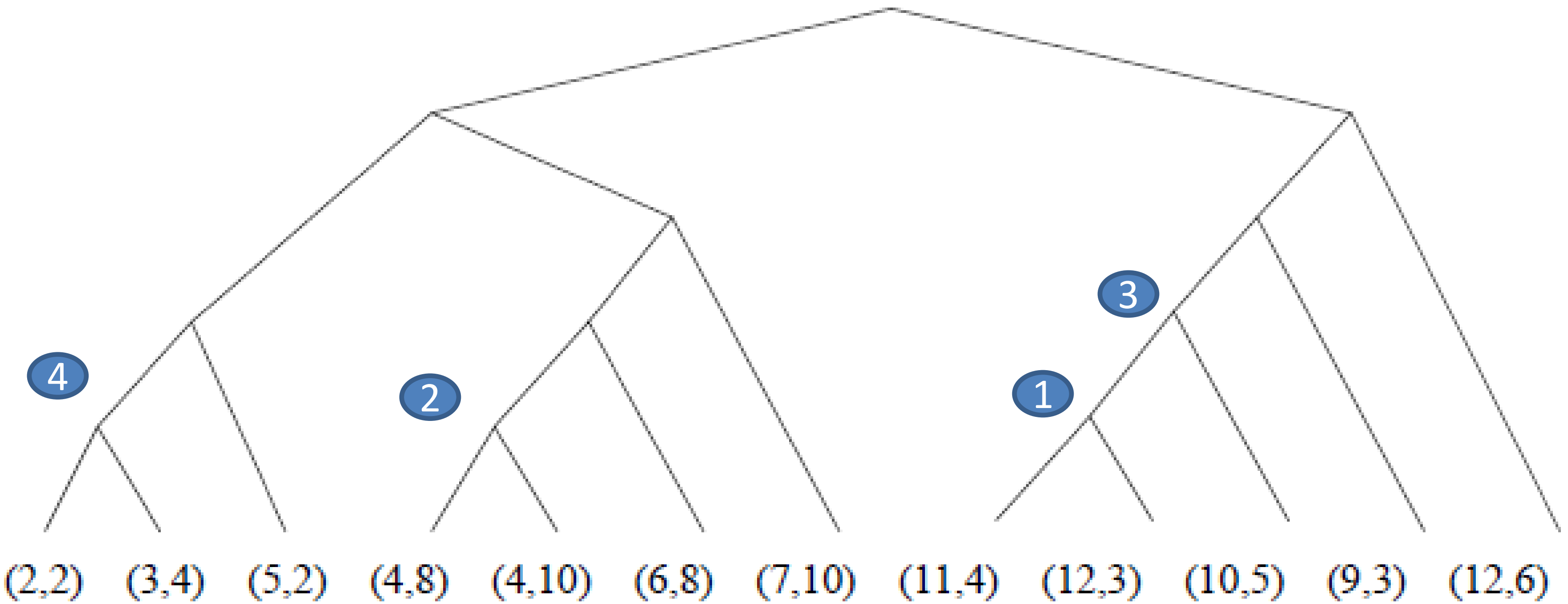


After 3 more merges



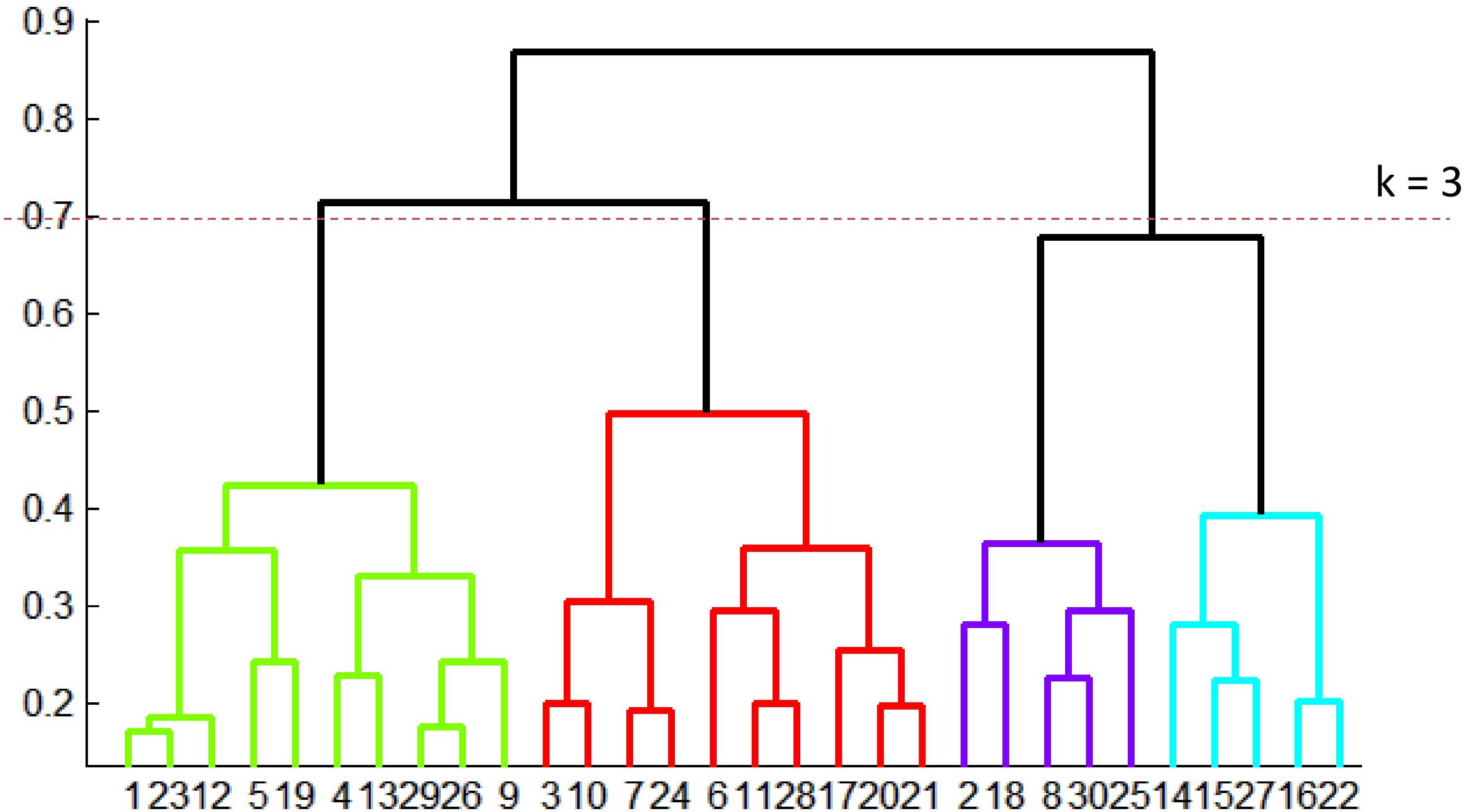
Final Result

- Represented as a tree



Dendrogram

- Can obtain k clusters from result for desired k
  - k can be any value between 1 and n



Complexity of Hierarchical Clustering

- n data points
- At most n – 1 step of merging

- Naive implementation, e.g., storing pairwise cluster distances in a matrix

	C1	**C2**	**C3**	**C4**
C1	0	2	3	2
C2		0	4	5
C3			0	3
C4				0

Triangular Distance Matrix

- Can save space by removing C_4 row and C_1 col.
  - Rows: $C_1, \ldots, C_{n-1}$
  - Columns: $C_2, \ldots, C_n$
- Can save even more space if stored as triangular array

	C1	**C2**	**C3**	**C4**
C1	0	2	3	2
C2		0	4	5
C3			0	3
C4				0

	C2	**C3**	**C4**
C1	2	3	2
C2		4	5
C3			3

Stored as Triples

- This may increase storage space
  - E.g., assume 4 bytes for distance value & index
  - Matrix: S_m=4(n-1)^2, triples: S_t=12*n(n-1)/2 = 6n(n-1)

  $\Rightarrow S_m < S_t$

  4 * n(n-1)/2 if stored as triangular array

- Note each cluster appears in n – 1 pairs

	C2	C3	C4
C1	2	3	2
C2		4	5
C3			3

n=4

⇨ $(C_1, C_2, 2), (C_1, C_3, 3), \ldots$

Updating Matrix After Merge

- Merge C_i and C_j
  - Delete row and column (if exist) for C_i
  - Delete row and column (if exist) for C_j
- Might delete fewer rows (if C_4 is merged) and fewer columns (if C_1 is merged)

	C2	**C3**	**C4**
C1	2	3	2
C2		4	5
C3			3

C5

	C3	**C4**
C1UC2	?	?
C3		3

Complexity of Naive Implementation

- Initially, $O(n^2)$ for creating matrix (n x n)

- Repeat: // merging two clusters at a time
  - Finding pair with minimum distance, say C_i and C_j (assuming currently k clusters): $O(k^2)$
  - Delete rows & columns for C_i and C_j: 2k-3 or O(k)
  - Add new row & column for new cluster C’: k-2 or O(k)
  - Compute dist. of C’ with other clusters: O(k)

=> Overall complexity: $O(n^3)$

Improved Version

- Use priority queue (e.g., heap-based) instead of matrix

1. Compute pairwise dist. of all points: $O(n^2)$
2. Build priority queue (time linear to queue size): $O(n^2)$
3. Repeatedly merge two closest clusters (among k clusters)
   a) Remove entries for old clusters: $(2k – 3) * O(\log(k))$
   b) Compute pairwise distances for new cluster: $k - 2$
   c) Add entries for new cluster: $(k-2) * O(\log(k))$

=> Overall complexity: $O(n^2 \log(n))$

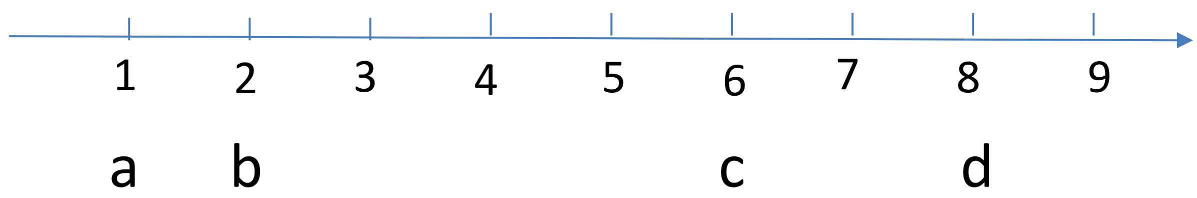
Additional data structure

- Maintain summary for each cluster C_i
  - Sum of point values: $sum(C_i)$
  - # of points: $cnt(C_i)$

- When C_i and C_j are merged into C_k:
  - $sum(C_k) = sum(C_i) + sum(C_j)$
  - $cnt(C_k) = cnt(C_i) + cnt(C_j)$

  => Centroid of $C_k = sum(C_k)/cnt(C_k)$

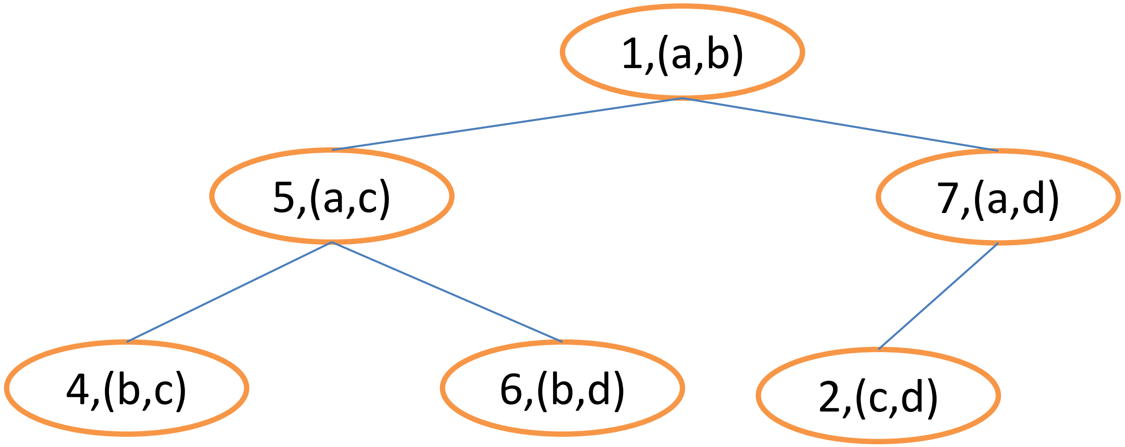
Example



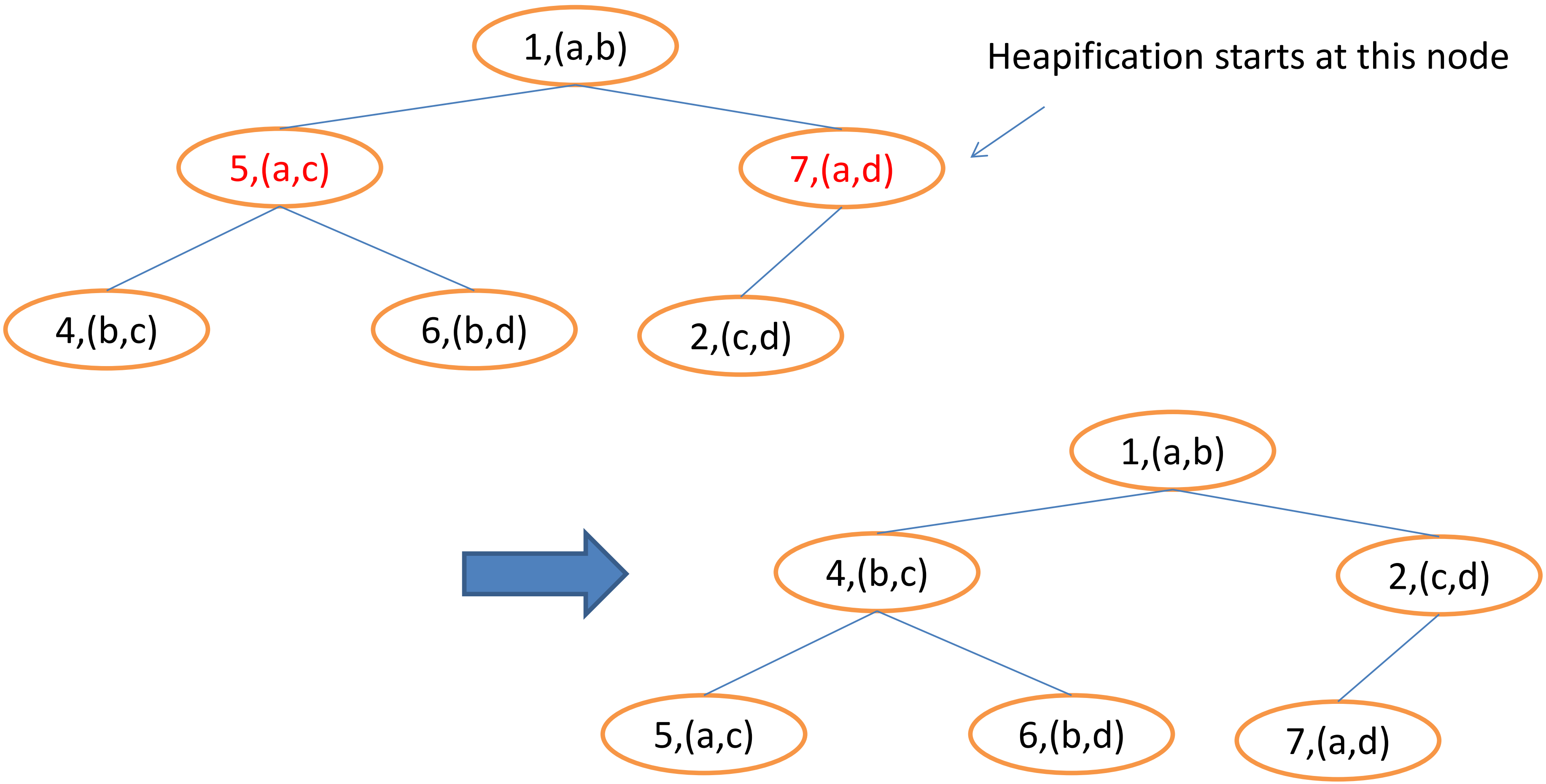
Pairwise distances:

- (a,b): 1
- (a,c): 5
- (a,d): 7
- (b,c): 4
- (b,d): 6
- …

Initial Heap Before Heapified

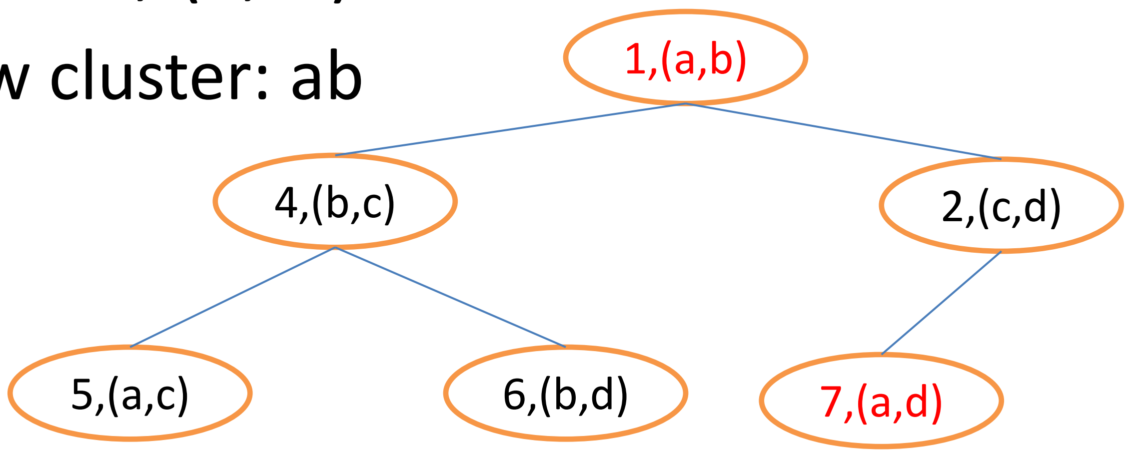


After Heapified

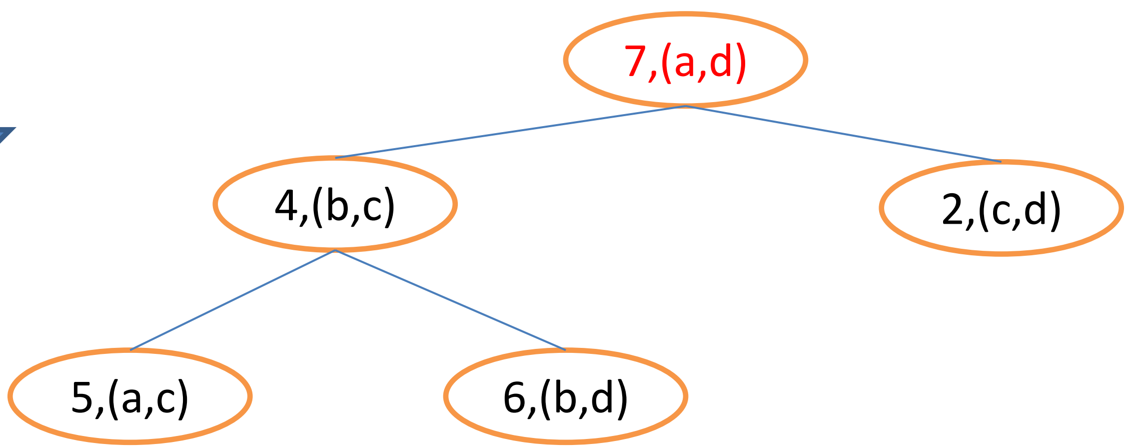


After one merge

- Extract: 1, (a, b)
  - New cluster: ab

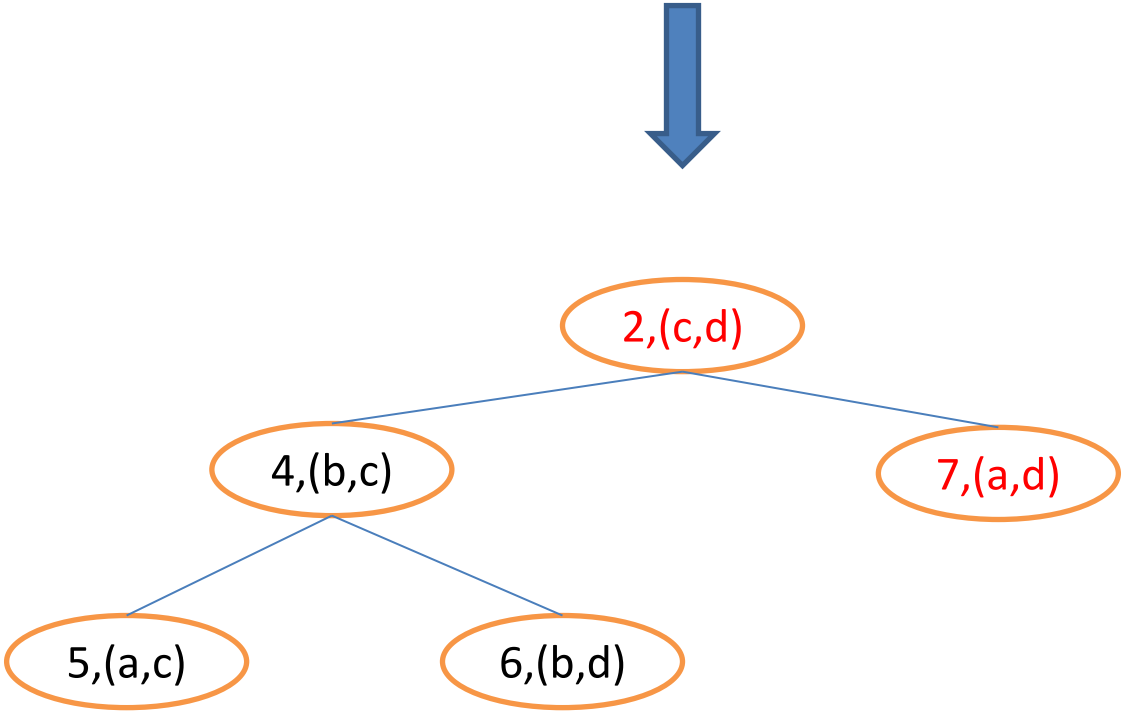


Replace root with last leaf



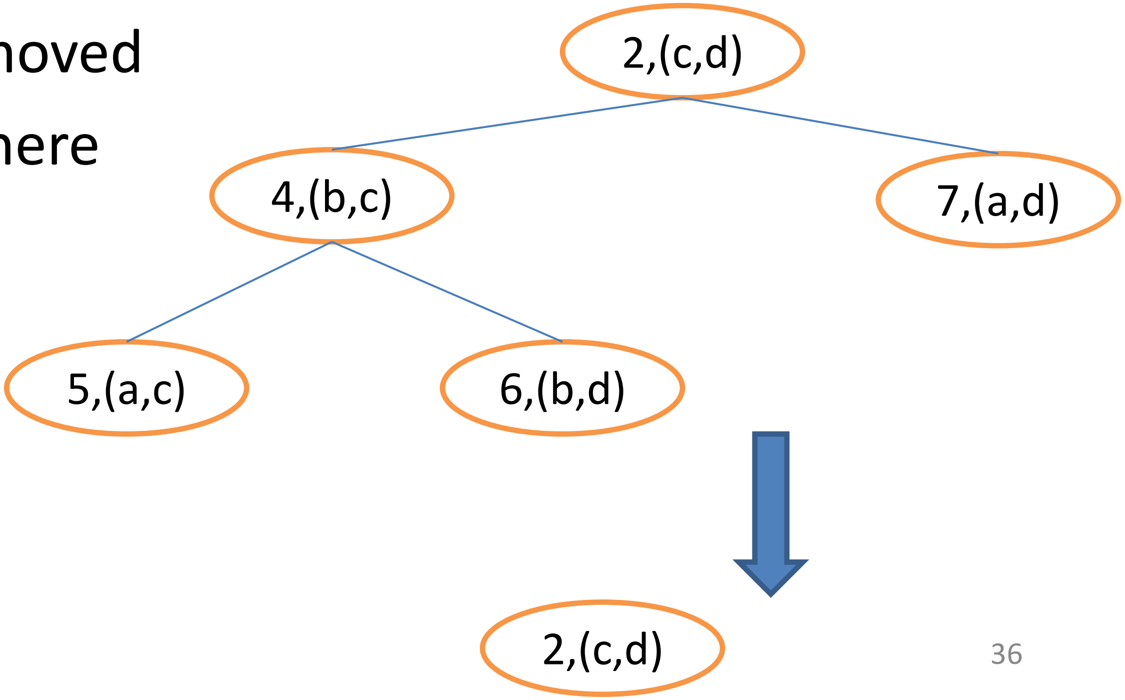
Fixing the Heap

- Sifting root down



Remove Nodes Having Old Clusters

- Need an index to know which nodes have a or b
  - # of clusters before merge = k (=4 here)
  - Need to remove 2k – 3 (= 5 here) nodes
    - (a, b) already removed
    - Remove 4 more here
  - And add 2 nodes

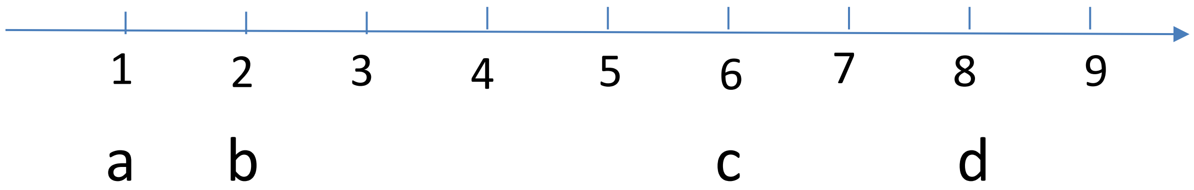


Compute Distance of ab with c & d

- Centroid for ab = (1+2)/2 = 1.5
  - sum({a}) + sum({b}) = 1 + 2 = 3
  - cnt{{a}) + cnt({b}) = 1 + 1 = 2

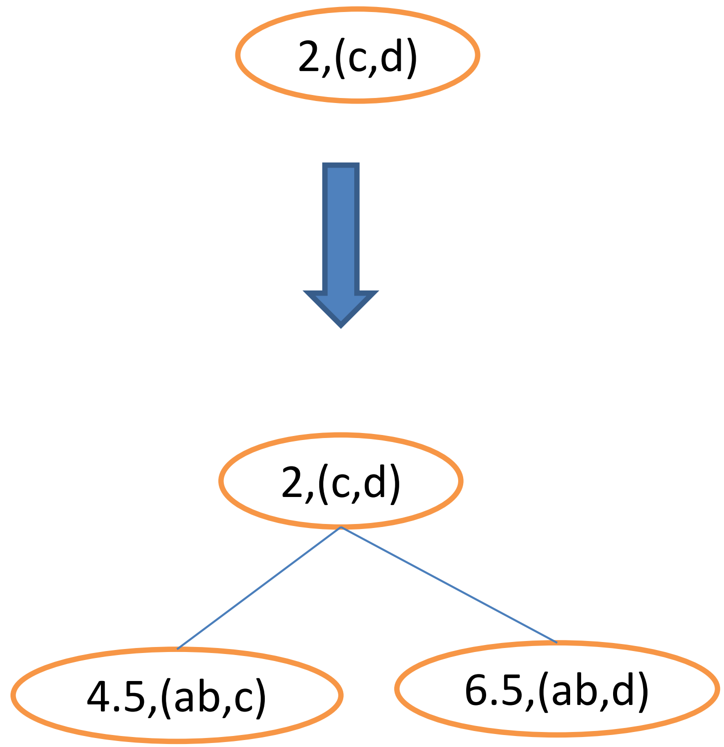
So

- dist(ab, c) = 4.5
- dist(ab, d) = 6.5



Adding new pairs to heap

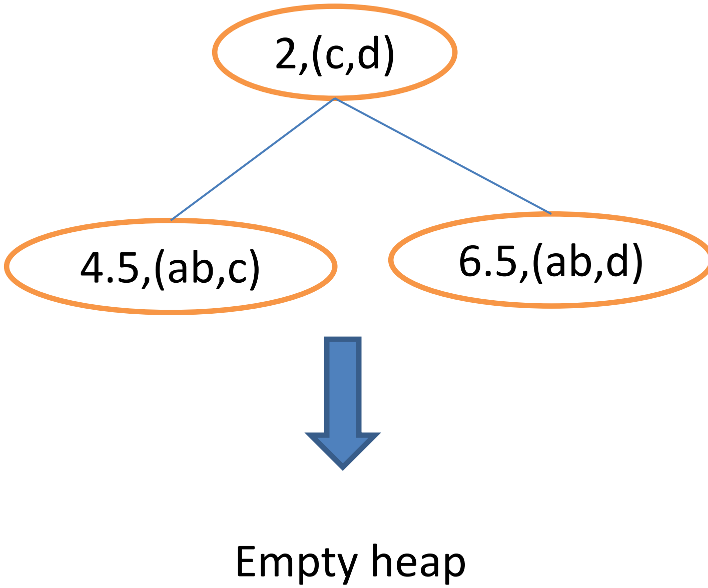
- Add two more pairs
  - k = 4 before merging
  - So adding k – 2 = 2 pairs



Merging c and d

- Extract 2, (c, d)
- Remove all nodes involving c or d
  - There are 2k - 3 = 3 such nodes, since k = 3
  - Need to remove two more
  - Since (c,d) already removed

$\Rightarrow$ Empty heap



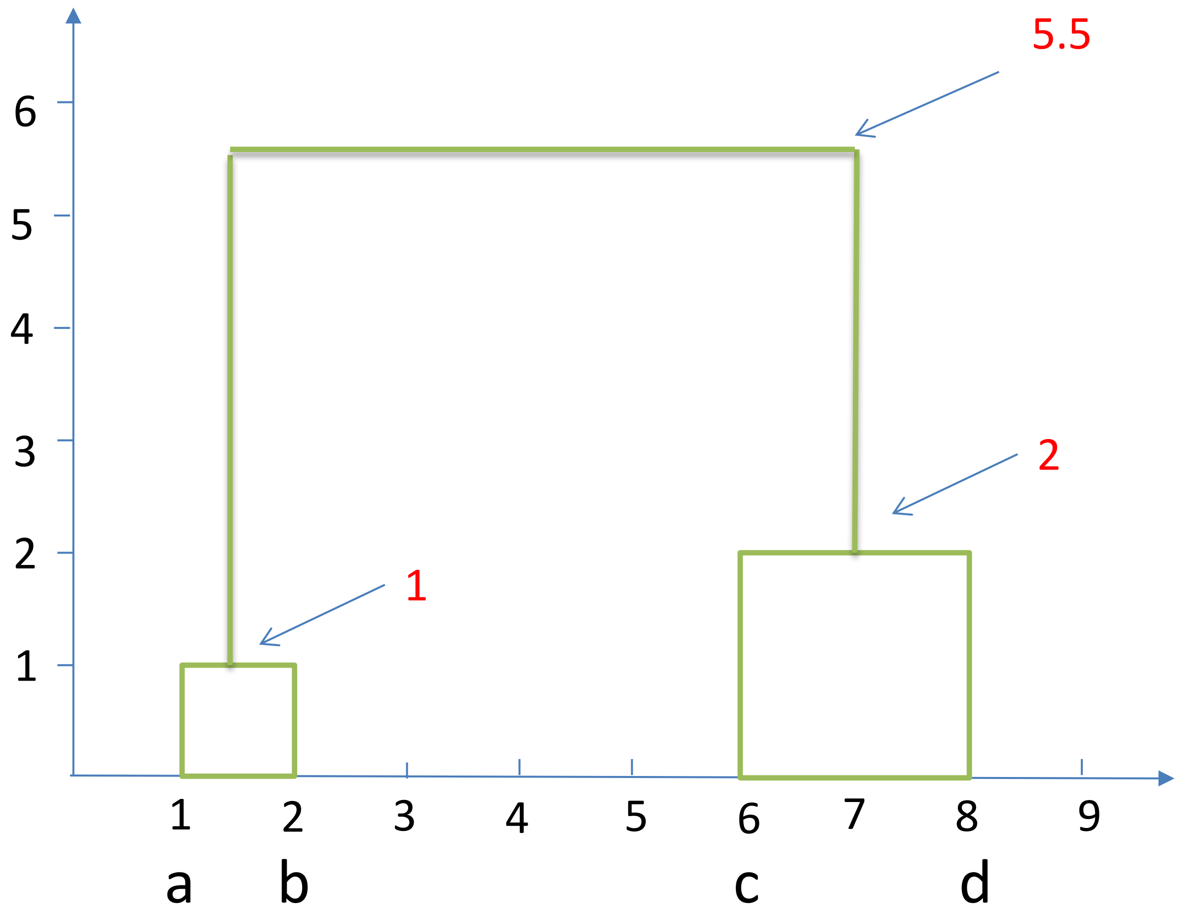
Adding new pairs to heap

- New cluster cd (centroid = 7)
  - Compute distance between cd and ab = 5.5
- Add distance between cd and ab to heap

5.5,(ab, cd)

- Final merge => (abcd)

Dendrogram



Notes

- Need to find entries in queue that need to be removed
  – Can not scan the queue: $O(n^2)$

- Can maintain an index (e.g., a hash table)
  – Given a cluster number, look up entries in queue
  – Need to update index when queue changes

- May also use lazy deletion strategy

Lazy deletion

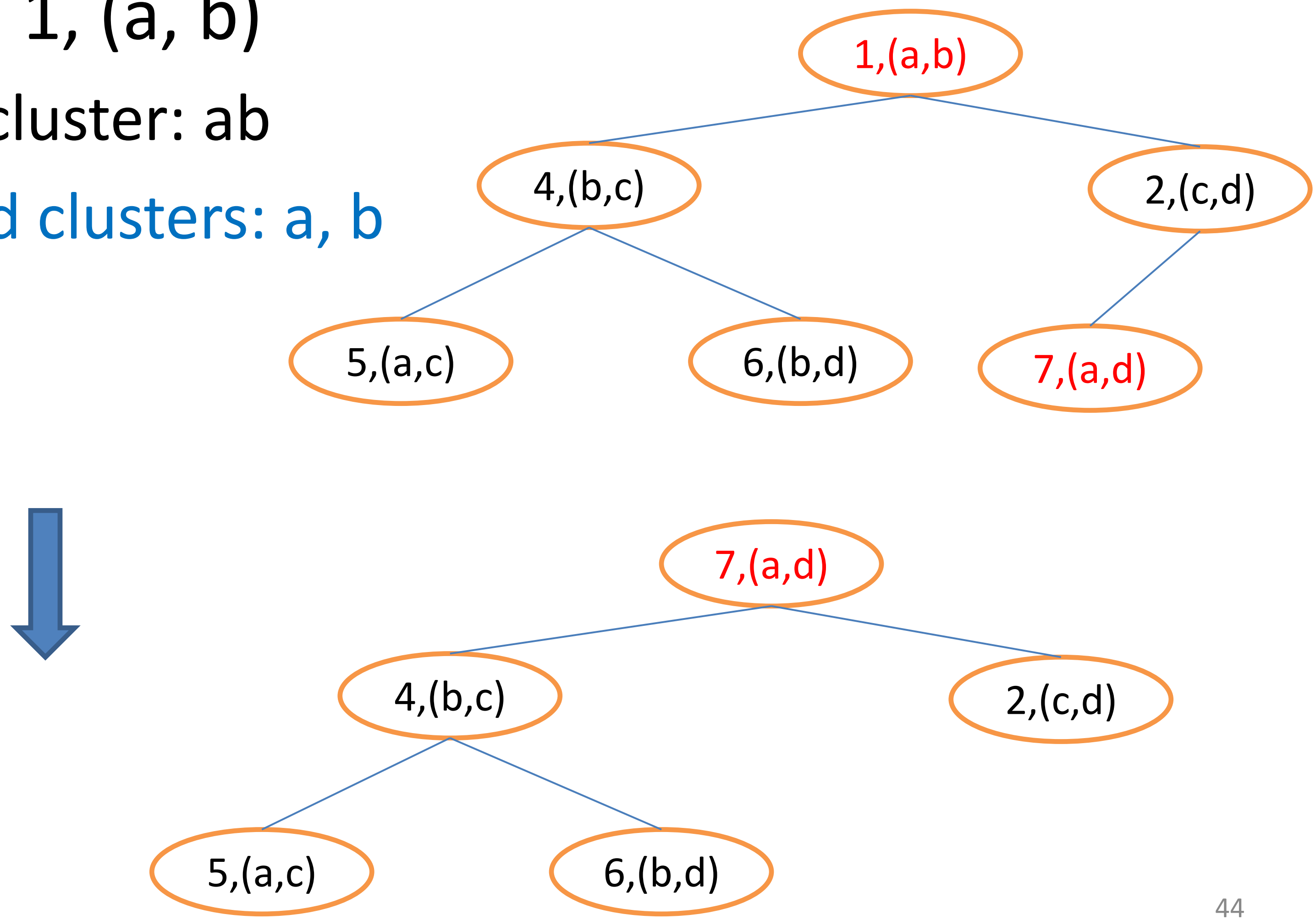
- Remove invalid entries only when they show up at root
  - Invalid: entries containing clusters which have been removed

- Does it affect the complexity of the algorithm?

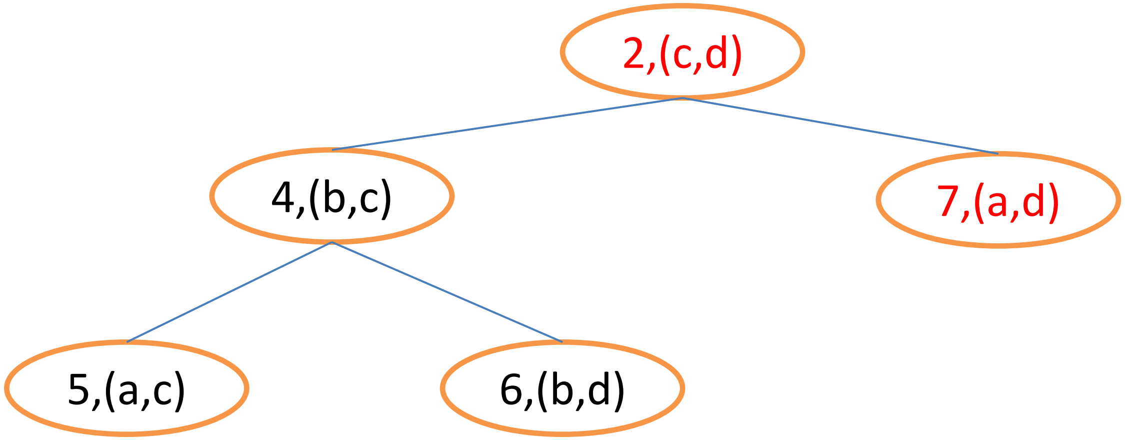
Current clusters: (ab) (c) (d)

After one merge

- Extract: 1, (a, b)
  - New cluster: ab
  - Invalid clusters: a, b

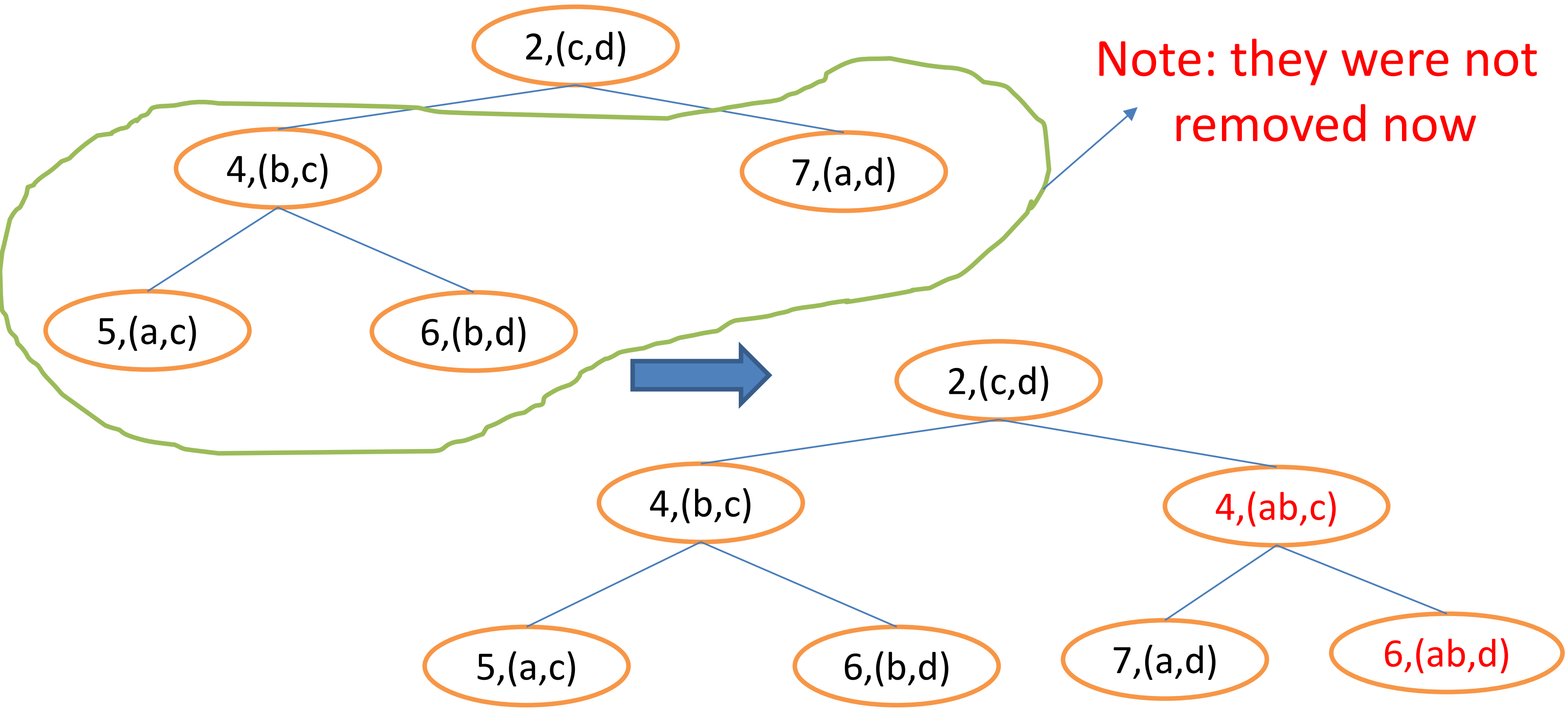


After sifting root down



Add Pairs for New Cluster ab

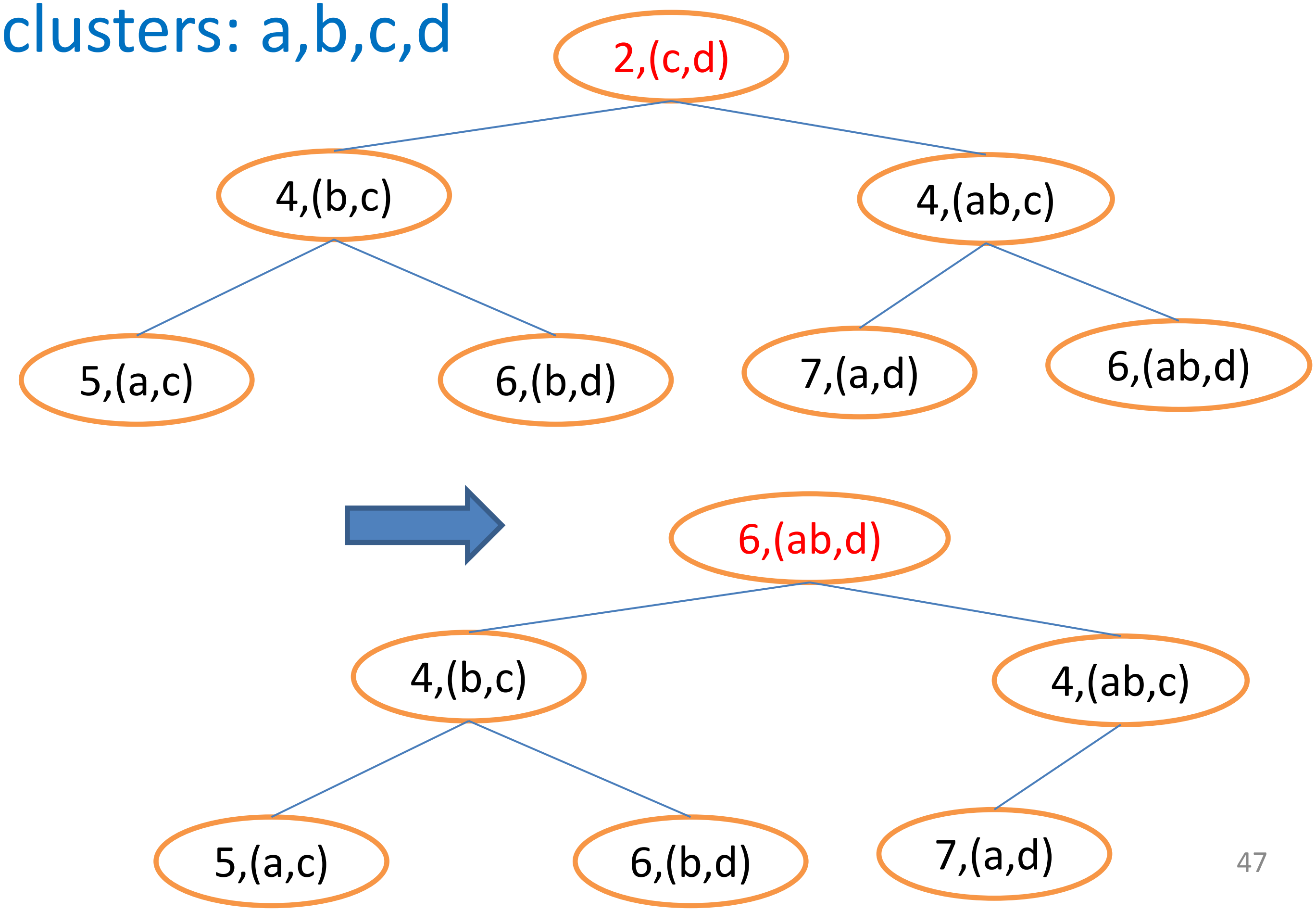
- 4, (ab, c) and 6, (ab, d): sifting up if needed



Current clusters: (ab) (cd)

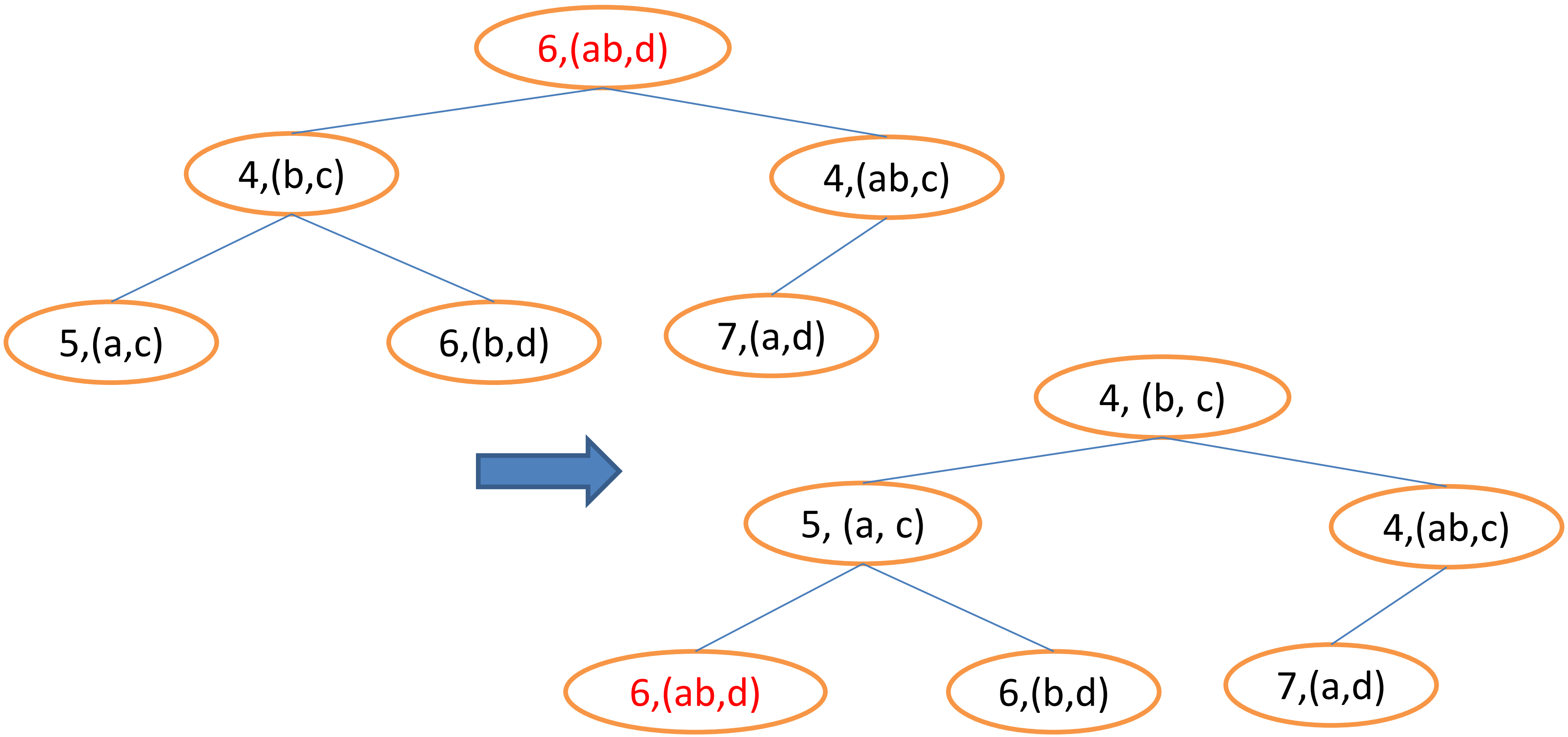
Second Merge

- Extract 2, (c, d), i.e., merge c and d
  - Move last leaf to root and sift down (next slide)
  - Invalid clusters: a,b,c,d



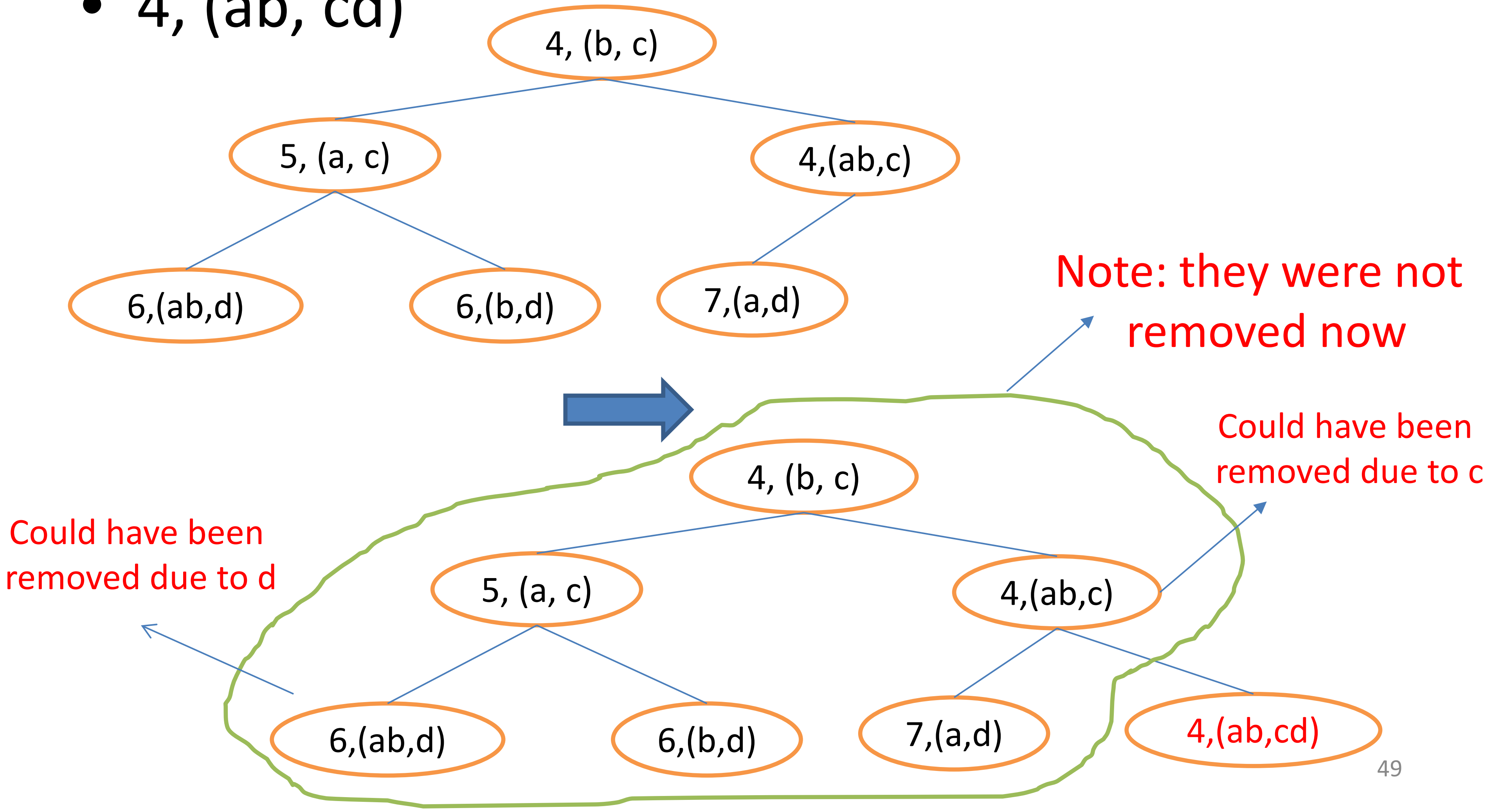
Sift down root

- Can go either way, say left



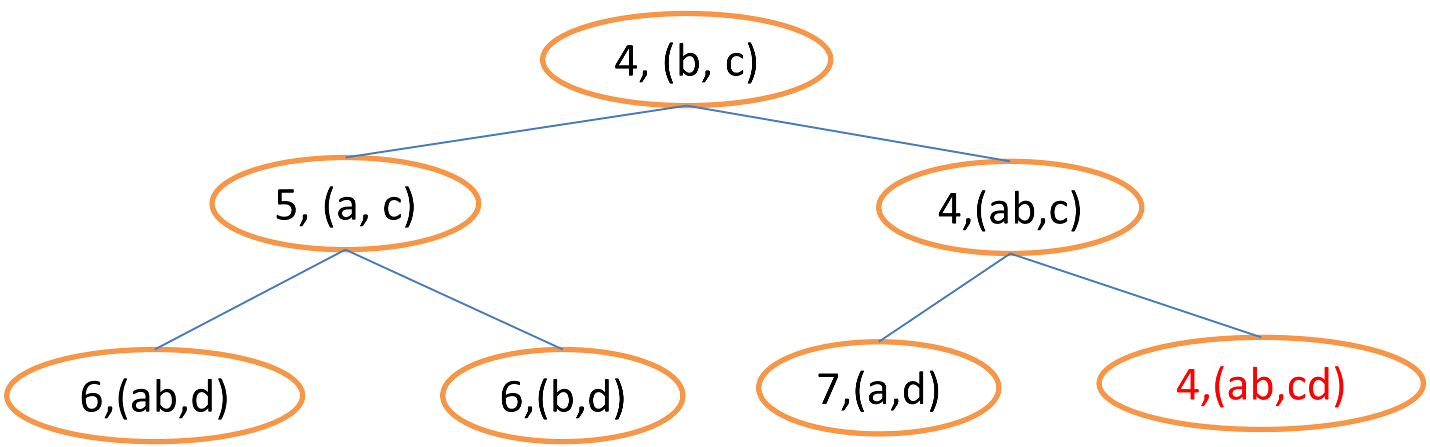
Add Pairs for New Cluster cd

- 4, (ab, cd)

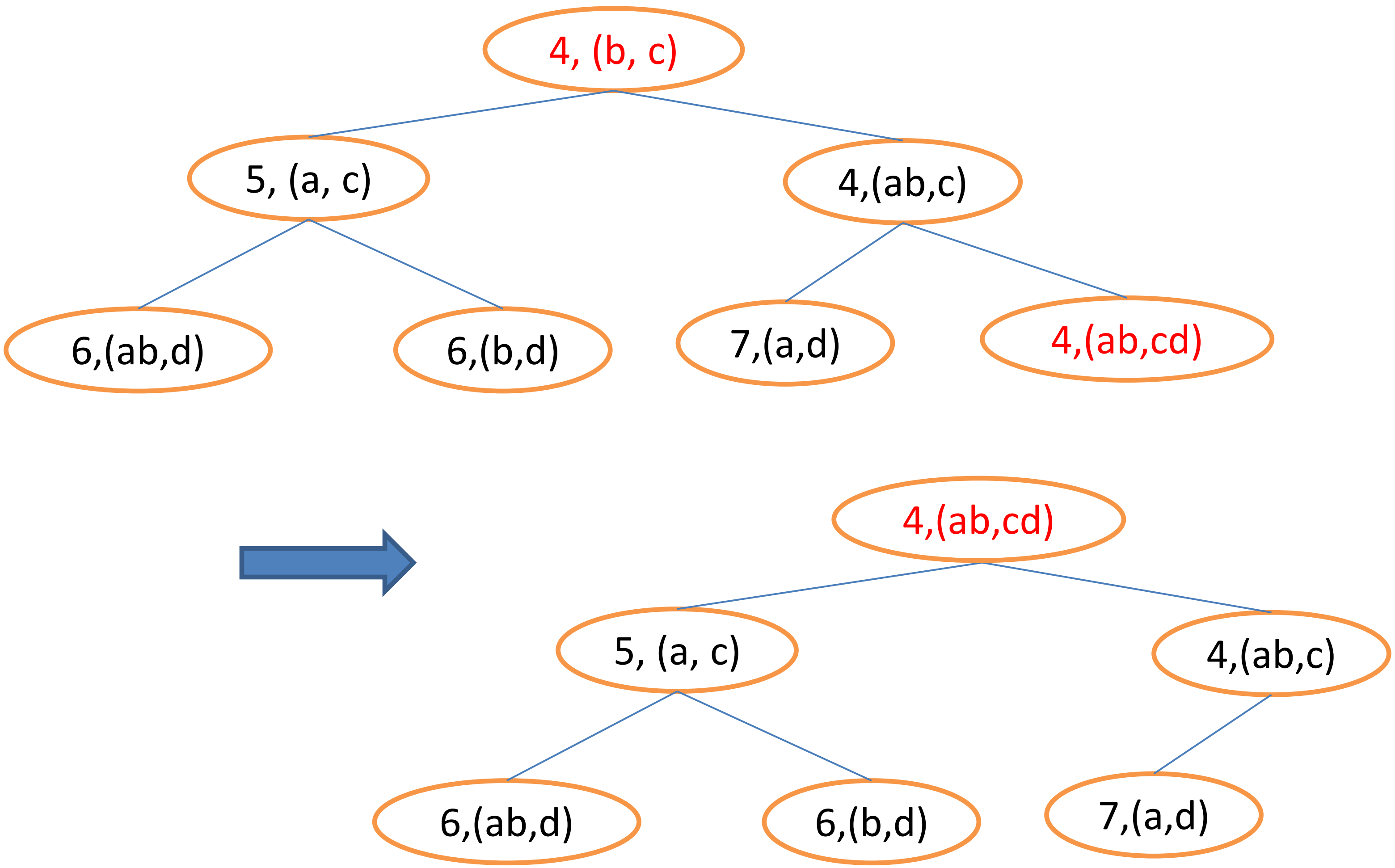


Find Next Two Clusters Merge

- Current invalid clusters: a, b, c, d
- Extract 4, (b, c)
  - found out it contains invalid cluster
  - ignore

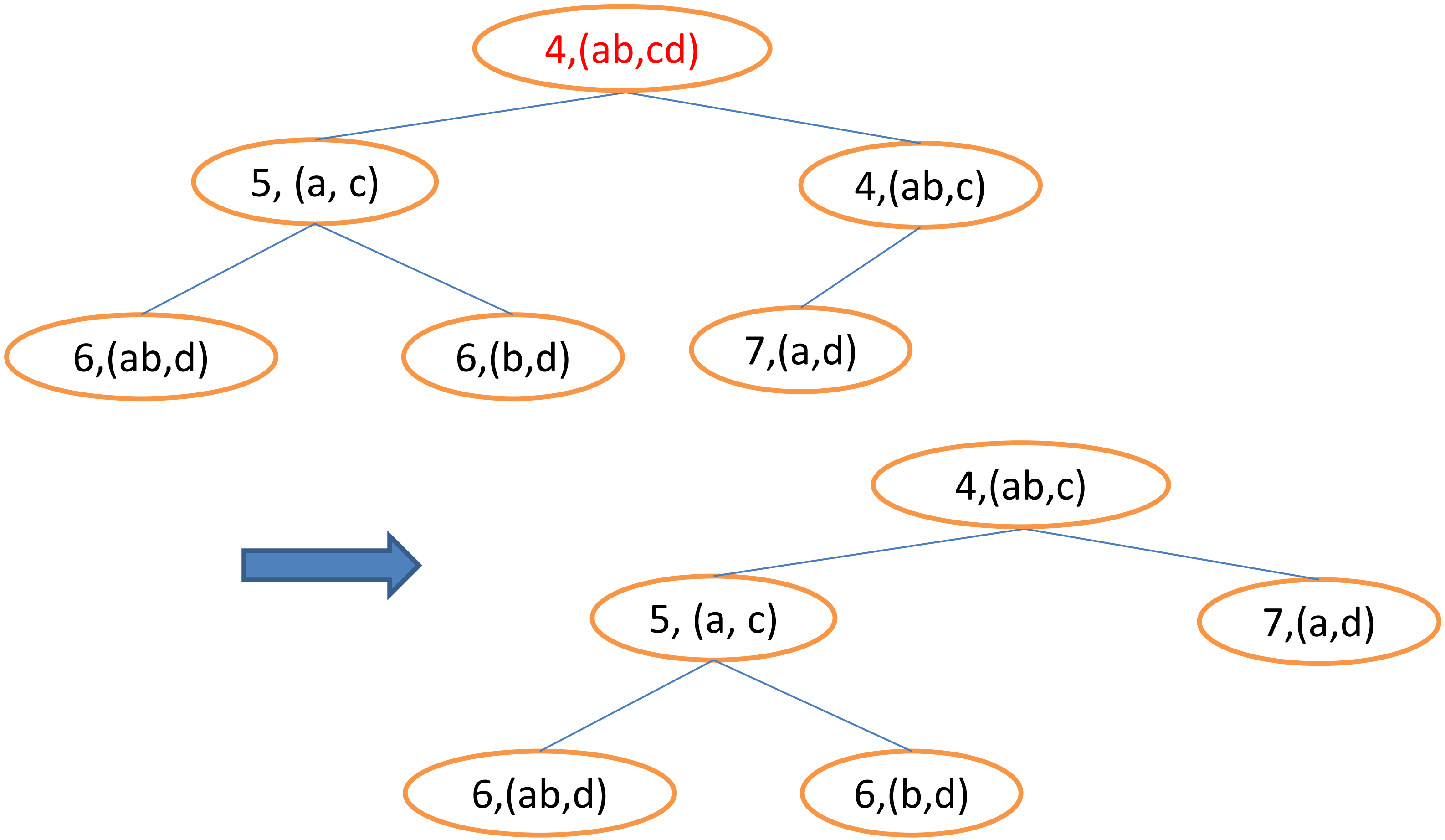


After Exacting 4, (b, c)



Found a Valid Pair

- Extract 4, (ab, cd)



Summary

Clusters	Pairs	Merge	K	Remove (2k-3)	Add (k-2)
a,b,c,d	(a,b) (a,c) (a,d) (b,c) (b,d) (c,d)	(a,b)	4	5 (red on left)	2 (ab, c) (ab, d)
ab,c,d	(c,d) (ab,c) (ab,d)	(c,d)	3	3	1 (ab,cd)
abcd					

Lazy deletion: initially $O(n^2)$ pairs
Each merge: add $O(k)$ pairs
Tree: has still $O(n^2)$ pairs

Other Measures of Cluster Distance

- Min/max of distances of any two points, one from each cluster
- Avg. distance of all pairs of points, one from each cluster
- Merge two clusters if resulting cluster has
  - lowest radius
  - lowest diameter

Cluster radius and diameter

- Radius
  - max distance from any point in the cluster to the centroid of the cluster
- Diameter
  - Max distance between any two points in the cluster

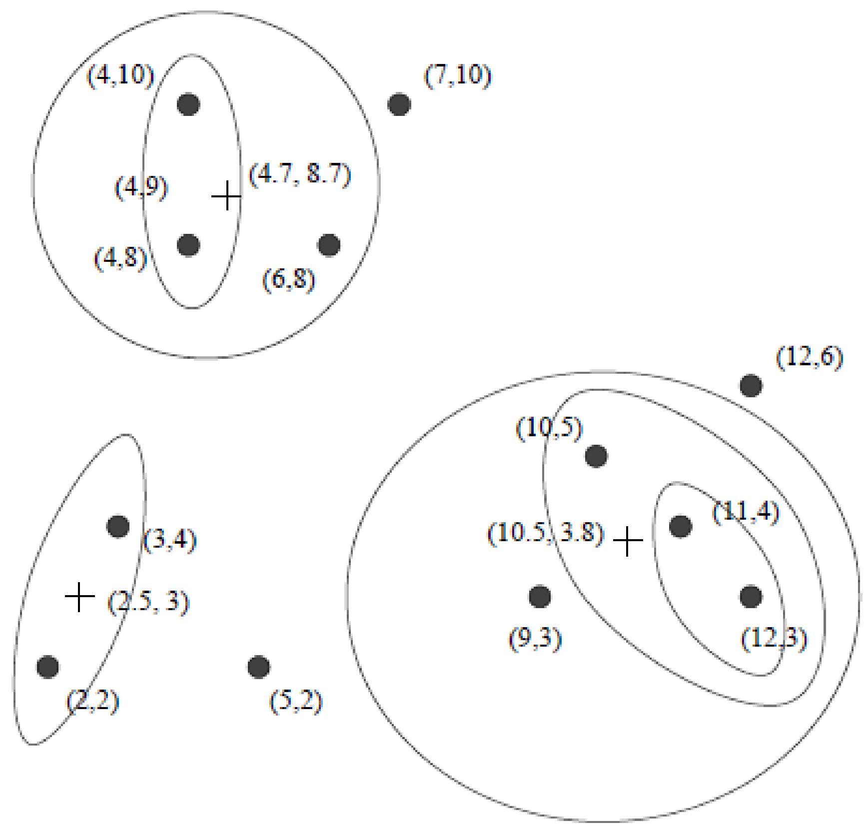
Stopping Rules

- Stop if diameter or radius of next cluster > threshold, e.g., 30% of that of entire data set

- Or stop if density of next cluster < threshold
  - Density: how many points per unit volume
  - Volume: estimated as some power, e.g., 1 or 2, of diameter or radius

Stopping Rules

- Or stop at a jump in avg. diameter of clusters
  - when will there be such a jump when merging the following clusters?



Non-Euclidean Space

- Distance of two points can NOT be measured by their locations (i.e., no concept of coordinates)
- May use Jaccard, Cosine, Edit, Hamming to compute distances as appropriate
- For example, Cosine can be used on two vectors
  - E.g., document vectors where values are word weights

Clusteroid

- Centroid is not meaningful in non-Euclidean
- Clusteroid = a point in the cluster that minimizes:
  - Sum of (squared) distances to other points, or
  - Maximum distance to another point

Example

- Consider a cluster of 4 points:
  - abcd, aecdb, abecb, ecdab

- Their edit distances:

	aecdb	abecb	ecdab
abcd	3	3	5
aecdb		2	2
abecb			4

Determine Clusteroid

- aecdb will be chosen as clusteroid
  - Located in “center” judged by all 3 measures

	aecdb	abecb	ecdab
abcd	3	3	5
aecdb		2	2
abecb			4

Point	Sum	Sum-sq	Max
abcd	11	43	5
aecdb	**7**	**17**	**3**
abecb	9	29	4
ecdab	11	45	5

Roadmap

- Problem, types, and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR
  - CURE
- Curse of dimensionality

K-means Algorithm

1. Pick k points as initial centroids of k clusters

2. Repeat until centroids stabilize

   Point assignment

   a) For each point p,

      Find the centroid to which p is closest

      Add p to the cluster of that centroid

   b) Re-compute centroids of clusters

Note

- Textbook version (Figure 7.7):
  - Adjust centroid immediately for every point assignment
- Our algorithm:
  - Adjust centroids after all point assignments
  - Use this approach in all homework, quiz, exam

Complexity

Assume n data points

1. Pick k points as centroids of k clusters
   – O(k) (but see later discussions)

I: # of iterations

2. Repeat until centroids stabilize O(I*n*k)
   a) For each point p, O(n*k)
      Find the centroid to which p is closest
      Add p to the cluster of that centroid
   b) Re-compute centroids of clusters: O(k*n/k) or O(n)

Objective function

- Minimize sum of squared errors

Center of cluster k

$$E = \sum_{k=1}^{K} \sum_{p \in C_k} (p - \mu_k)^2$$

Sum over all points in cluster C_k

- What happens to E…
  – When points are reassigned?
  – When centers are recomputed?

EM interpretation

- Model (M): centers of k clusters
- Training data (T): points belong to which cluster(s)
- But we do not know either

EM interpretation

- So we start with initial guess of M

Using M to estimate T

- Expectation: compute probability of points belonging to each cluster
  – K-means does hard assignment

Using T to estimate M

These are probabilistic training data (T)

- Maximization: estimate M based on the point assignments using MLE
  – K-means computes the average as estimate

Important Issues in k-means

- Picking points for initial centroids
  - May produce different clusters with diff. choices

- Picking right values for k: # of clusters

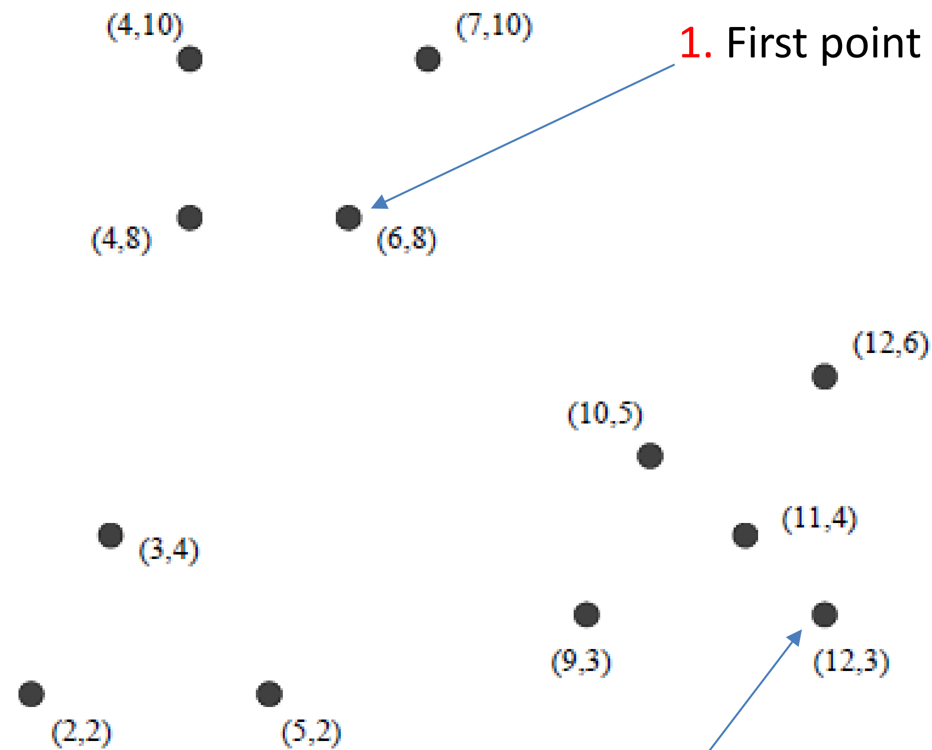
Picking Points for Initial Centroids

- Method 1: Pick points as far away as possible
  - First, pick one randomly
  - Next, repeatedly pick a point which is far away from ones already chosen
- Method 2: produce k clusters by hierarchical methods, and select one point from each cluster
  - May cluster on a sample of points instead

Method 1 example

- First point: (6, 8)
- 2nd point: (12, 3)
- 3rd point: (2, 2)
- …

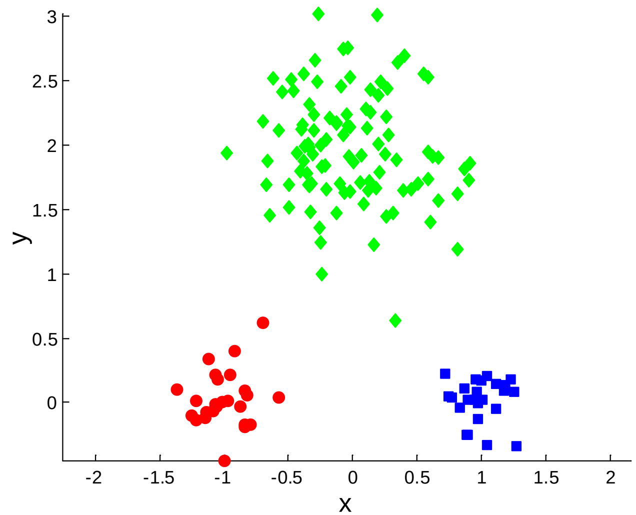
1. First point



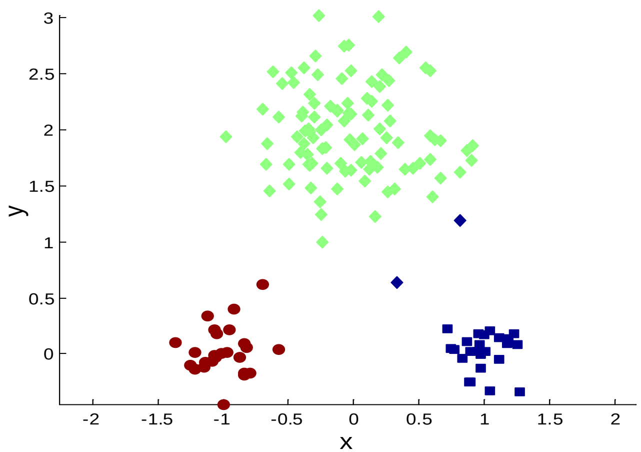
3. (2, 2) is farthest away from both (6,8) and (12,3)

2. (12,3) is farthest away from (6,8)

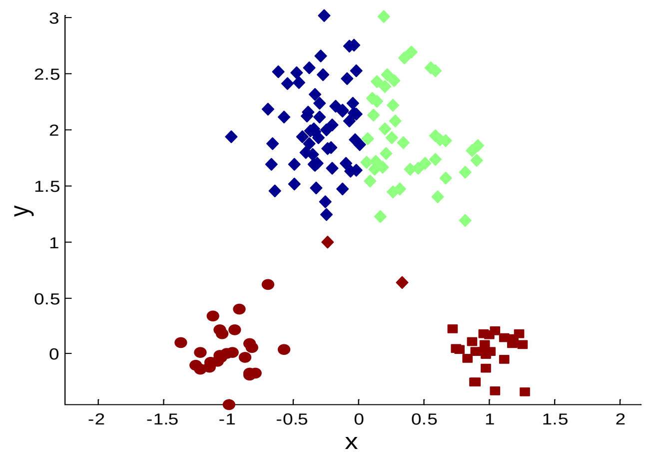
Two different K-means Clusterings



Original Points

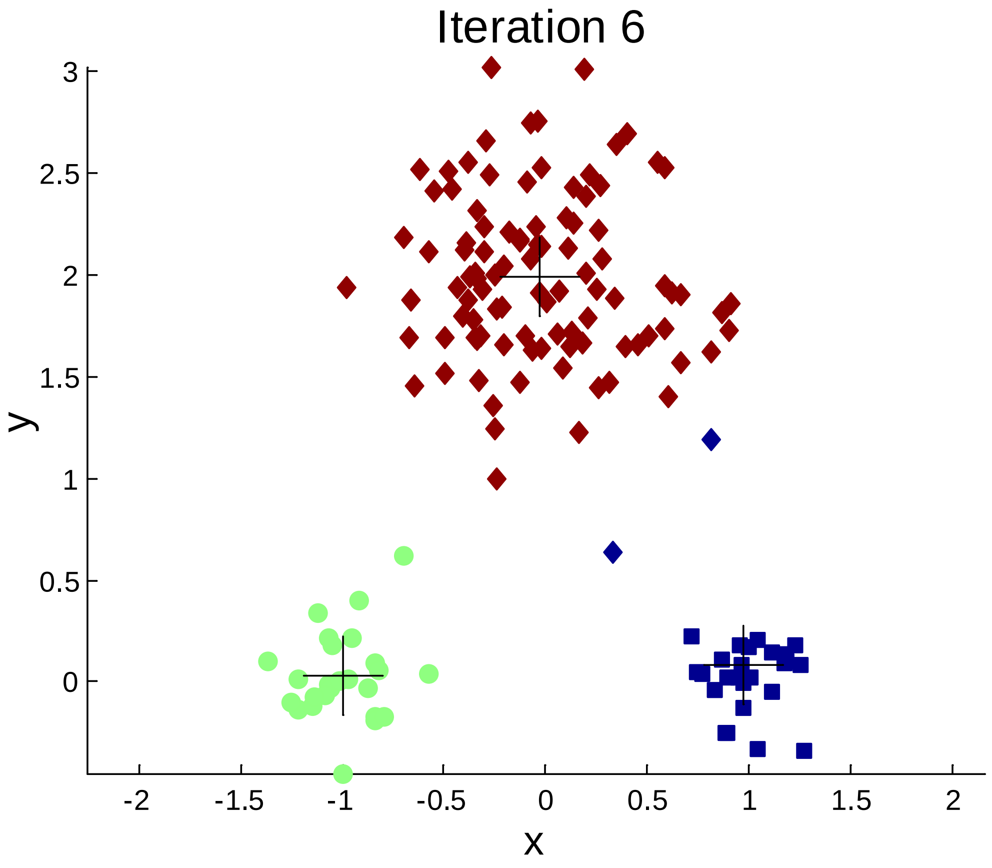


Optimal Clustering

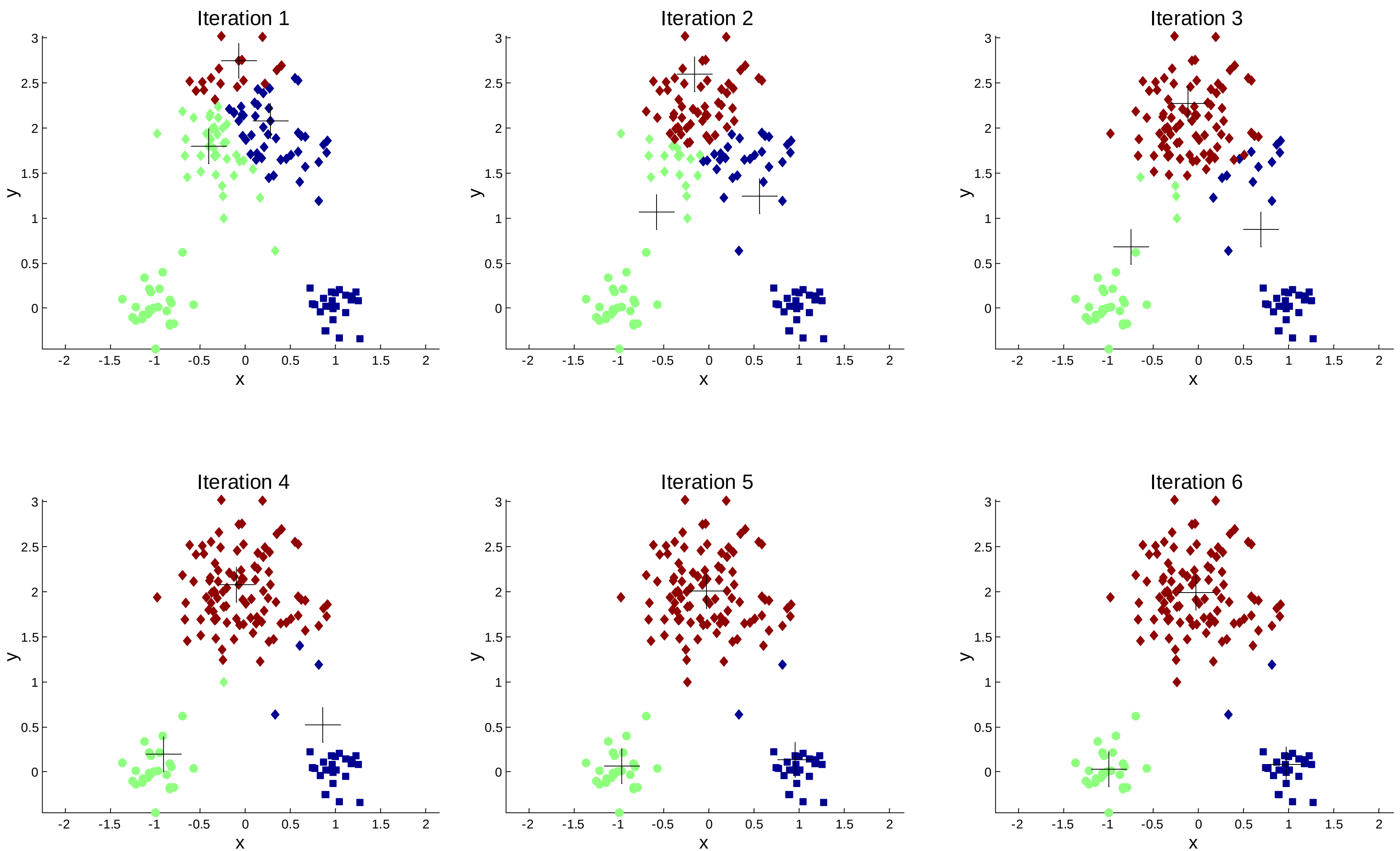


Sub-optimal Clustering

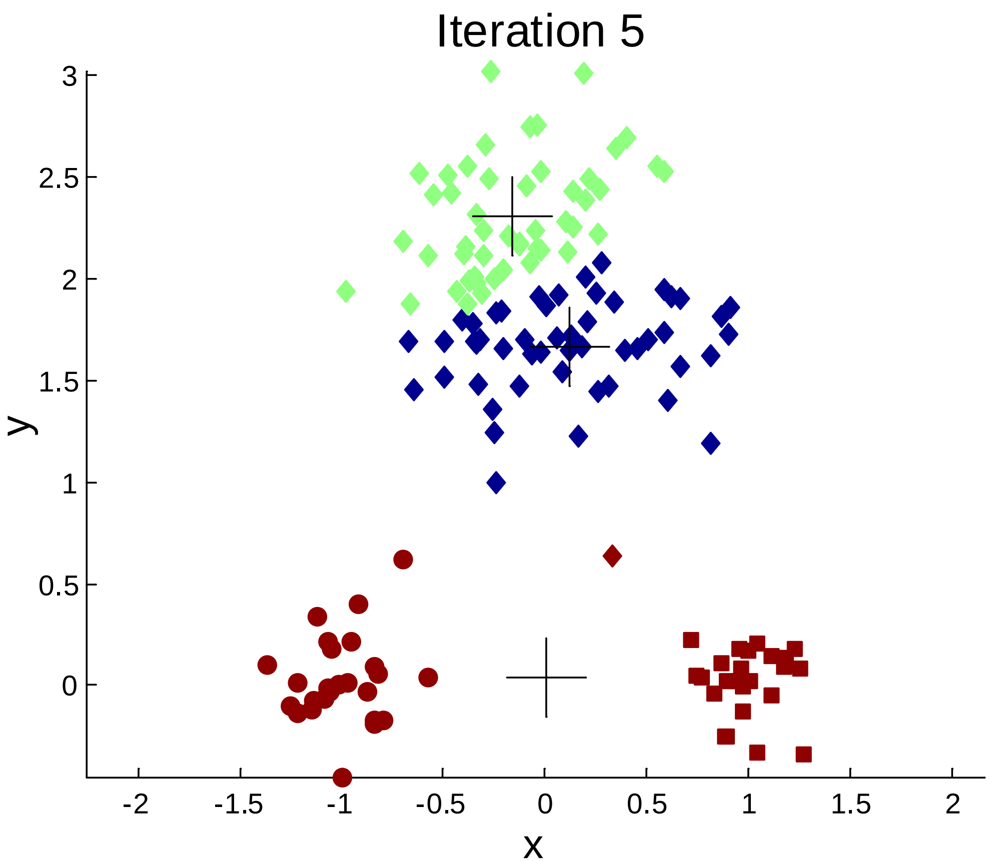
Importance of Choosing Initial Centroids



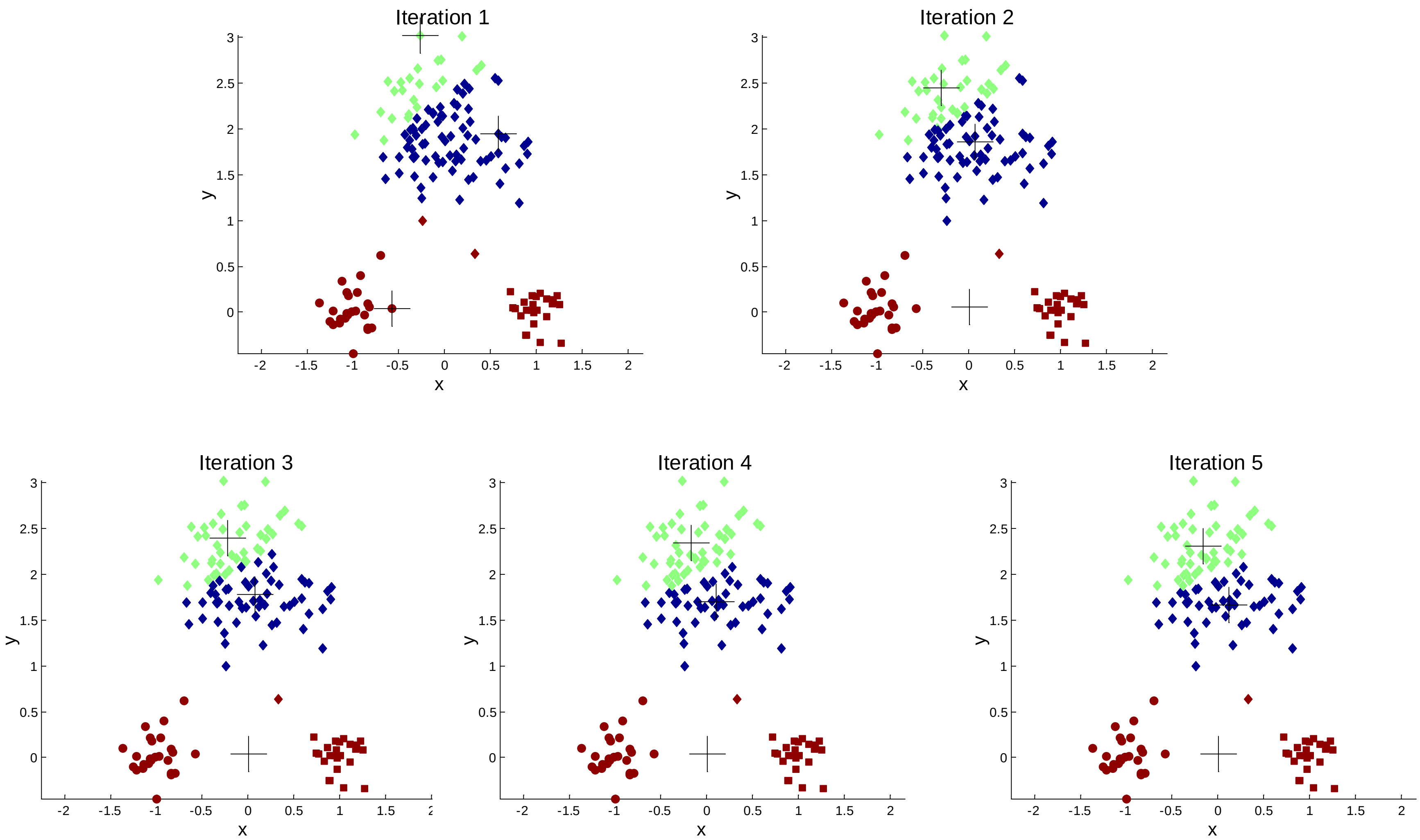
Importance of Choosing Initial Centroids



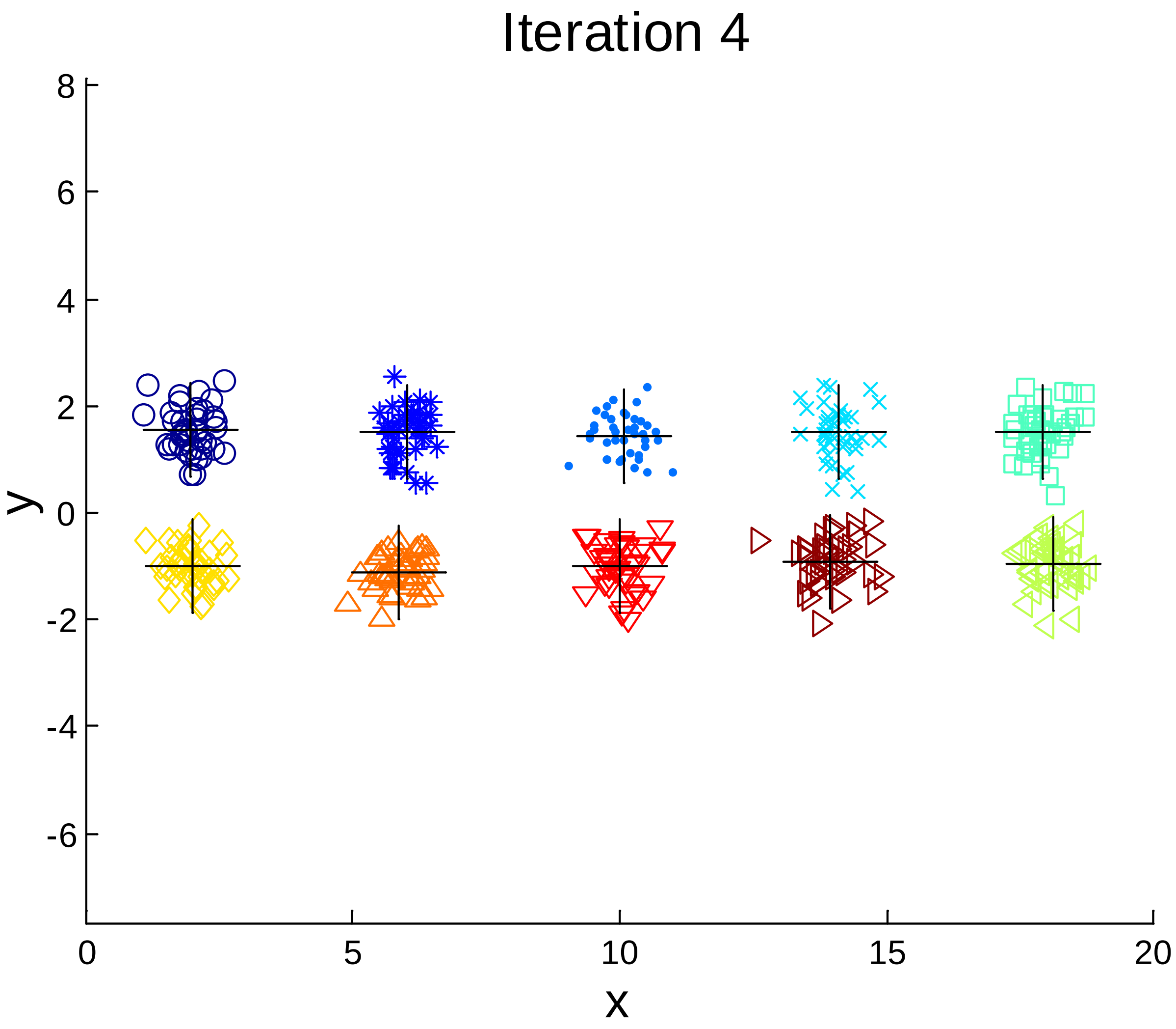
Importance of Choosing Initial Centroids …



Importance of Choosing Initial Centroids …

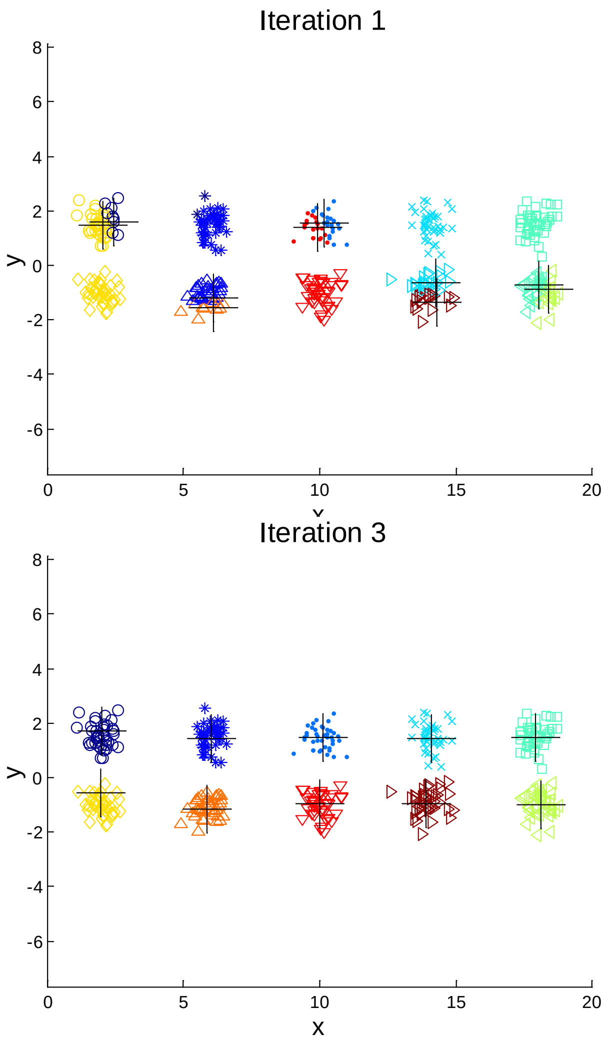


10 Clusters Example



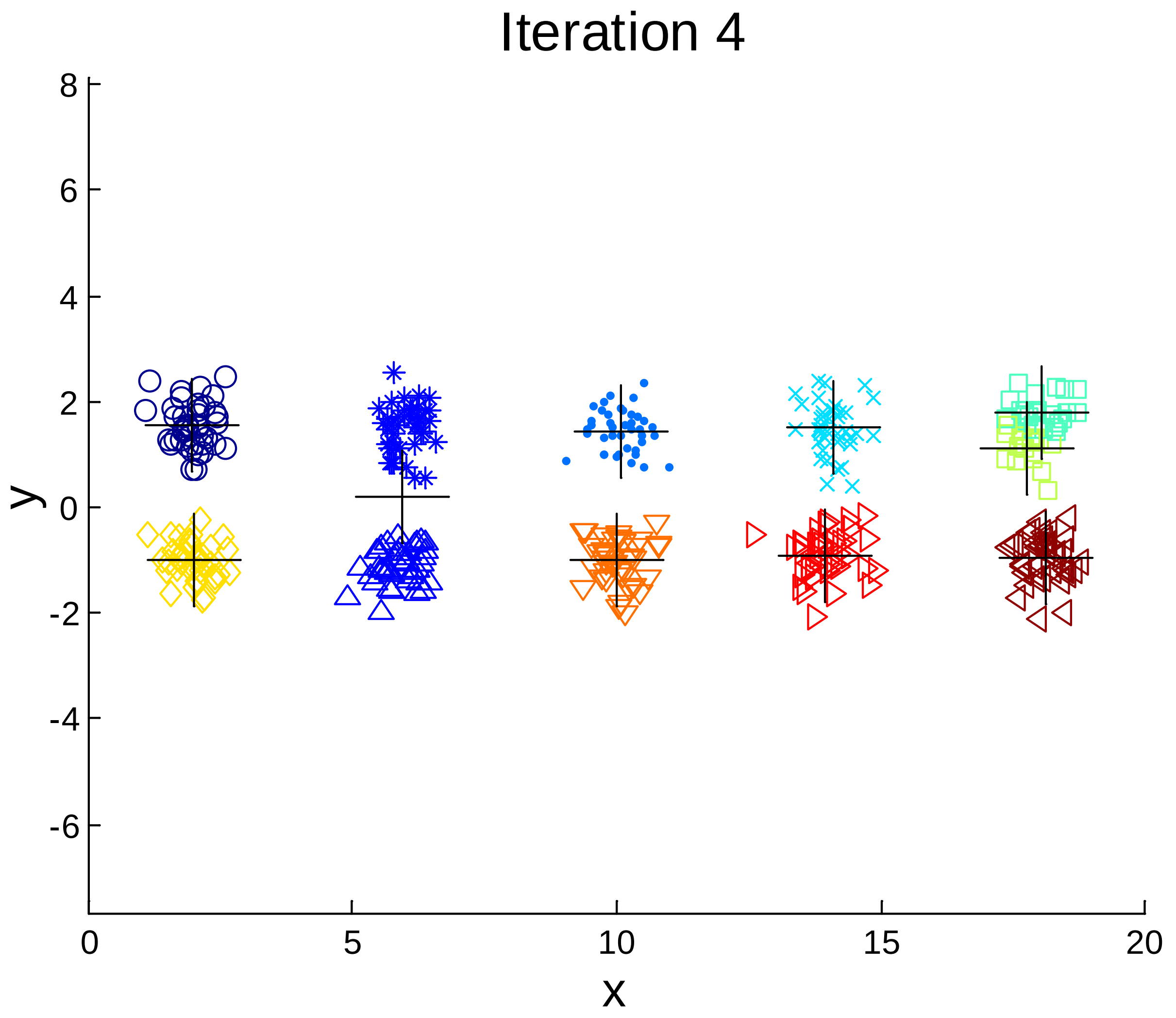
Starting with two initial centroids in one cluster of each pair of clusters

10 Clusters Example



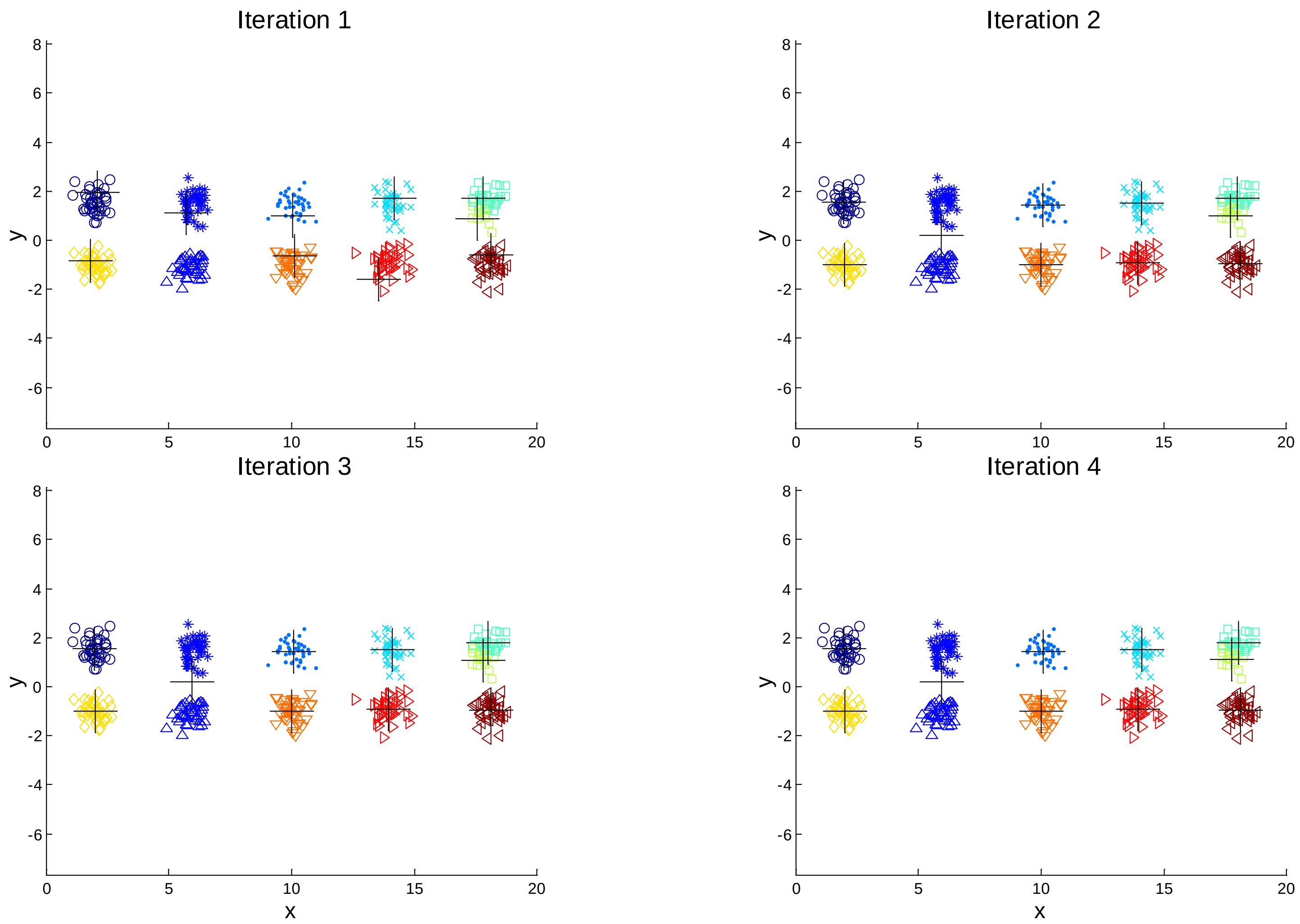
Starting with two initial centroids in one cluster of each pair of clusters

10 Clusters Example



Starting with some pairs of clusters having three initial centroids, while other have only one.

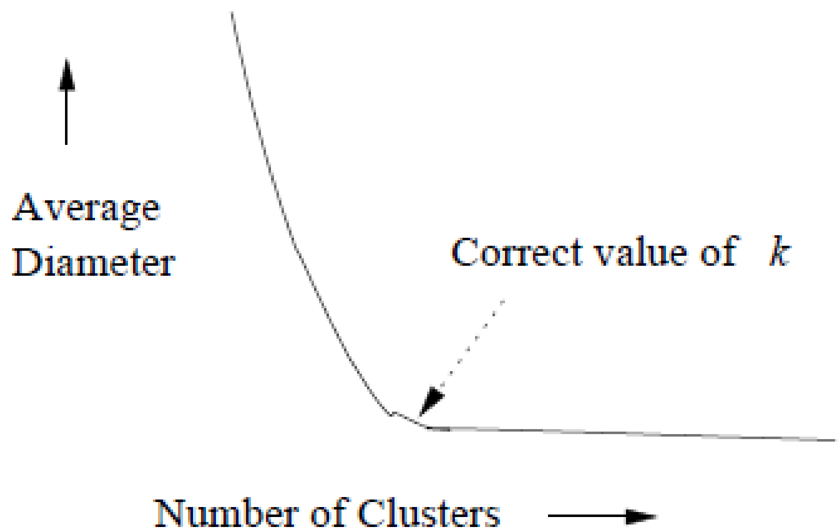
10 Clusters Example



Starting with some pairs of clusters having three initial centroids, while other have only one.

Picking Right Value for k

- When # of clusters x is smaller than some k
  - Cohesion of clusters increases dramatically when x increases
  - Cohesion = avg. diameter of clusters



Finding Right k

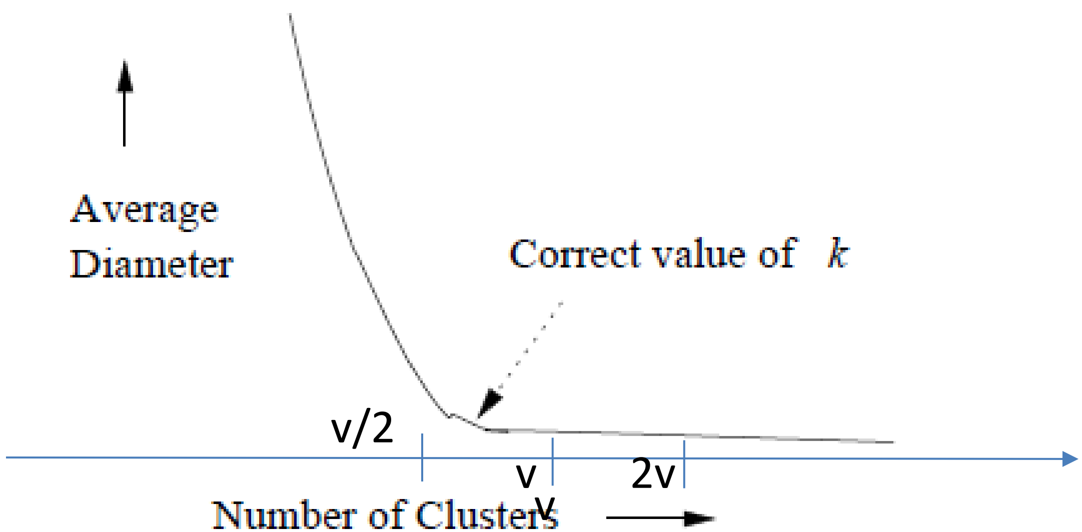
First, find the elbow of the curve:

- Run k-means for $k = 1, 2, 4, 8, \ldots 2^{m-1}$
  - i.e., double # of clusters at each clustering/run
- Stop at $k = 2v$
  - where cohesion changes little from $k = v$
  - Elbow: $[v/2, v]$

Since $k = 2^{m-1} = 2v$

$\Rightarrow m = 2 + \log_2 v$

(m: # of clusterings)



Define "little change"

- When k increases from v to 2v, the rate of decrease in cohesion is given by:
  - $r = |c(2v) - c(v)| / (c(v) * v)$
  - Which is the rate of relative decrease in cohesion normalized by the decrease in the number of clusters (i.e., v)

- Little change if r < some threshold, e.g., 10%

Finding Right k (k*)

- Next, binary search on [v/2, v]
  - Suppose current range [x, y]
  - Midpoint z = $\lfloor (x + y)/2 \rfloor$
  - If there is little change of cohesion between [z, y]

    k* in [x, z]

    else

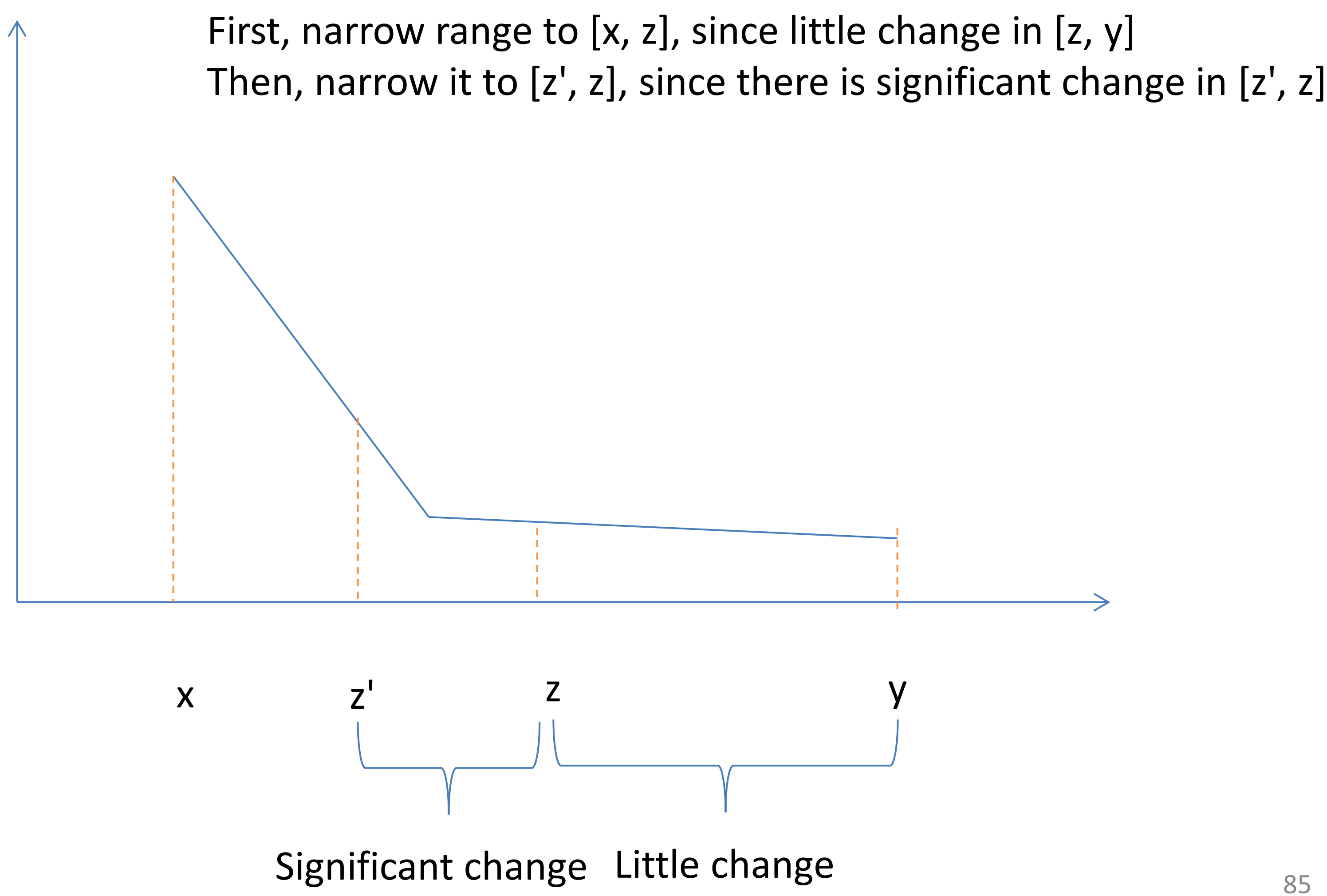
    k* in [z, y]
  - Continue search in [x, z] or [z, y]

$\Rightarrow$ Every search/clustering divides range by half

$\Rightarrow$ # of clustering = $\log_2 (v/2) = \log_2 v - 1$

Overall, need to perform about $2\log_2 v$ clusterings

Example



First, narrow range to [x, z], since little change in [z, y]
Then, narrow it to [z', z], since there is significant change in [z', z]
x
z'
z
y
Significant change
Little change

k-means in Spark

- examples/src/main/python/kmeans.py
  - Simple implementation
  - bin/spark-submit kmeans.py kmeans_data.txt 2 .1
- examples/src/main/python/mllib/k_means_example.py
  - Sample code using k-means algorithm implemented in MLlib

kmeans.py

```
sc = SparkContext(appName="PythonKMeans")
lines = sc.textFile(sys.argv[1])
data = lines.map(parseVector).cache()
K = int(sys.argv[2])
convergeDist = float(sys.argv[3])

kPoints = data.takeSample(False, K, 1)
tempDist = 1.0

while tempDist > convergeDist:
    closest = data.map(
        lambda p: (closestPoint(p, kPoints), (p, 1)))
    pointStats = closest.reduceByKey(
        lambda p1_c1, p2_c2: (p1_c1[0] + p2_c2[0], p1_c1[1] + p2_c2[1]))
    newPoints = pointStats.map(
        lambda st: (st[0], st[1][0] / st[1][1])).collect()

    tempDist = sum(np.sum((kPoints[iK] - p) ** 2) for (iK, p) in newPoints)

    for (iK, p) in newPoints:
        kPoints[iK] = p

print("Final centers: " + str(kPoints))

sc.stop()
```

Persist data points in memory

Initial centers

New centers

Sum of distances between new and old centers

Parse input & find closest center

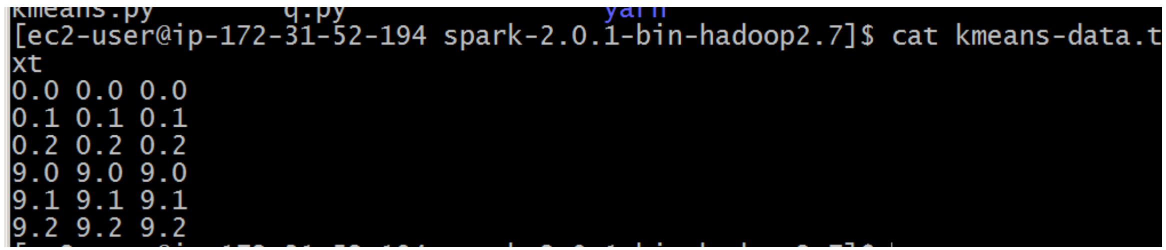
```
def parseVector(line):
    return np.array([float(x) for x in line.split(' ')])


def closestPoint(p, centers):
    bestIndex = 0
    closest = float("+inf")
    for i in range(len(centers)):
        tempDist = np.sum((p - centers[i]) ** 2)
        if tempDist < closest:
            closest = tempDist
            bestIndex = i
    return bestIndex
```

- Assign p to the closest cluster
- Return cluster index (index of centers)

Kmeans_data.txt

- A text file contains the following lines
  - 0.0 0.0 0.0
  - 0.1 0.1 0.1
  - 0.2 0.2 0.2
  - 9.0 9.0 9.0
  - 9.1 9.1 9.1
  - 9.2 9.2 9.2



- Each line is a 3-dimensional data point

Parse & cache the input dataset

- "data" RDD is now cached in main memory

```
>>> lines = sc.textFile("kmeans-data.txt")
>>> lines.collect()
[u'0.0 0.0 0.0', u'0.1 0.1 0.1', u'0.2 0.2 0.2', u'9.0 9.0 9.0', u'9.1 9
.1 9.1', u'9.2 9.2 9.2']
>>>
>>> def parseVector(line):
...     return np.array([float(x) for x in line.split(' ')])
...
>>> data = lines.map(parseVector).cache()
>>> data.collect()
[array([ 0.,  0.,  0.]), array([ 0.1,  0.1,  0.1]), array([ 0.2,  0.2,
0.2]), array([ 9.,  9.,  9.]), array([ 9.1,  9.1,  9.1]), array([ 9.2,
9.2,  9.2])]
```

Generating initial centers

- Recall takeSample() action
  - False: sample without replacement
  - K: sample size (= 2 here)
  - 1: seed for random number generator

```
>>> kPoints = data.takeSample(False, K, 1)
>>> kPoints
[array([ 0.1,  0.1,  0.1]), array([ 0.2,  0.2,  0.2])]
```

Assign point to its closest cluster

- Cluster 0 has points: (0, 0, 0) and (.1, .1, .1)
- Cluster 1 has the rest: (.2, .2, .2), (.9, .9, .9), …

```
>>> def closestPoint(p, centers):
...     bestIndex = 0
...     closest = float("+inf")
...     for i in range(len(centers)):
...         tempDist = np.sum((p - centers[i]) ** 2)
...         if tempDist < closest:
...             closest = tempDist
...             bestIndex = i
...     return bestIndex
...
>>> closest = data.map(lambda p: (closestPoint(p, kPoints), (p, 1)))
>>> closest.collect()
[(0, (array([ 0.,  0.,  0.]), 1)), (0, (array([ 0.1,  0.1,  0.1]), 1)),
(1, (array([ 0.2,  0.2,  0.2]), 1)), (1, (array([ 9.,  9.,  9.]), 1)), (
1, (array([ 9.1,  9.1,  9.1]), 1)), (1, (array([ 9.2,  9.2,  9.2]), 1))]
>>>
```

Getting statistics for each cluster

- pointStats has a key-value pair for each cluster

- Key is cluster # (0 or 1 for this example)
- Value is a tuple (sum, count)
  - sum = the sum of corresponding coordinates over all points in the cluster
  - Count = # of points in the cluster

```
>>> pointStats = closest.reduceByKey(lambda p1_c1, p2_c2: (p1_c1[0] + p2
_c2[0], p1_c1[1] + p2_c2[1]))
>>> pointStats.collect()
[(0, (array([ 0.1,  0.1,  0.1]), 2)), (1, (array([ 27.5,  27.5,  27.5]),
 4))]
```

Computing coordinates of new centroids

- Centroid coordinate = sum/count
  - E.g., cluster 0: [.1, .1, .1] /2 = [.05, .05, .05]

```
>>> newPoints = pointStats.map(lambda st: (st[0], st[1][0] / st[1][1])).
collect()
>>> newPoints
[(0, array([ 0.05,  0.05,  0.05])), (1, array([ 6.875,  6.875,  6.875]))
]
>>>
```

Can use mapValues here too:

newPoints = pointStats.mapValues(lambda stv: stv[0]/stv[1]).collect()

Distance between old & new centroids

- Old centroids: [.1, .1, .1] and [.2, .2, .2]
- New centroids: [.05, .05, .05] and [6.875, 6.875, 6.875]

Sum of squared distance between corresponding centroids

- Distance = $(.1-.05)^2*3 + (6.875-.2)^2*3 \sim 133.67$

For first centroid
(diff btw old and new)

For second centroid

```
>>> tempDist = sum(np.sum((kPoints[iK] - p) ** 2) for (iK, p) in newPoin
ts)
>>> tempDist
133.67437499999994
```

kmeans_example.py

clusters.centers
[array([9.1, 9.1, 9.1]), array([0.1, 0.1, 0.1])]

# of clusters

```
# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))
```

Return either 0 or 1

- With set = within cluster
- compute distance btw point and centroid of its closest cluster

Spark session

- spark = SparkSession.builder\
  .appName("PythonKMeans")\
  .getOrCreate()
- lines = spark.read.text('kmeans_data.txt').rdd.map(lambda r: r[0])

⇔ lines = sc.textFile('kmeans_data.txt')

Data frame

- Similar to a table in RDBMS
- df = spark.read.text('kmeans_data.txt')
- df.show()

```
+-----------+
|      value|
+-----------+
|0.0 0.0 0.0|
|0.1 0.1 0.1|
|0.2 0.2 0.2|
|9.0 9.0 9.0|
|9.1 9.1 9.1|
|9.2 9.2 9.2|
+-----------+
```

Turning data frame into an RDD

- df.rdd // this turns df into an rdd
  - MapPartitionsRDD[6] at javaToPython at NativeMethodAccessorImpl.java:0
- df.rdd.collect()

Each row is a tuple

  - [Row(value=u'0.0 0.0 0.0'), Row(value=u'0.1 0.1 0.1'), Row(value=u'0.2 0.2 0.2'), Row(value=u'9.0 9.0 9.0'), Row(value=u'9.1 9.1 9.1'), Row(value=u'9.2 9.2 9.2')]

Loading JSON documents

- df1 = spark.read.json('people.json')
- df1.show()

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

Each row is a tuple

- df1.rdd.collect()
  - [Row(age=None, name=u'Michael'), Row(age=30, name=u'Andy'), Row(age=19, name=u'Justin')]

people.json

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

Spark SQL

- df1.createOrReplaceTempView("people")
- sqlDF = spark.sql("SELECT * FROM people where age > 20")
- sqlDF.show()

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

Bisecting K-means

- A hierarchical divisive algorithm where division step is done by k-means (2-means)

1. Start with a single cluster with all data points
2. Repeat until found desired # of clusters (k)
   a) Find a cluster C to split
   b) Split C into C_1 and C_2 using k-means alg. with k=2

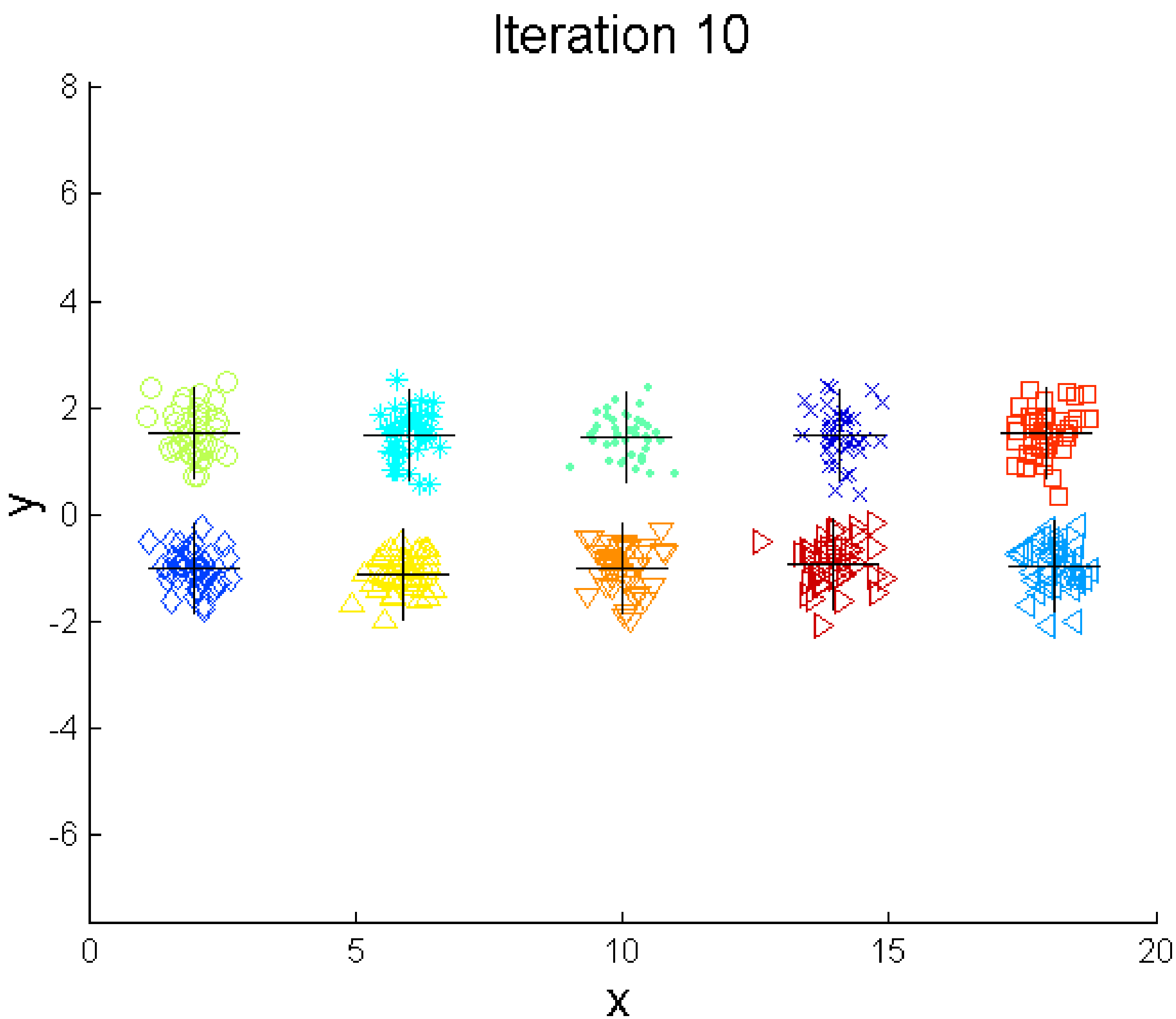
May also run step 2.b multiple times and pick the best clustering

Find a cluster to split

- A cluster with the largest # of data points

- A cluster with the largest diameter
  - Recall the diameter is the distance between two farthest points in the cluster

Example

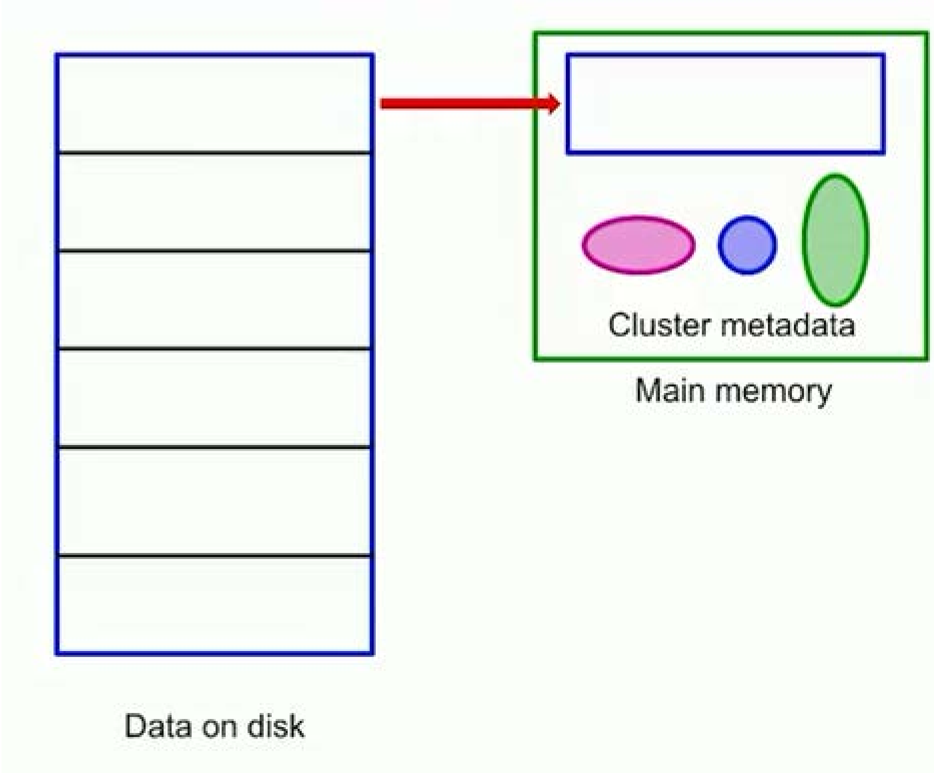


Roadmap

- Problem, types, and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR
  - CURE
- Curse of dimensionality

BFR [Bradley-Fayyad-Reina] Algorithm

- Extend k-means to handle large data set
  - So large that can not be fit in main memory
  - Need to process one chunk at a time



BFR Algorithm

- Select k points as initial centroids
  - E.g., k points farthest away from each other in 1^{st} chunk

- Load one chunk of data into memory at a time

- For each chunk, its points are either:
  a) assigned to existing clusters, or
  b) used to form new mini-clusters, or
  c) retained

  Points in case a and b are not retained in main memory

Case a: assigned to existing cluster

- When point is sufficiently close to the centroid of the cluster

- Update summary of cluster C_i
  - N: # of points in C_i
  - SUM_i: Sum of values of points in each dim
  - $SUMSQ_i$: Sum of squared values of points in each dim

  => 2d + 1 values, where d = # of dimensions

Cluster Summary

- Points in cluster: (5, 1), (6, -2), (7, 0)
- N = 3, SUM = [18, -1], SUMSQ = [110, 5]

$\Rightarrow$Centroid = SUM/N = [6, -1/3]

$\Rightarrow$Variance = SUMSQ/N – $(SUM/N)^2$

= [$110/3 - 6^2$, $5/3 - (-1/3)^2$] = [.667, 1.56]

=>Standard deviation = [.816, 1.25]

Variance

- $Var(X) = E\left[\left(X - E(X)\right)^2\right]$

$$= E[X^2 - 2XE[X] + (E[X])^2)]$$

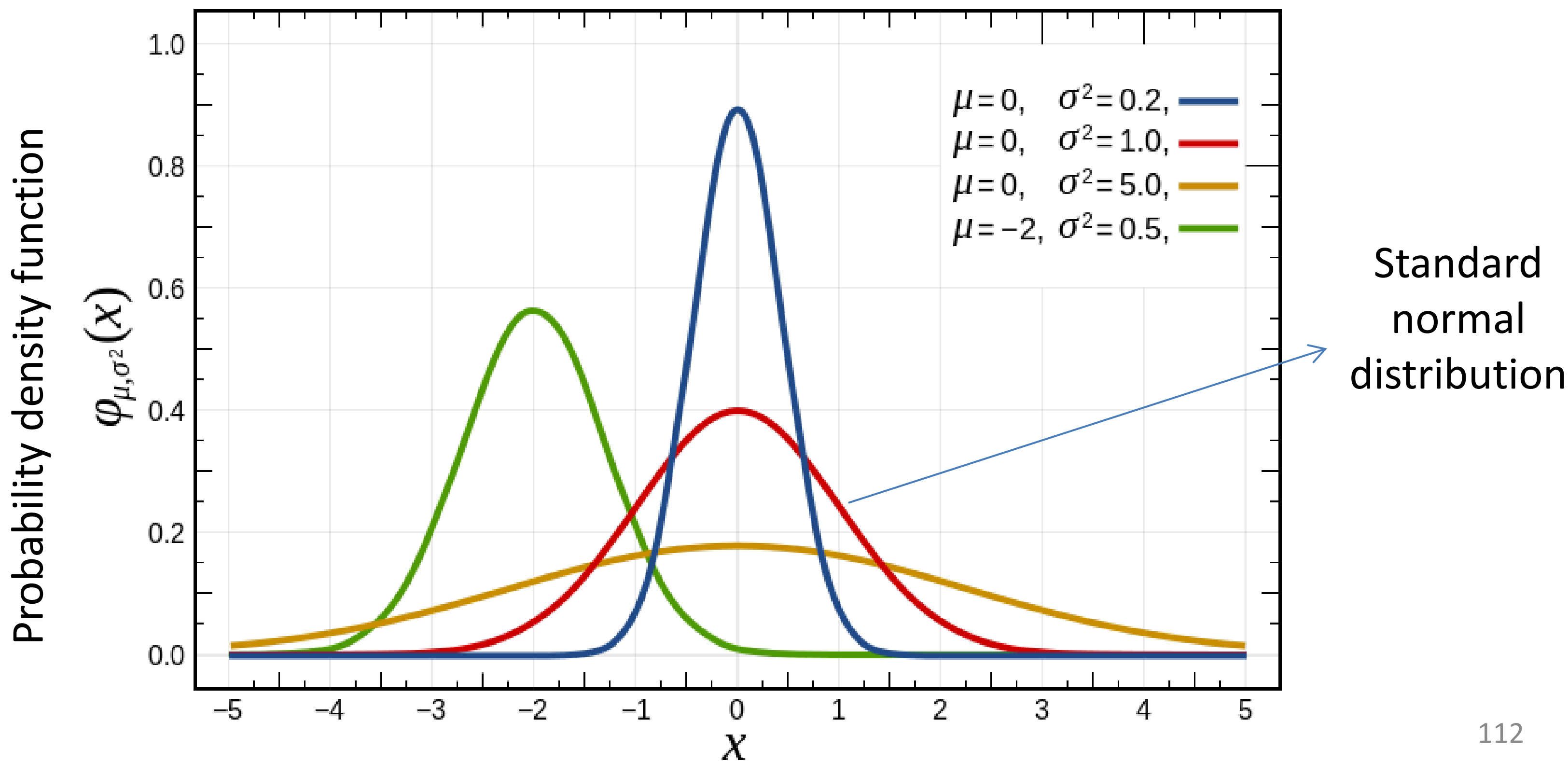
$$= E[X^2] - 2E[X]E[X] + (E[X])^2$$

$$= E[X^2] - (E[X])^2$$

Define “Sufficiently Close”

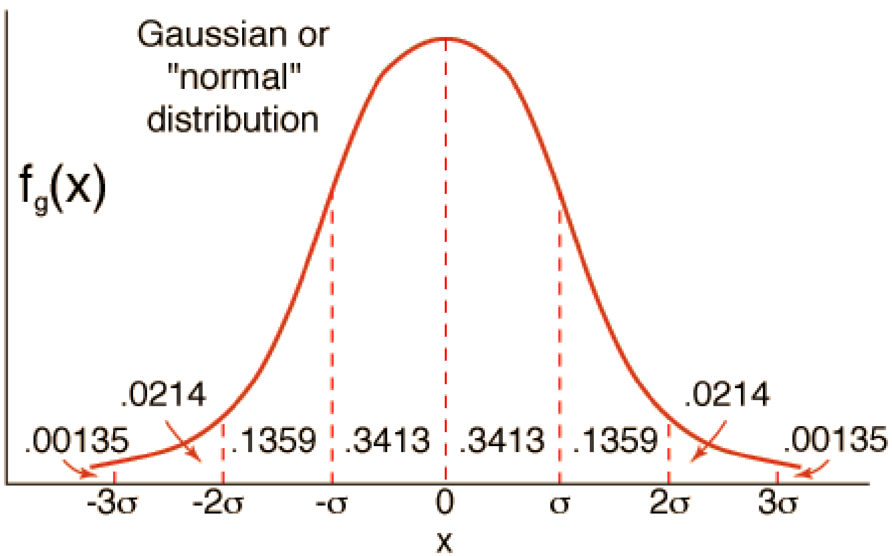
- Assume points in cluster are normally distributed

  => we know prob. of particular distance from mean



Define “Sufficiently Close”

- ~68% of points: 1 σ away from mean
- ~95% of points: 2 σ away
- ~99% of points: 3 σ away



Mahalanobis Distance

- Normalized distance for multi-dimensional data
  - How many σ away from centroid
  - This assumes no-correlation among diff. dimensions

Combine dist. in Euclidean space

$$\sqrt{\sum_{i=1}^{d}\left(\frac{p_i - c_i}{\sigma_i}\right)^2}$$

Normalized distance in i-th dim

- Point p: $[p_1, \ldots, p_d]$; centroid: $[c_1, \ldots, c_d]$

Multivariate Normal Distribution

Covariance matrix Column vector

$$f_{\mathbf{x}}(x_1, \ldots, x_k) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}\underbrace{(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}_{\text{Squared Mahalanobis distance}}\right)$$

Non-Correlating Dimensions

- When dimensions are not correlated, covariance matrix becomes diagonal

$$\Sigma = \begin{bmatrix} \mathrm{E}[(X_1-\mu_1)(X_1-\mu_1)] & 0 & \cdots & 0 \\ 0 & \mathrm{E}[(X_2-\mu_2)(X_2-\mu_2)] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathrm{E}[(X_n-\mu_n)(X_n-\mu_n)] \end{bmatrix}.$$

Squared Mahalanobis Distance

- Two-dimensional case

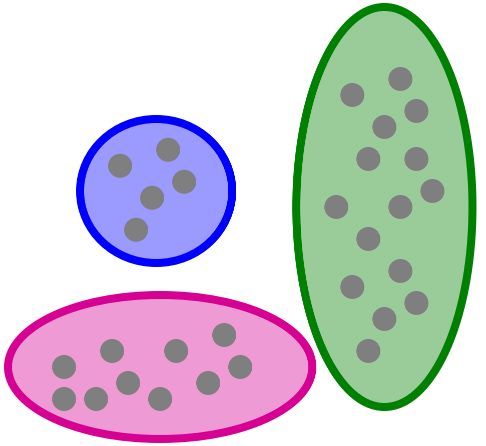
$$
\begin{bmatrix} x_1 - \mu_1 & x_2 - \mu_2 \end{bmatrix} \begin{bmatrix} {\sigma_1}^2 & 0 \\ 0 & {\sigma_2}^2 \end{bmatrix}^{-1} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}
$$
$$
= \frac{(x_1 - \mu_1)^2}{{\sigma_1}^2} + \frac{(x_2 - \mu_2)^2}{{\sigma_2}^2}
$$

Define “Sufficiently Close”

- Use Mahalanobis to measure distance
- Pick centroid with smallest distance
- If distance < threshold (e.g., 4), add point to cluster
  - Prob. of 4σ away from mean is less than 10^{-6}

Assumptions

- Axes of cluster (ellipse) align with axes of space



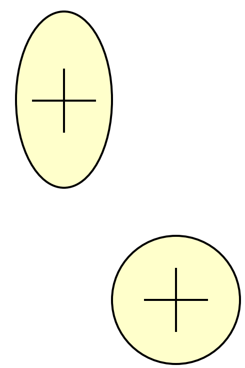
Case b: form new mini-clusters

- Use
  - points not assigned to any existing clusters
  - points retained from last rounds

- Can use a hierarchical clustering algorithm with proper stopping condition

Merge Mini-Clusters

- Merge new and existing mini-cluster if variance of merged cluster is small enough
  - Variance can be computed from: N, SUM, SUMSQ



- Note that none of mini-clusters can be merged with existing (non-mini) clusters

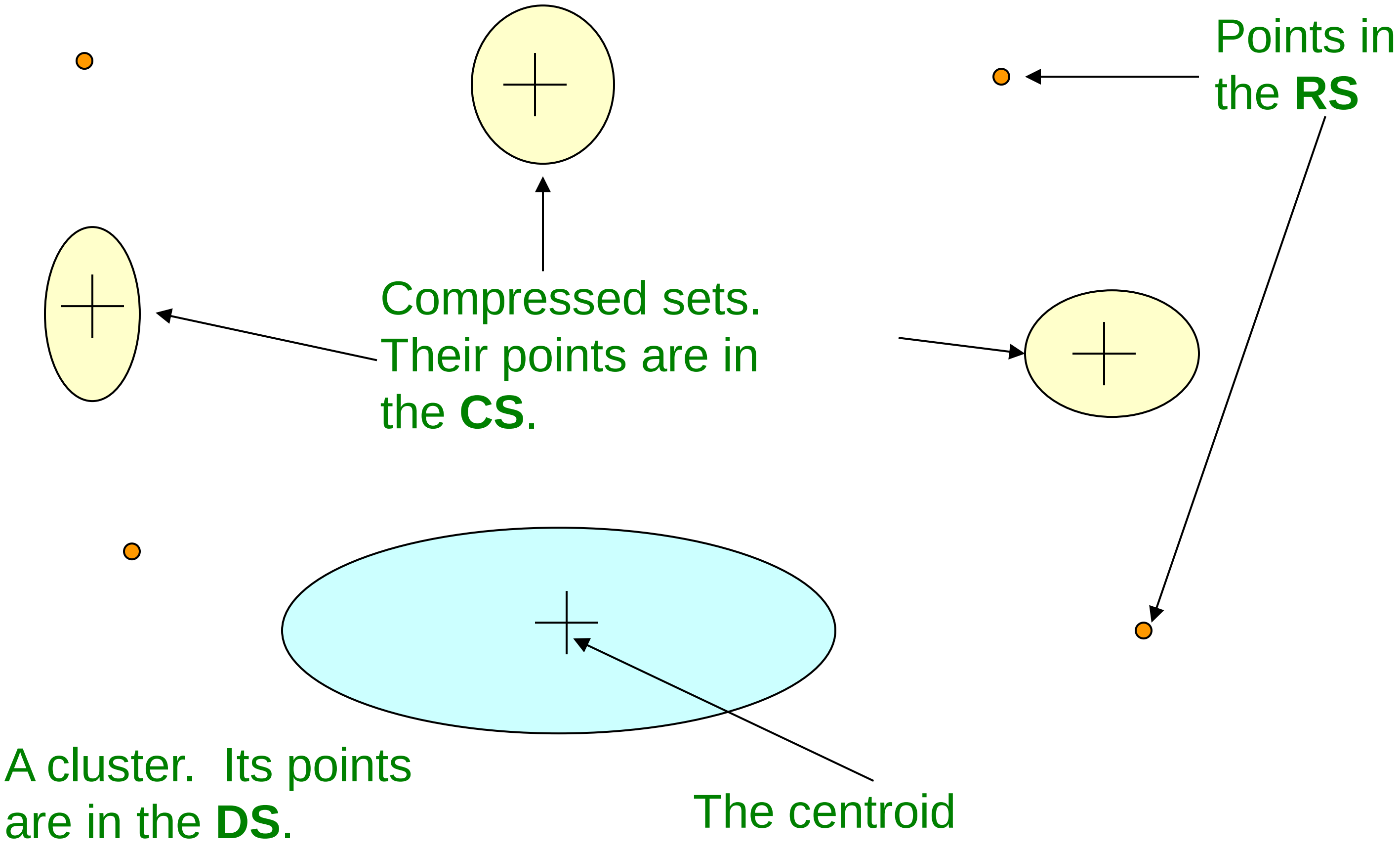
Case c: retained points

- Points in the singleton mini-clusters
- Retained for the next round

Classification of Points

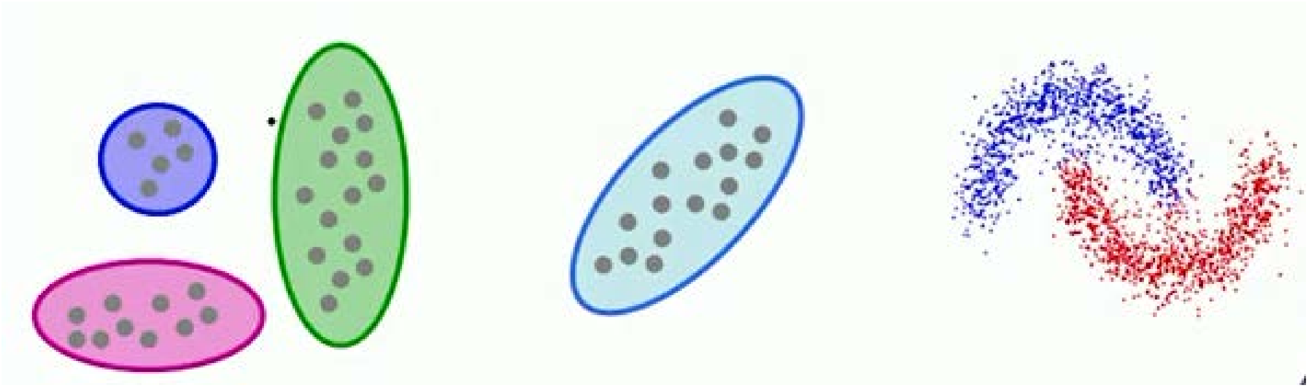
- Discard set (DS)
  - Points close enough to an existing cluster
- Compression set (CS)
  - Points close to each other to form mini-clusters
  - But not close enough to any existing cluster
- Retained set (RS)
  - Isolated points retained for next rounds

BFR: "Galaxy" Picture



Limitations of BFR

- Strong assumptions about clusters
  - Normally distributed in each dimension
  - Axis-parallel: not ok to have ellipses at an angle



(a) OK

(b) Not OK

(c) Not OK

Roadmap

- Problem, types, and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR
  - CURE
- Curse of dimensionality

CURE (Clustering Using REpresentatives)

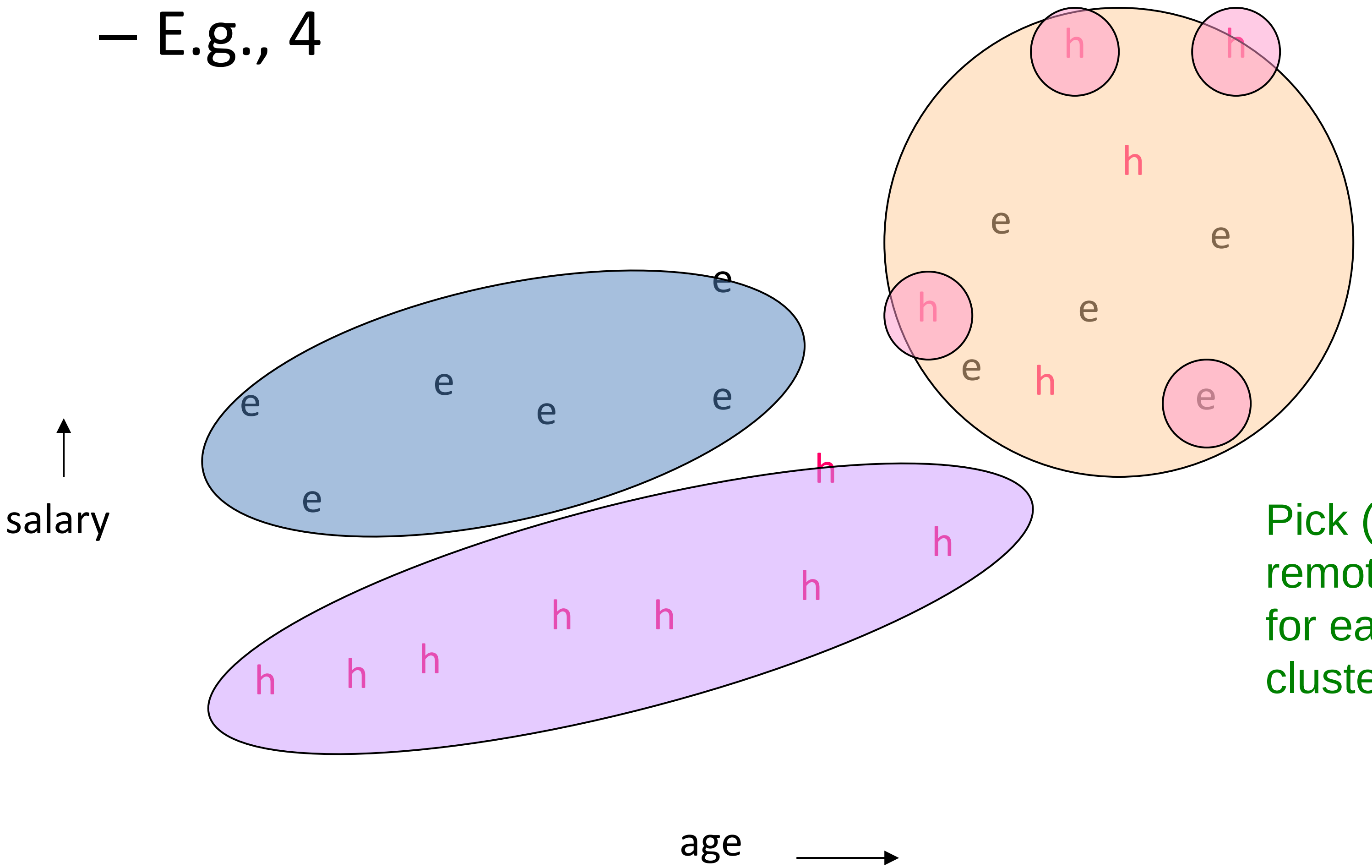
- Also handle large-scale data

- Two passes:
  1. Pick a sample, cluster it hierarchically, and determine k clusters from dendrogram
  2. Scan data and assign points to closest cluster

- Distance between point p and cluster C
  – Distance of p from the closest representative in C

Cluster Representatives

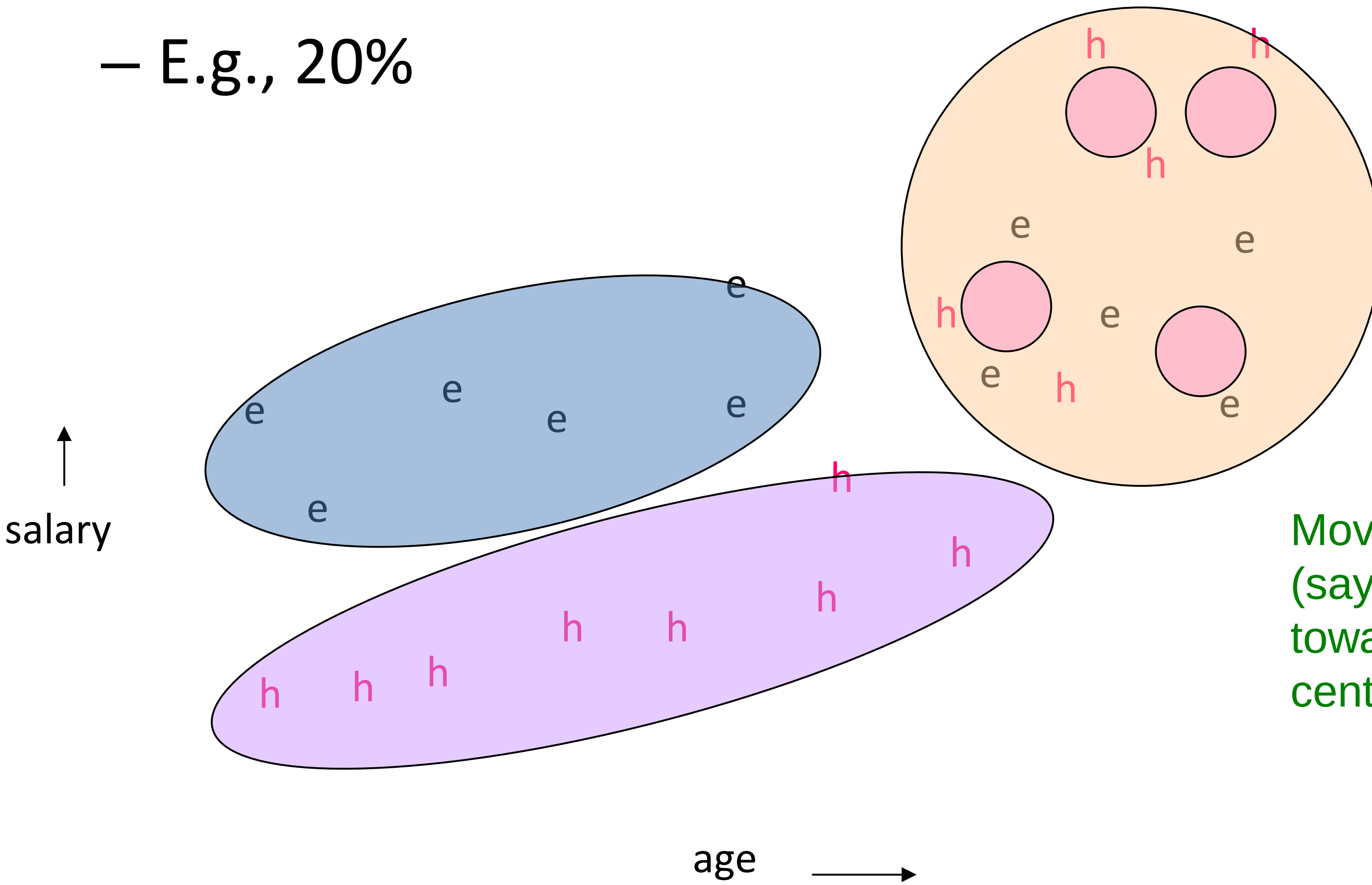
- A small set of points far away from each other
  - E.g., 4



Pick (say) 4 remote points for each cluster.

Moving Representatives

- A fixed fraction of distance toward centroid
  - E.g., 20%



Move points (say) 20% toward the centroid.

Compare CURE with BFR

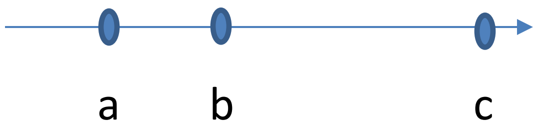
- Distribution of data
  - CURE: do not assume any particular distribution
  - BFR: data should be normally distributed
- Representation of cluster
  - CURE: a set of representatives
  - BFR: centroid
- Common: both assume data in Euclidean space

Roadmap

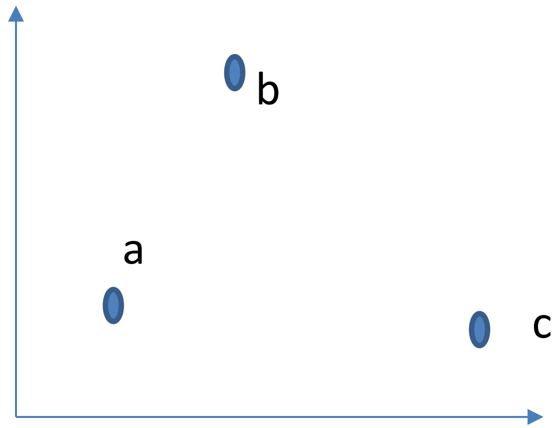
- Problem, types, and distance functions
- Hierarchical clustering
- Point assignment
  - K-means
  - BFR
  - CURE
- Curse of dimensionality

Effect of High Dimension: Euclidean

- Consider a set of data points on a line
  - dist(a, b) < dist(a, c)

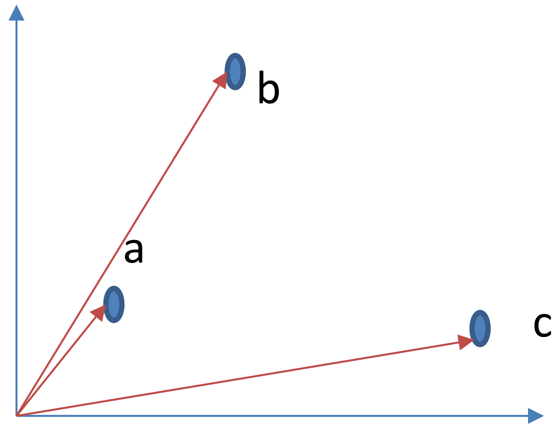


- Consider increasing the dimension by 1
  - dist(a, b) ~ dist(a, c)

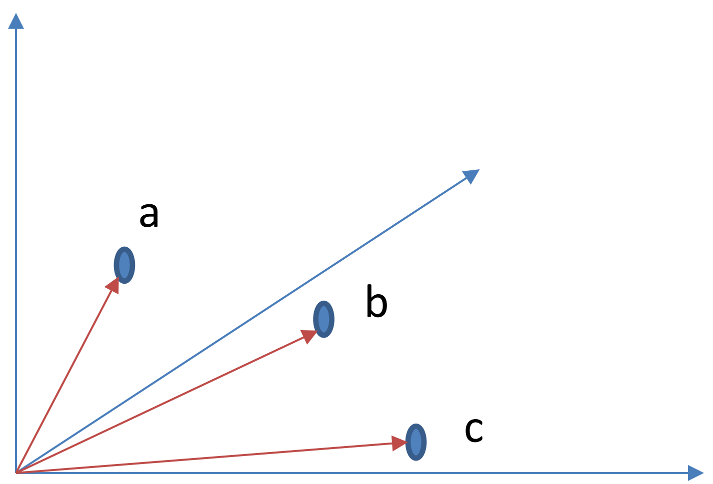


Effect of High Dimension: Cosine

- Cosine(a, b) > Cosine(a, c)



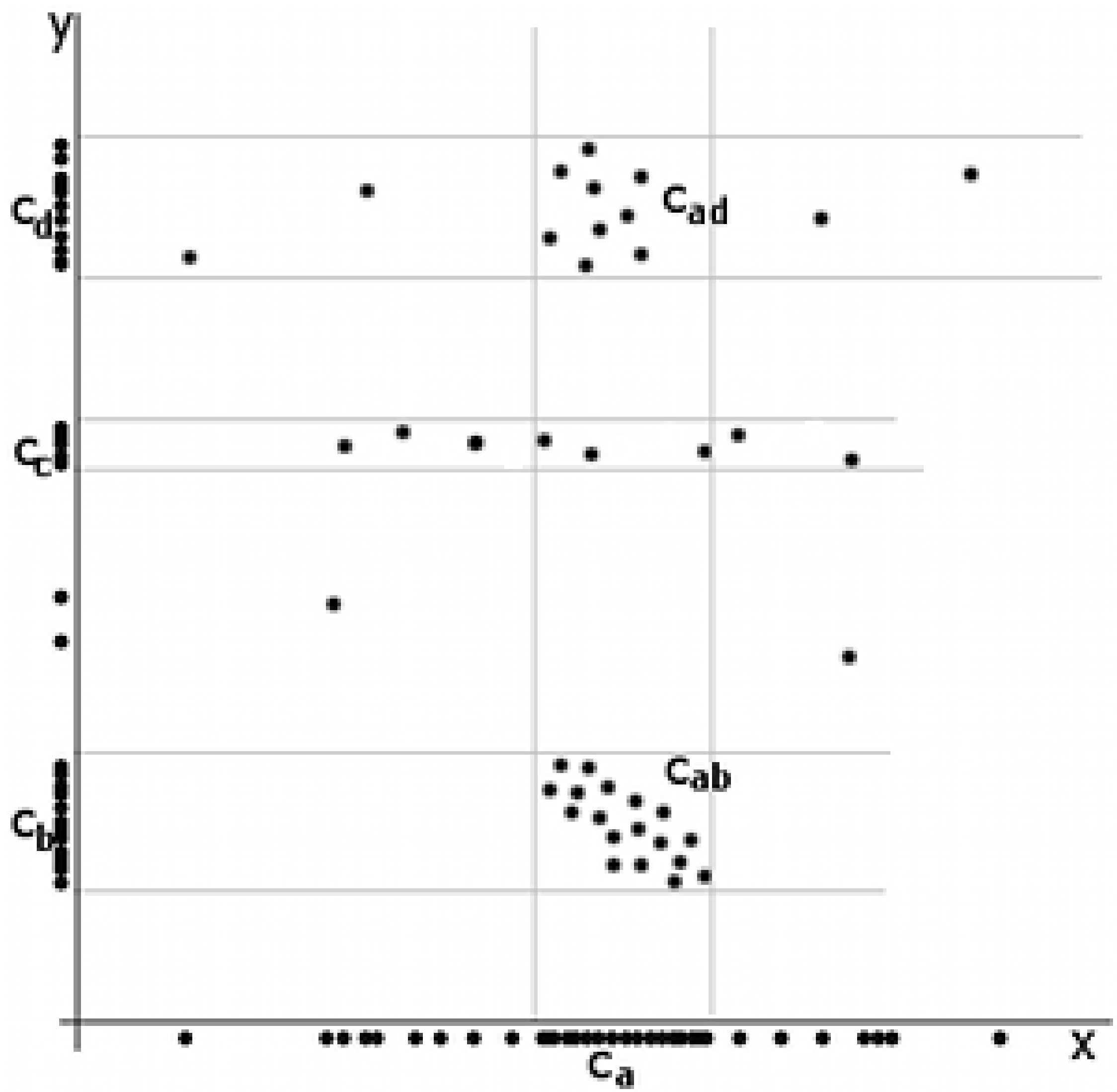
- Increase d to 3
  - Cosine(a, b) ~ Cosine(a, c)
- Higher d
  - Angle -> 90°
  - Cosine -> 0



Curse of Dimensionality

- Data points have similar distance btw each other
  - Euclidean distance breaks
- Data vectors become orthogonal
  - Cosine function breaks

Subspace Clustering



References

- Spark SQL, DataFrames and Datasets Guide
  - http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes