Algorithms and Data Structures for Data Science

Sets and Sketches

CS 277

December 6, 2021

Brad Solomon

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Lab_hash is optional

Release date: December 1st

Due date: December 8th

Wednesday Lecture ‘Open Office Hours’

Short lecture at beginning on special topics / concluding thoughts

Work on finishing up your final projects

Final Project Deliverables

Only code in final project repo by 12/12/21 will be graded!

Written report (.PDF or .MD) must be in results directory

Report focuses on summary of final dataset and results

Final presentation link should be in results directory

Presentation focuses on development and concluding thoughts

Don’t forget your README and Development log!

ICES Evaluations

You are strongly encouraged to provide course feedback

go.illinois.edu/ices-online

Learning Objectives

Motivate and define sets

Introduce set notation and set similarity

Show how sets can be used to address large-scale problems

Fundamental Data Structures

List

Stack and Queue

Tree

Graph

Hash Table

Set

Sets

A set is an **unordered** and **unindexed** collection of data

sets.py

```
1  mySet = set(["A", "B", "C", "D", "E"])
2
3  print(mySet)
4
5  mySet.add("F")
6
7  print(mySet.pop())
8  print(mySet)
9
10
11
```

Sets

By convention a set contains **no duplicates**.

sets.py

```
1  mySet = set(["A", "B", "C", "D", "E", "E"])
2
3  print(mySet)
4
5  mySet.add("A")
6
7  mySet.add("F")
8
9  print(mySet.pop())
10 print(mySet)
11
```

Sets

In Python, a set can only store immutable data — **why?**

sets.py

```
1  #mySet = set([ ['Not allowed', "List"] ])
2  mySet = set([ ("Am Allowed", "Tuple") ])
3
4  print(mySet)
5
6  mySet.add("A")
7  mySet.add("F")
8
9  print(mySet.pop())
10 print(mySet)
11
```

Logic used sets (we ignored it)

*The **universal quantifier** states a predicate is True for all values in the domain*

$P(x)$ = "Person x gets a car!" D = Everyone in the audience

$$\forall x \in D, P(x)$$

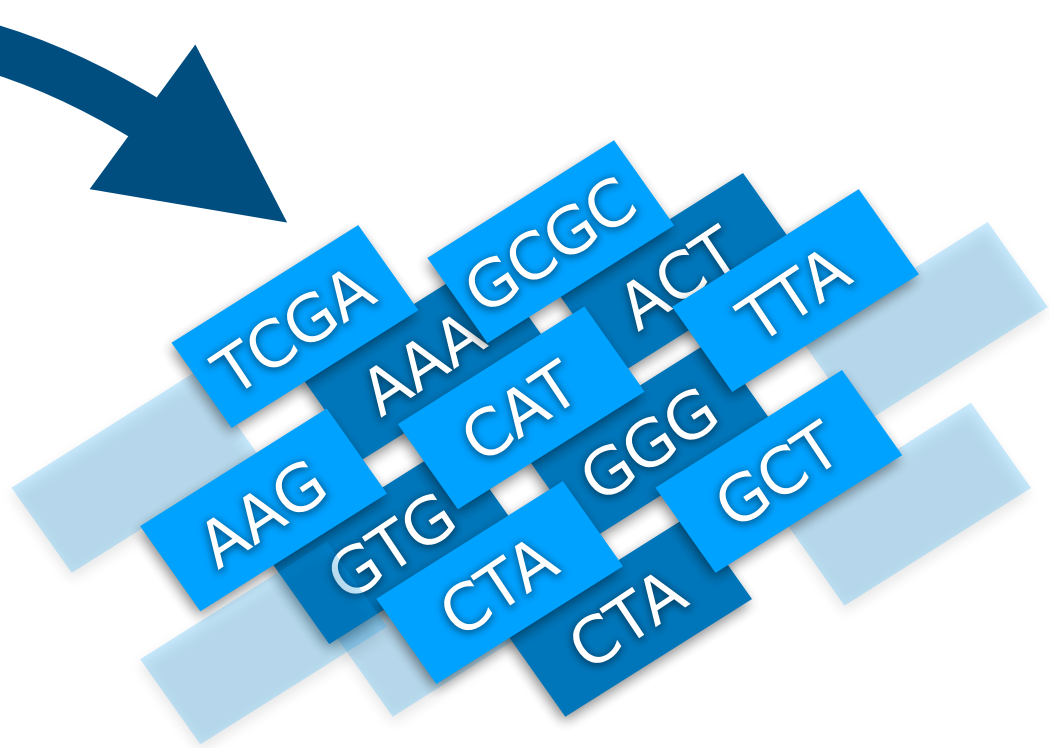
*The **existential quantifier** states that there exists a value in the domain for which a predicate is True*

$P(x)$ = "I've read x" D = All books in my library

$$\exists x \in D, \neg P(x)$$

Data processing using sets

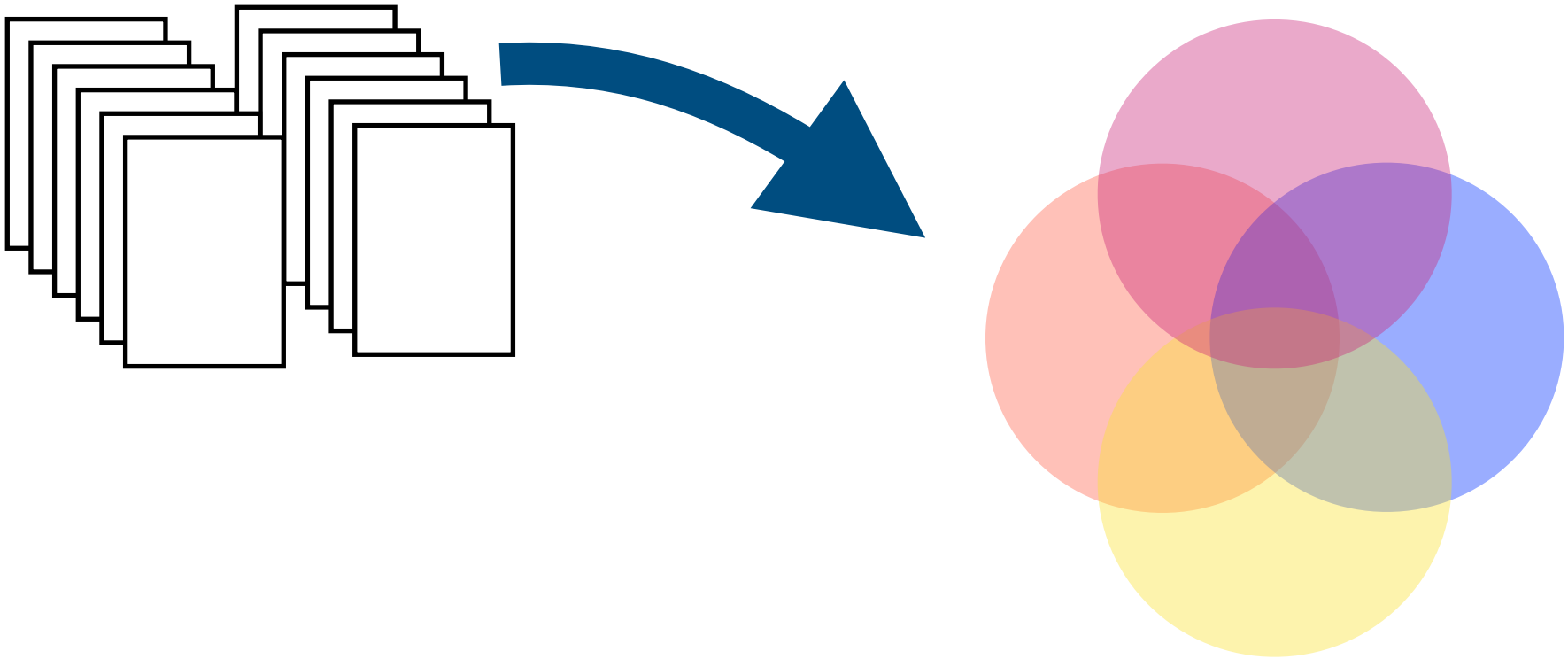
```
>Read 1
ATGGTTAGAATTAAACCCGG
TGCTAATAAACCUAGTGATG
>Read 2
CGATAGCACAGGTAGATCC
TACGTAGAGGTCATTAGCC
>Read 3
TACGTAGAGGTCATTAGCCG
TGCTAATAAACCUAGTGATG
```



How many unique objects are present?

Is a particular object or set of objects **present**? **Absent**?

Data processing using sets



Find other files that are ***similar*** to me

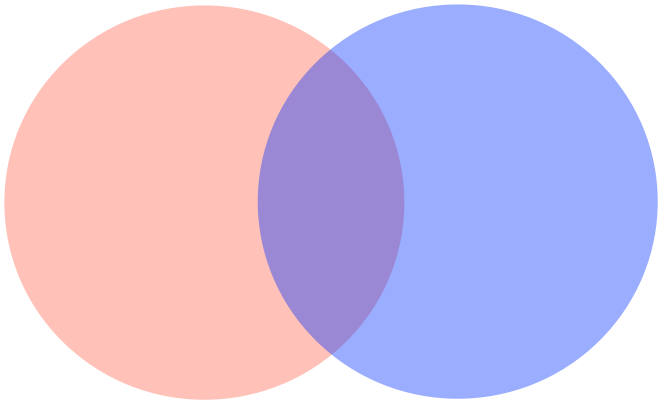
What objects can be found in **multiple sets**? What objects are **unique**?

Set Union

The **union** of two or more sets is _____.

sets.py

```
1  set1 = set([1,2,3,4,5])
2  set2 = set([2,4,6,8,10])
3
4  set3 = set1.union(set2)
5
6  set2.update(set1)
7
8
9
10
11
```

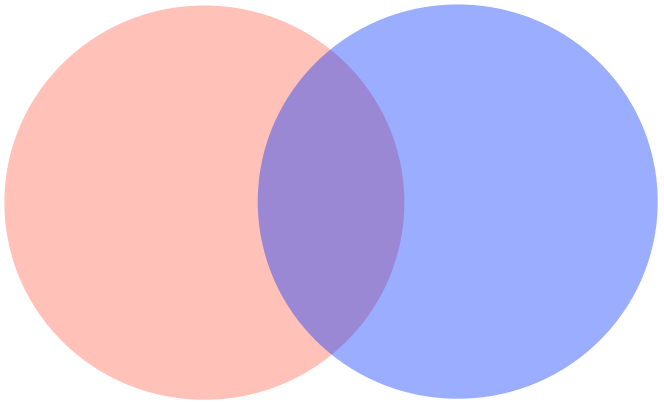


Set Intersection

The **intersection** of two or more sets is _____.

sets.py

```
1  set1 = set([1,2,3,4,5])
2  set2 = set([2,4,6,8,10])
3
4  set3 = set1.intersection(set2)
5
6
7
8
9
10
11
```

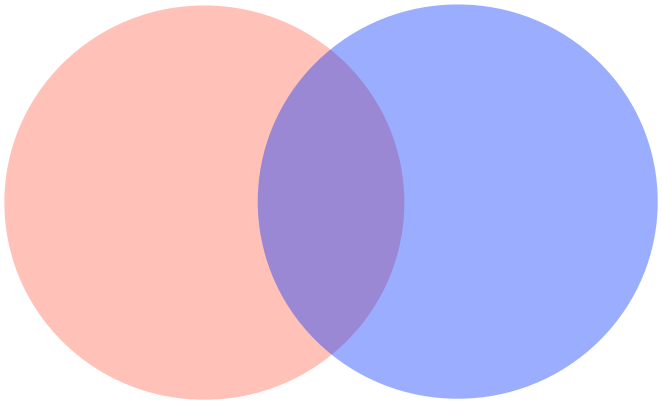


Set Difference

The **difference** of two sets is _____.

sets.py

```
1  set1 = set([1,2,3,4,5])
2  set2 = set([2,4,6,8,10])
3
4  set3 = set1.difference(set2)
5
6
7
8
9
10
11
```

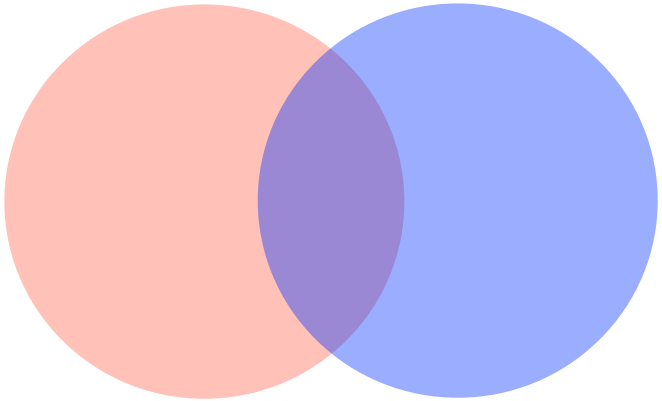


Set Symmetric Difference

The **symmetric difference** of two sets is _____.

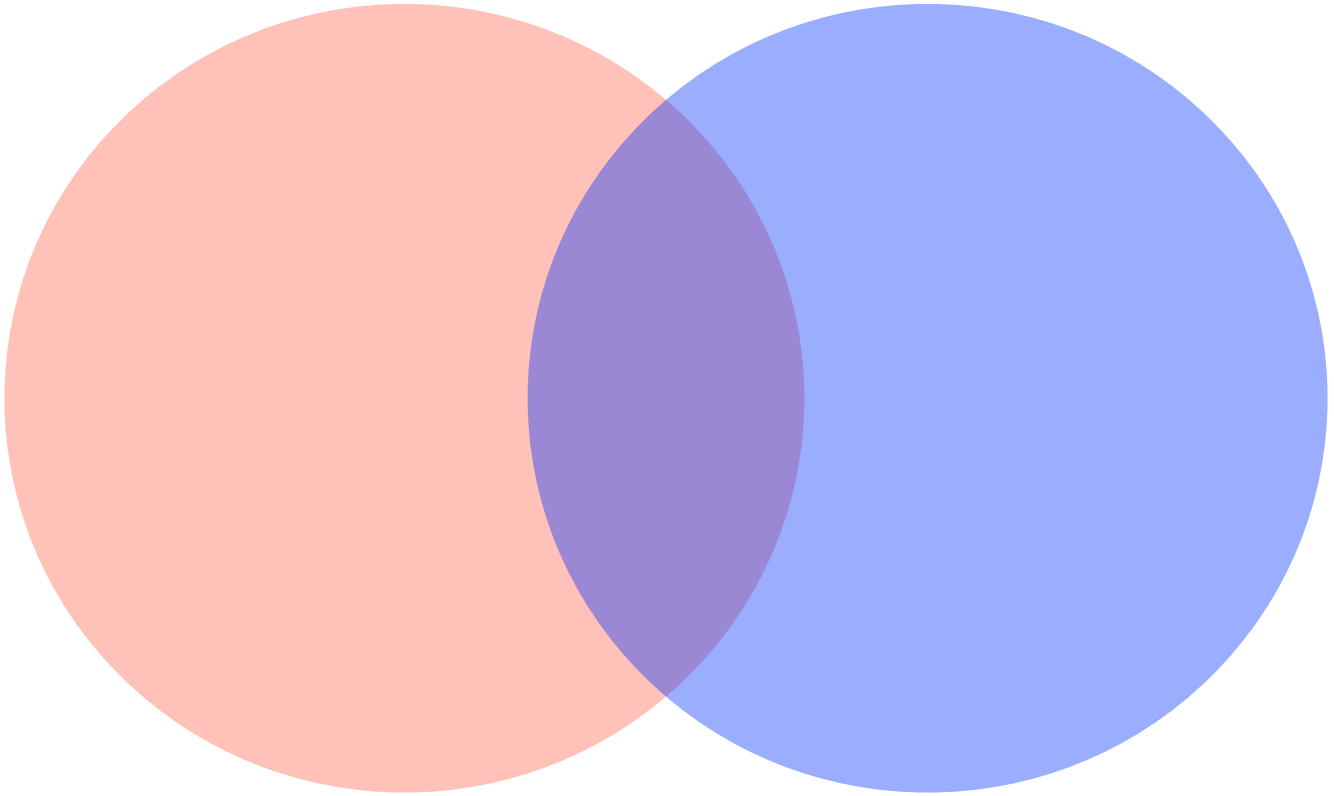
sets.py

```
1  set1 = set([1,2,3,4,5])
2  set2 = set([2,4,6,8,10])
3
4  set3 = set1.symmetric_difference(set2)
5
6
7
8
9
10
11
```



Inclusion-Exclusion Principle

$$|A \cup B| =$$



Set Operations

$A = \{1, 2, 3, 4\}$ $B = \{3, 4, 5, 6, 7\}$

Union $A \cup B$

Intersection $A \cap B$

Difference $A \setminus B$

Symmetric difference $A \triangle B$

Uses for sets

1) Sets (as a hash table) are an efficient data structure for **find**

2) Sets are a natural way to identify **cardinality**

3) Sets can be used to **estimate similarity** between objects

Cardinality

The **cardinality** of a set is the number of unique objects.

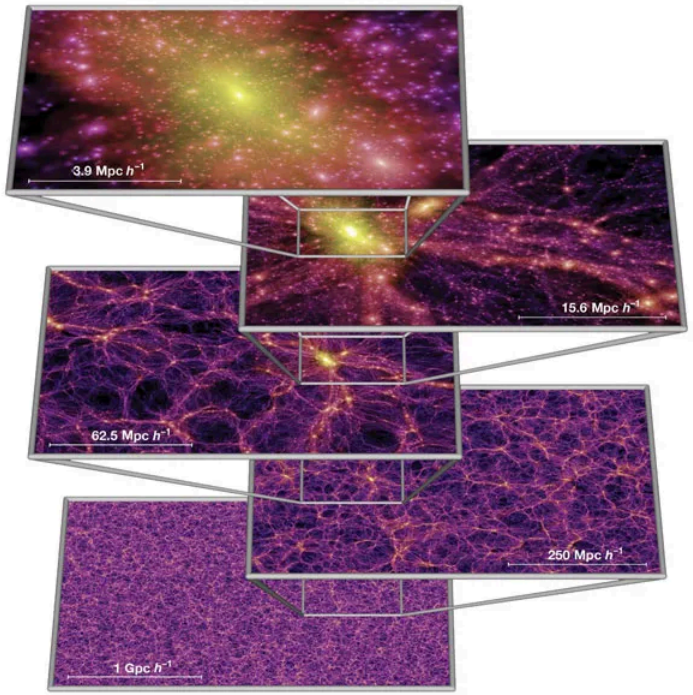
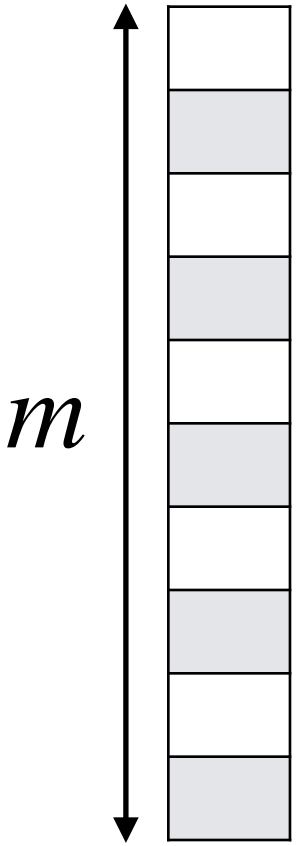


Image: https://doi.org/10.1038/nature03597



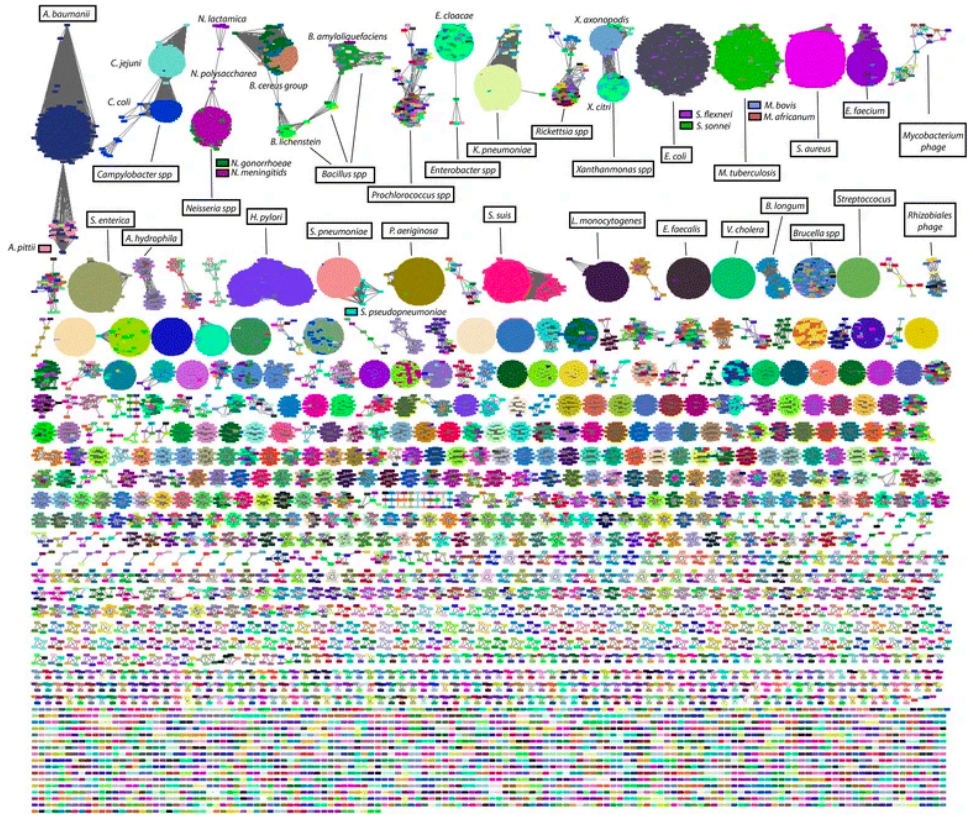


Image: https://doi.org/10.1186/s13059-016-0997-x

List Cardinality

cardinality.py

```
 1 def naive_cardinality(li):
 2     card=0
 3
 4     for i,v1 in enumerate(li):
 5         new=True
 6
 7         for v2 in s[:i]:
 8             if v1 == v2:
 9                 new=False
10                 break
11
12         if new==True:
13             card+=1
14
15     return card
16
17
```

Time:

Space:

List Cardinality

cardinality.py

```
1  def sort_cardinality(li):
2      li.sort()
3      card=1
4
5      for i in range(len(li[1:])):
6          if li[i] != li[i-1]:
7              card+=1
8
9      return card
10
11
12
13
14
15
16
17
```

Time:

Space:

Set Cardinality

cardinality.py

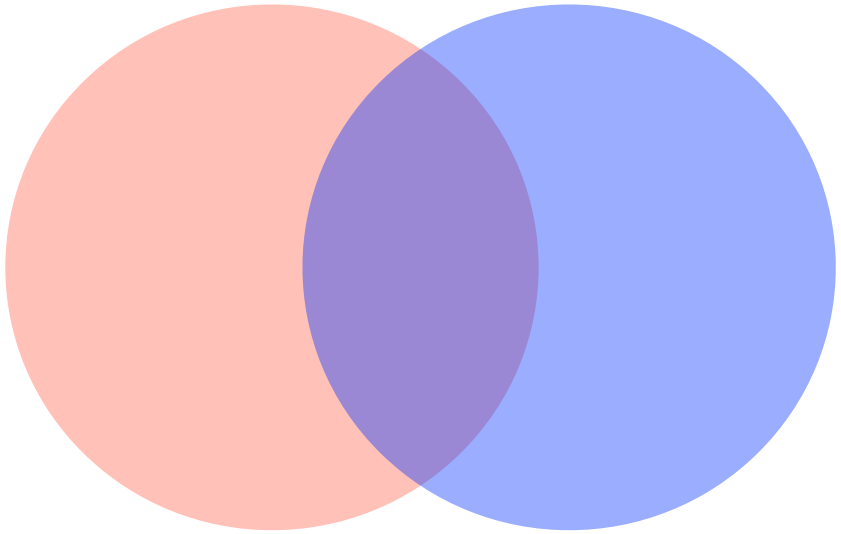
```
1  def set_cardinality(s):
2
3      mySet = set([s])
4
5      return len(mySet)
6
7
8
9
10
11
12
13
14
15
16
17
```

Time:

Space:

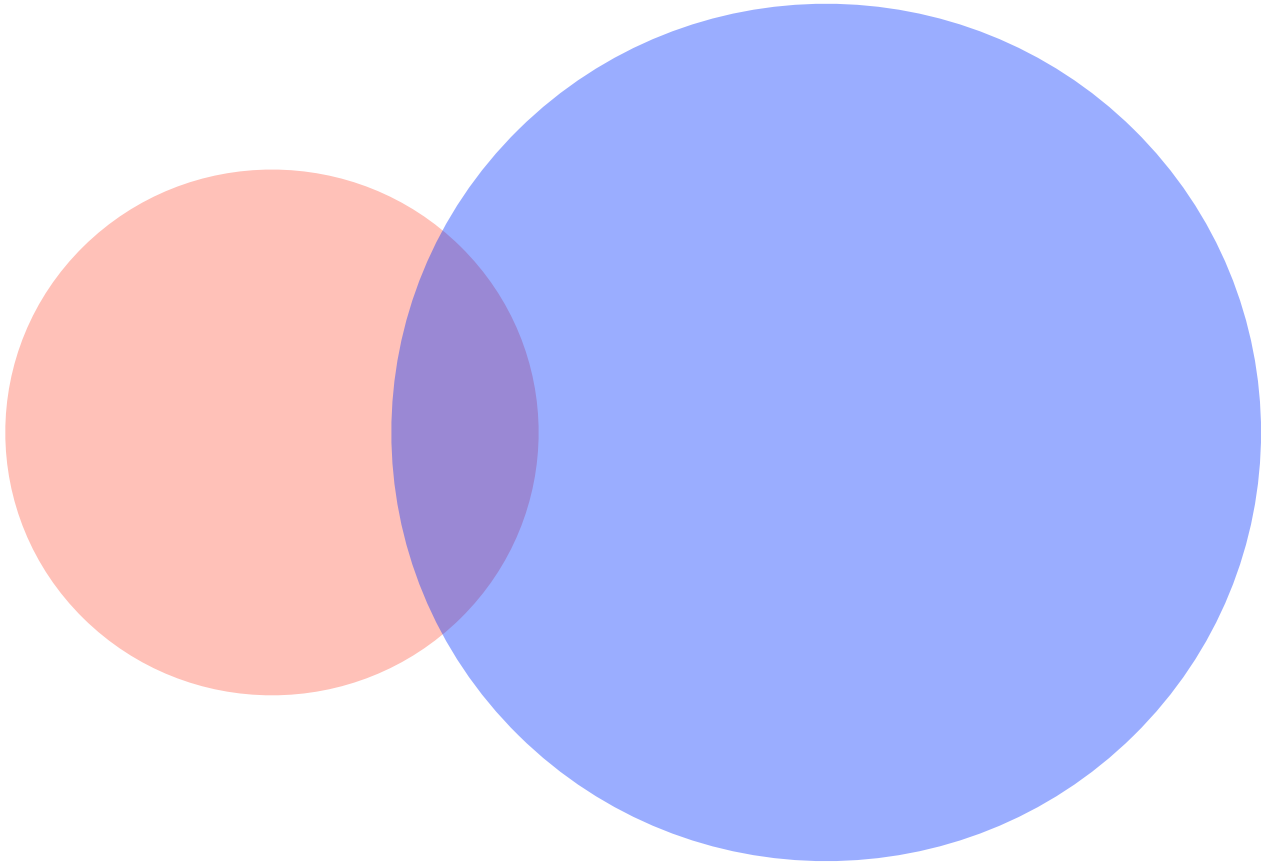
Set Similarity

Given two sets A & B, how can we describe how ***similar*** they are?



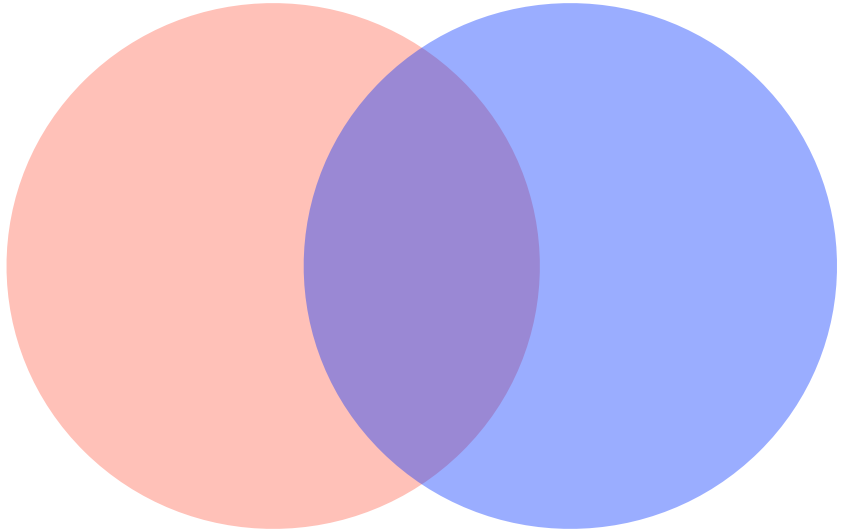
Set Similarity

Given two sets A & B, how can we describe how ***similar*** they are?



Set Similarity

To measure **similarity** of A & B, we need both a measure of how similar the sets are but also the total size of both sets.



$$J = \frac{|A \cap B|}{|A \cup B|}$$

J is the ***Jaccard coefficient***

Jaccard Coefficient

$$J = \frac{|A \cap B|}{|A \cup B|}$$

$$= \frac{|A \cap B|}{|A \cap B| + |A \triangle B|}$$

$$= \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

"Double counting" to eliminate union

$$= \frac{|A| + |B| - |A \cup B|}{|A \cup B|}$$

"Double counting" to eliminate intersection

Jaccard Similarity

$$A = \{1, 2, 3, 4\} \qquad B = \{3, 4, 5, 6, 7\}$$

Cardinality in the real world

Cardinalities

$$|A|$$

$$|B|$$

$$|A \cup B|$$

$$|A \cap B|$$

Set similarities

$$O = \frac{|A \cap B|}{min(|A|, |B|)}$$

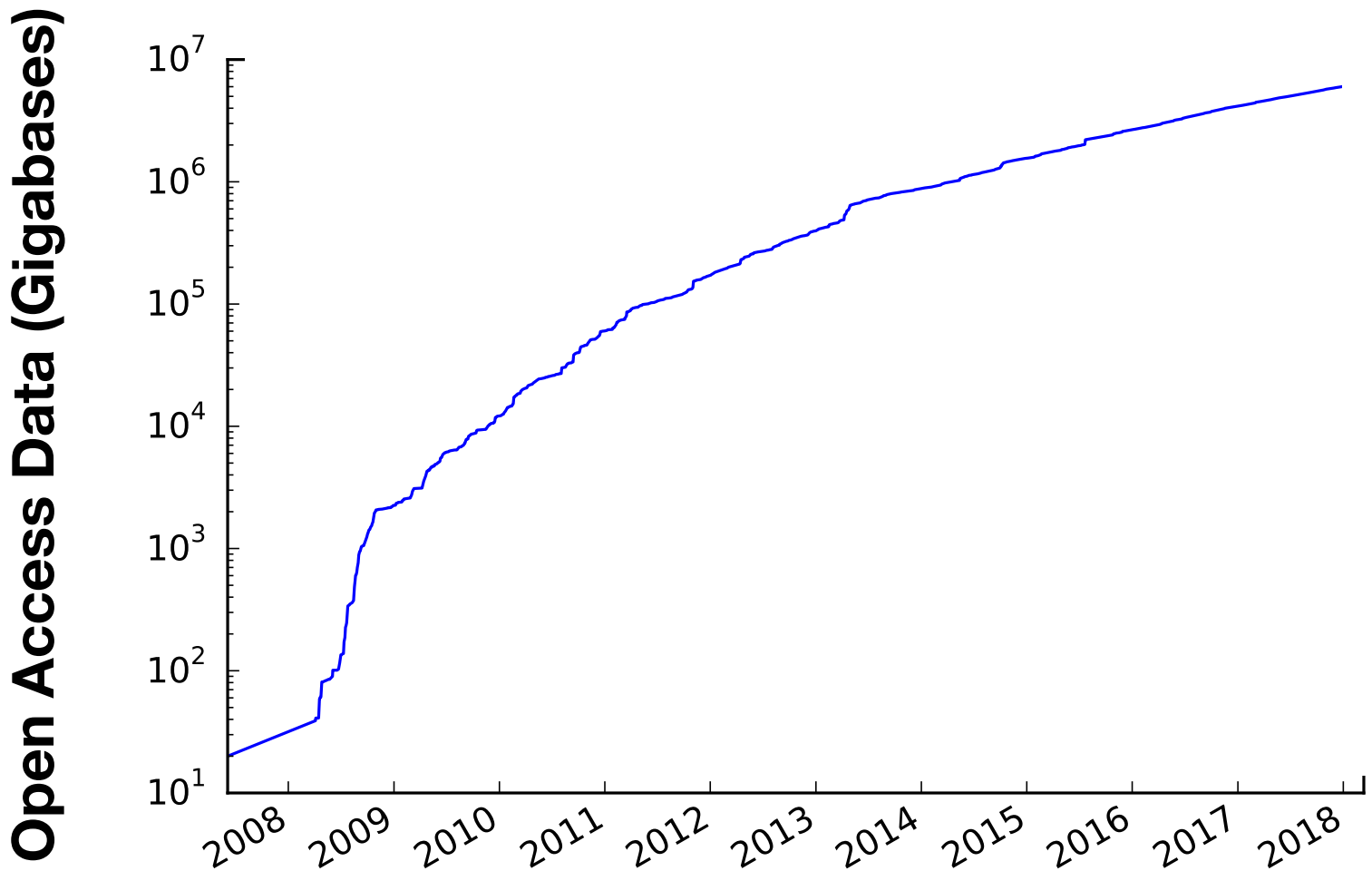
$$J = \frac{|A \cap B|}{|A \cup B|}$$

Real-world Meaning

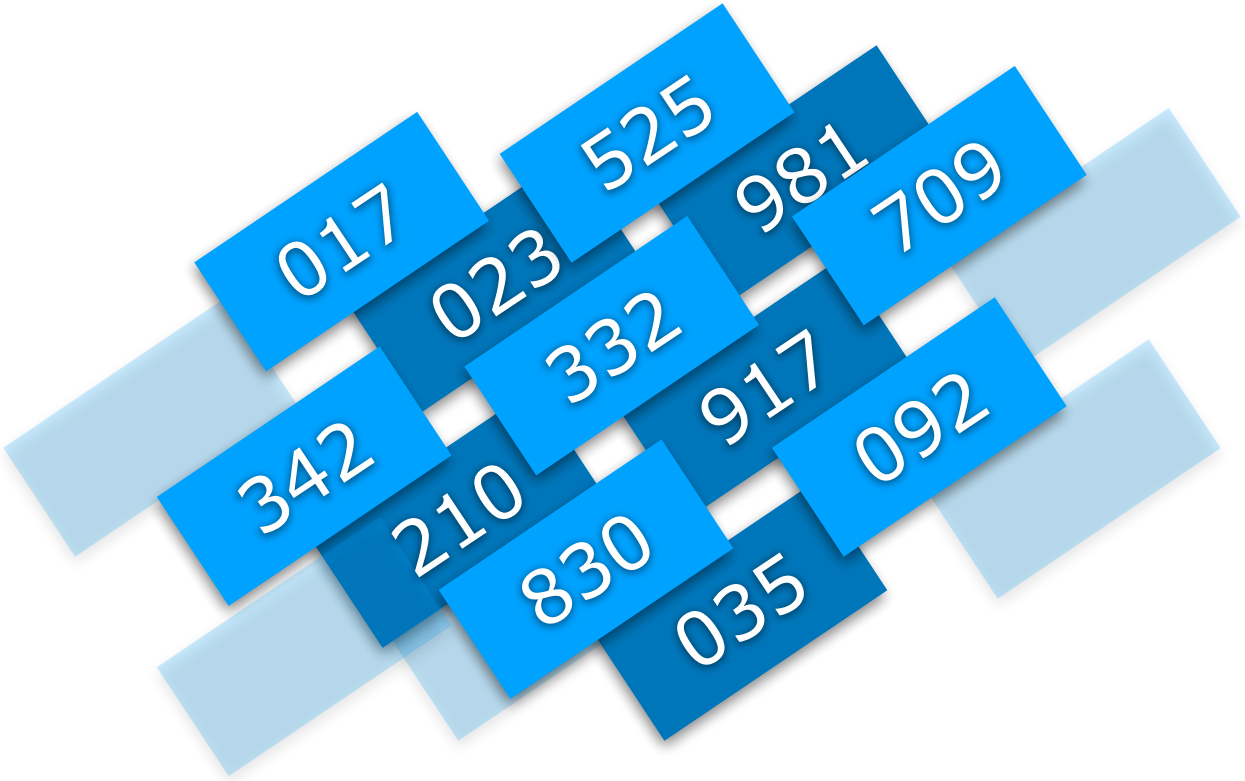
```
AGGCCACAGTGTATTATGACTG
|||||||||||  |||||||||
AGGCCACAGTGAGTTATGACTG

AAAAAAAAAAAGATGT-AAGTA
|||||||||||||||| |||||
AAAAAAAAAAAGATGTAAAGTA

GAGG--TCAGATTCACAGCCAC
||||  ||||||||||||||||
GAGGGGTCAGATTCACAGCCAC
```

Estimating Cardinalities

Claim: We don't need to look at all items in a set to estimate similarity!



Cardinality Estimation



I take cards labeled 1--1,000 and choose a random subset of size N to hide in my hat

You would like to estimate N

You may see **one representative** from the cards in the hat; which to pick?

Minimum, median, maximum? Something else?

Cardinality Estimation

What if **minimum** was 500? ...10? ... 4?

If minimum is 95, what's our estimate for N?

Informally: N points scattered randomly across interval
divide it in $N+1$ parts, each about $1000/(N+1)$ long

Cardinality Estimation

What if **minimum** was 500? ...10? ... 4?

If minimum is 95, what's our estimate for N?

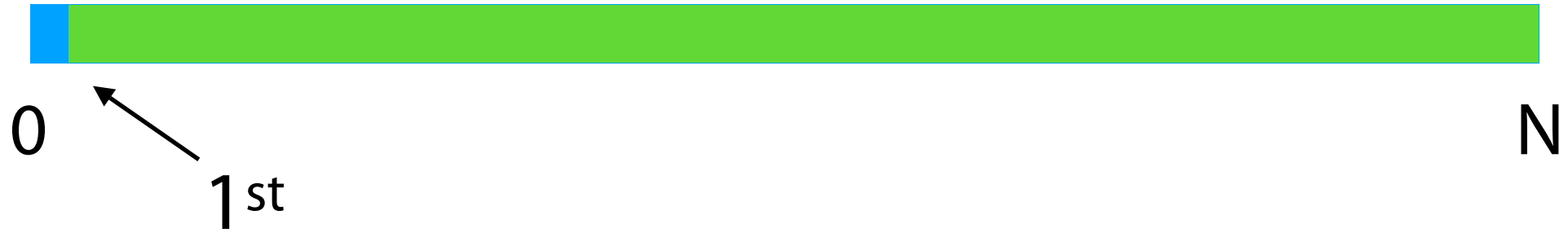
$$95 \approx 1000/(N+1)$$

$$N+1 \approx 10.5$$

$$N \approx 9.5$$

Cardinality Estimation

Does k^{th}-smallest estimate better than minimum?



Or is it "just another point" on the line?

Cardinality Estimation

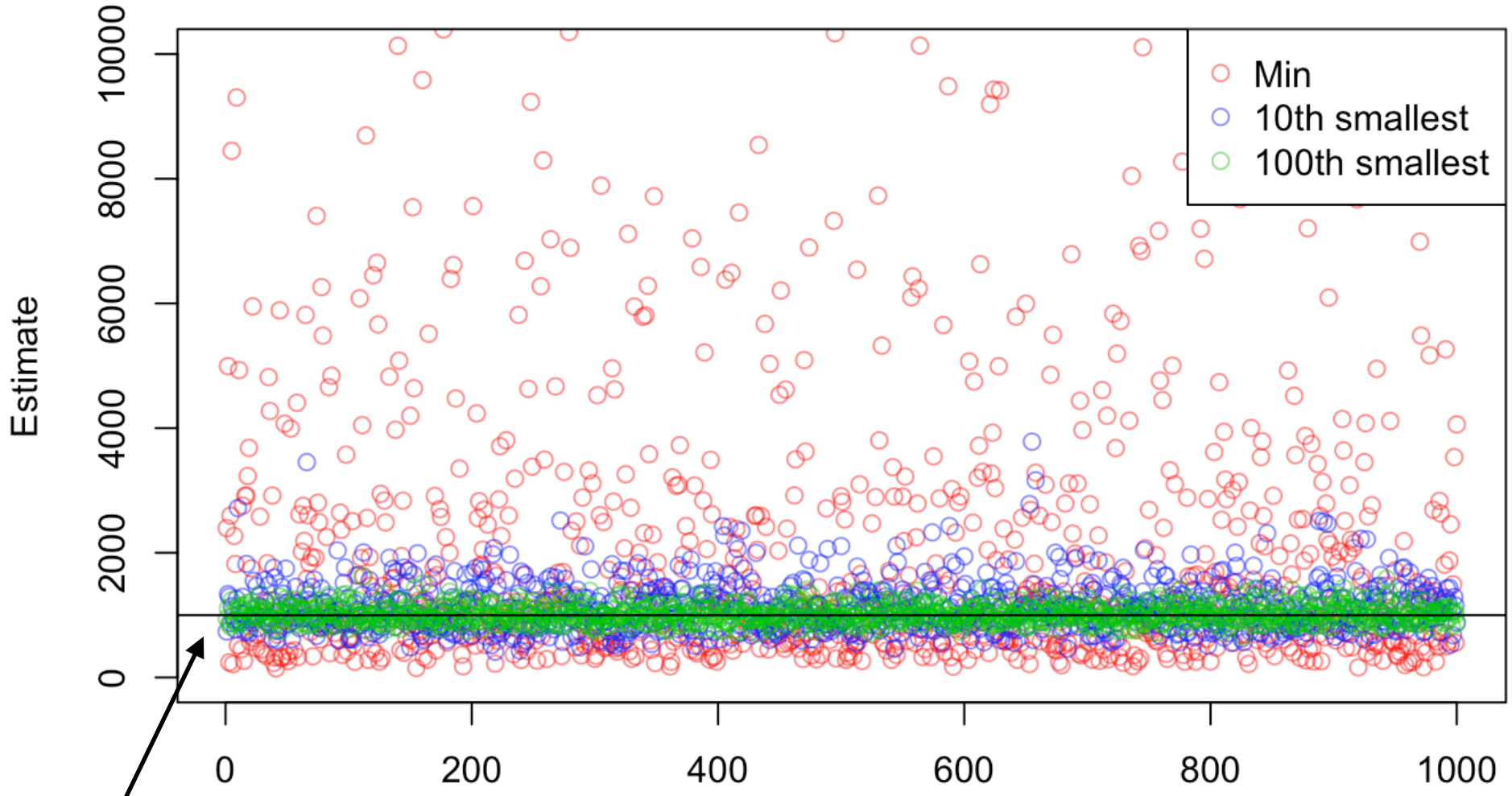
The interval from 0 to the k^{th}-smallest is the sum of the intervals between all the i^{th} smallests up to k

$$min \approx N/(n+1)$$

$$k^{\text{th}}\text{-smallest} \approx k \cdot N/(n+1)$$

K$^{\text{th}}$ Minimum Value (KMV) estimates better than minimum *via averaging*

Cardinality Estimation



True cardinality = 1,000

Cardinality Estimation

A hash function randomly "spreads" items along the range of the hash function — **any hashable dataset is a "hat problem" !**

Say h_{64} is a 64-bit hash function;

$$\frac{h_{64}(x)}{2^{64}-1}$$

spreads outputs along [0, 1] with super fine resolution

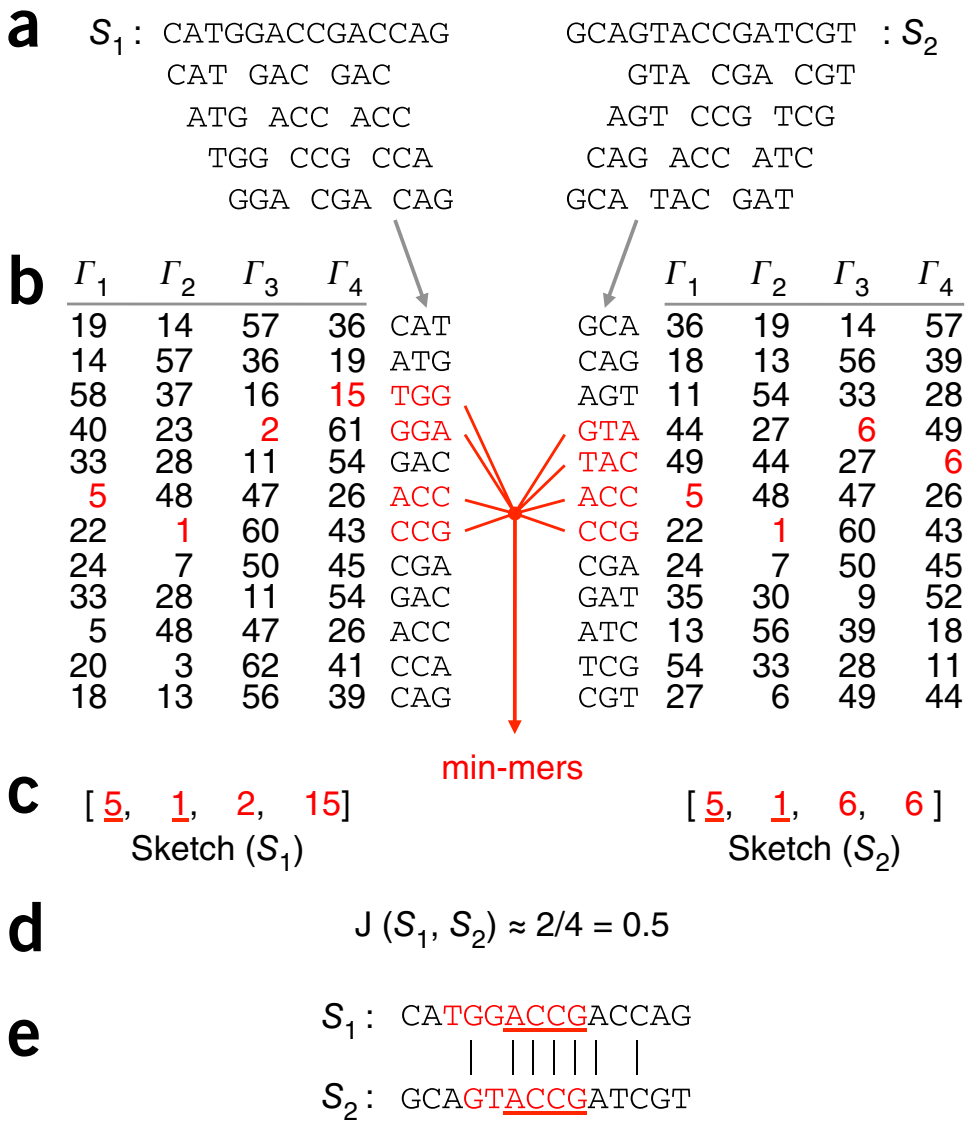
Minhash Sketch

1) Sequence decomposed into **kmers**

2) Multiple hash functions (**Γ**) map kmers to values.

3) The smallest values for each hash function is chosen

4) The Jaccard similarity can be estimated by the overlap in the **Min**imum **Hash**es (**Minhash)**



Assembling large genomes with single-molecule sequencing and locality-sensitive hashing
Berlin et al (2015) *Nature Biotechnology*