

# Text Processing / Mining / Search

Mat Kelly, PhD

Assistant Professor, Information Science  
Drexel University, College of Computing & Informatics (CCI)  
Philadelphia, PA

[mkelly@drexel.edu](mailto:mkelly@drexel.edu)  
<https://matkelly.com>  
[@machawk1](https://twitter.com/machawk1)

LIS Education And Data Science Integrated Network Group (LEADING) Bootcamp  
Week 1: June 10, 2021

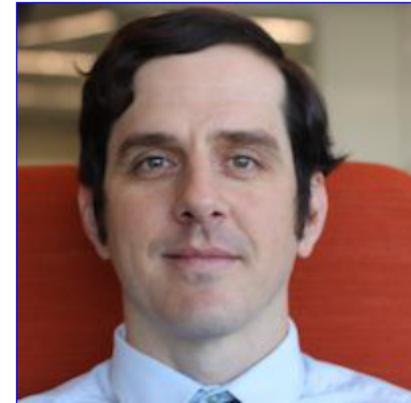
# What We Will Cover

- Text Processing
  - Normalization
  - Regular Expressions (RegEx)
  - Stop words
  - Stemming
  - n-grams
- Text Mining  incorporated into above
- Text Search
  - IR methods
  - Evaluating similarity

# Goals of this Presentation

- Introduce aforementioned topics at a high level
- Practical:
  - Process data (text)
  - Be able to identify and extract relevant information
  - Small corpus, intent is to extrapolate
- Basic Information Retrieval concepts
  - Applicable both to text corpora and other kinds.

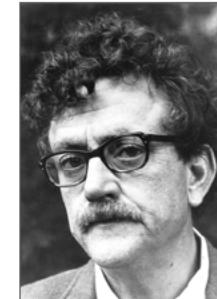
# My Background/Specialization



- Computer Science PhD, research in web archiving
- In Information Science Dept. @ Drexel CCI
- Teaching:
  - INFO624: Information Retrieval
  - INFO600: Web Systems & Architecture
  - INFO202: Data Curation
- Authored multiple Python-based tools while a grad student
  - <https://github.com/machawk1>
- Coming from a Systems Development Background (cf. data science)
  - The tools to generate, consume the data

# Sample Corpus (small collection of “Documents”)

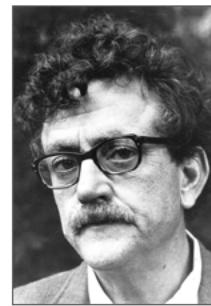
- $D_1$  : The quick brown fox jumped over the lazy dog.
- $D_2$  : The sum of the square of the legs of a right triangle are equal to the square of the hypotenuse.
- $D_3$  : We are what we pretend to be, so we must be careful about what we pretend to be.



# Preparation

- Tokenize
- Remove Stopwords
- Stem Words
- Construct N-grams

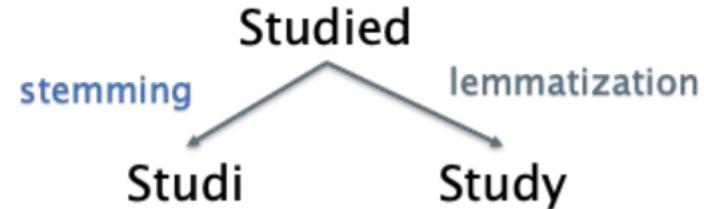
# Tokenization



- Counting instances
  - D<sub>3</sub> : We are what we pretend to be, so we must be careful about what we pretend to be.  
○ (1) We (2) are (3) what (4) we (5) pretend (6) to (7) be (8) so (9) must (10) careful (11) about
- Normalization
  - dog, dogs → dog
  - leading, leads, leader, leaders → lead
  - Simpler examples, e.g., Leading, leading, LEADING → leading
  - co-education? San Francisco? Hewlett-Packard

# Tokenization

- Tokenization, normalization, and inverted index
- Normalization: casefolding, stemming, lemmatization
  - Leading, LEADING, leading → leading
  - C.A.T., cat, Cat → cat?
  - General Motors / general motors?
- Lemmatization: reducing to canonical/dictionary form
  - am, are, is → be
  - car , cars, car's, cars' → car
  - For example:
    - the boy's cars are different colors
    - The boy car be different color



# Remove Stopwords

- The quick brown fox jumped over the lazy dog.
- The sum of the square of the legs of a right triangle are equal to the square of the hypotenuse.
- We are what we pretend to be, so we must be careful about what we pretend to be.
- Some bases/rationale for stopwords
  - Common / noisy / not semantically relevant
- To Be Or Not To Be (Hamlet)

# Remove Stopwords

- The quick brown fox jumped over the lazy dog.
- The sum of the square of the legs of a right triangle are equal to the square of the hypotenuse.
- We are what we pretend to be, so we must be careful about what we pretend to be.
- Some bases/rationale for stopwords
  - Common / noisy / not semantically relevant
- To Be Or Not To Be (Hamlet)

# Remove Stopwords

- quick brown fox jumped over lazy dog.
- sum square the legs a right triangle are equal square hypotenuse.
- We are what we pretend, so we must be careful about what we pretend.

- Some bases/rationale for stopwords
  - Common / noisy / not semantically relevant
- To Be Or Not To Be (Hamlet)



# Stem Words



- Words “leading”, “leads”, “leaders”, “leader” all have same “root”
  - We want to coalesce for semantic/syntactic analysis
  - **Porter stemmer:** fast but not very precise
  - **Snowball (Porter2):** more aggressive but generally better than Porter
  - **Lancaster:** most aggressive, least readable result

Typical **Porter** rules

- sses → ss
- ies → i
- ational → ate
- tional → tion

Weight of word sensitive to rules  
(m>1) EMENT →

- replacement → replac
- cement → cement

# Text Processing Preparation

## Stem Words

```
import nltk
from nltk.stem.porter import *

d1 = "The quick brown fox jumped over the lazy dog."
d2 = "The sum of the square of the legs of a right triangle are equal to the
square of the hypotenuse."
d3 = "We are what we pretend to be, so we must be careful about what we pretend to
be."
docs = [d1, d2, d3]

porter_stemmer = PorterStemmer()

term_list = []
# Tokenize each document
for document in docs:
    term_list.append(nltk.word_tokenize(document))

stemmed_term_list = []
# Stem all tokens in each document
for i, term_list in enumerate(term_list):
    stemmed_term_list.append([porter_stemmer.stem(word) for word in
    term_list_for_doc])

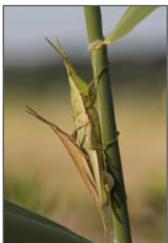
# Print out stemmed term list per document
for i, doc in enumerate(docs):
    print(f'D{i} stemmed terms: {" ".join(stemmed_term_list[i]) }')
```

% python3 (code to the right)

D0 stemmed terms: the quick brown fox jump over the lazi dog .

D1 stemmed terms: the sum of the squar of the leg of a right triangl are equal to the squar of the hypotenus .

D2 stemmed terms: we are what we pretend to be , so we must be care about what we pretend to be .

The Colab logo consists of the word "colab" in a stylized, yellow-orange font.

# Construct N-grams



- Helps to identify multi-word semantics
- Bi-grams examples:
  - the quick; quick brown; brown fox; fox jumped; jumped over; over the; the lazy; lazy dog
- Tri-gram examples:
  - the quick brown; quick brown fox; brown fox jumped; fox jumped over; etc.
- Useful for identifying compound tokens
  - Drexel University
  - data science
  - The Queen of England

```
import nltk
from nltk.tokenize import word_tokenize

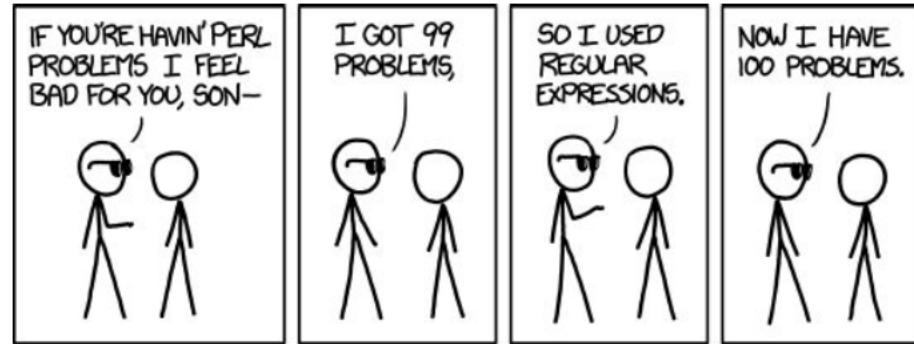
doc = "The quick brown fox jumped over the lazy dog."
tokens = nltk.word_tokenize(doc)
bigrm = nltk.bigrams(tokens)
print(*map(' '.join, bigrm), sep=', ')
```

> The quick, quick brown, brown fox, fox jumped,  
jumped over, over the, the lazy, lazy dog, dog .

colab

# Regular Expressions

- Test whether a string matches a pattern
- Useful for search/replace character string
- Commonly implemented: Perl, Python, text editors, UNIX command-line
- Powerful but confusing syntax

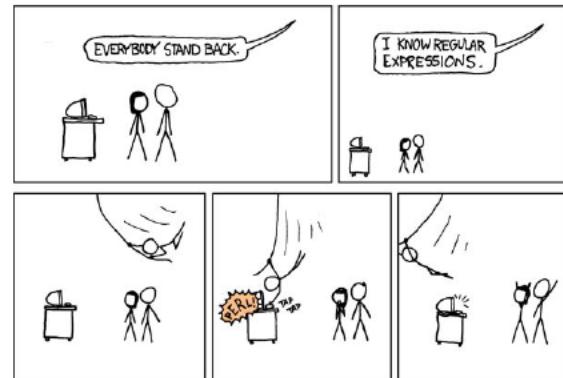


[CC-BY-NC](#) by [XKCD](#)

# Regular Expressions

- Basic usage

- Boolean: `analy(s|z)e` → matches analyze, analyse
- Zero-or-one: `colou?r` → matches color, colour
- Zero-or-more: `[a-z]*` → matches a, leading, dog, (empty string), etc. (but not LEADING)
- One-or-more: `b[oO]+` → matches bo, bOooOOOoo, bOO
- Matches  $n$  times: `b[o]{2}` → matches boo
- Matches minimally  $i$  times: `a[r]{2,}gh` → matches arrgh, arrrrrrrrrrgh
- Matches maximally  $j$  times: `b[o]{,4}` → matches boooo, boo, b
- Matches between between  $i$  and  $j$  times (inclusive): `[bo]{1,3}` → bobobo, bo



[CC-BY-NC](#) by [XKCD](#)

# How are RegEx useful?

- `re.findall('[a-z]+ of the [a-z]+', d2)`
  - Result: ['sum of the square', 'square of the hypotenuse']
- `re.findall('[a-z]{7,}', d2)`
  - Result: ['square', 'triangle', 'square', 'hypotenuse']
- Can also be compiled for reuse
  - `r = re.compile('\sthe\s[a-z]+')` # find all instance of ' the XXXX'
  - `r.findall(d2)`
  - Result: [' the square', ' the legs', ' the square', ' the hypotenuse']

Shorthands  
\s → whitespace  
\d → digits  
\w → word  
...many other

D<sub>2</sub> : The sum of the square of the legs of a right triangle are equal to the square of the hypotenuse.

colab

# Search: Evaluation

- Supposing we are searching for text in a collection, how do we measure what is a “good” result?
- An information retrieval standpoint
  - TF-IDF (term frequency - inverse document frequency)
  - Cosine Similarity
  - Precision & Recall

# TF-IDF

- Term Frequency ( $tf_{t,d}$ ): The number of times term  $t$  appears in document  $d$
- A document with **10** occurrences of the term is *more relevant* than a document with 1 occurrence of the term.
  - But not **10 times** more relevant.
  - We use logarithms to mitigate this effect
- $tf = \frac{\text{term count}}{\# \text{ of terms in document}}$



Karen Spärck Jones

By University of  
Cambridge  
[CC-BY-SA](#)

## Search: Evaluation

# TF-IDF

- Document Frequency (df)
  - Rare terms more informative than common terms (e.g., stopwords)
  - Give rare terms high weights
- $df = \frac{\text{# documents with term in it}}{\text{# of documents}}$
- Inverse df =  $1/df = \frac{\text{# documents}}{\text{# documents w/ term}}$

higher weights

lower weights

rare terms  
(small DF)

frequent terms  
(small DF)



Karen Spärck Jones

By University of  
Cambridge  
[CC-BY-SA](#)

# TF-IDF

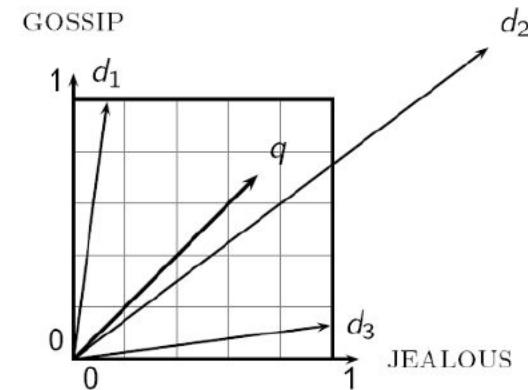
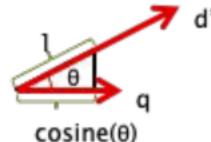
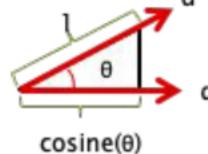
- TF-IDF: weight of each term in a document to signify importance
- $\text{tf.idf} = \log(\text{tf} \times \text{idf})$
- Ranking of a document for a query (which is most relevant):

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

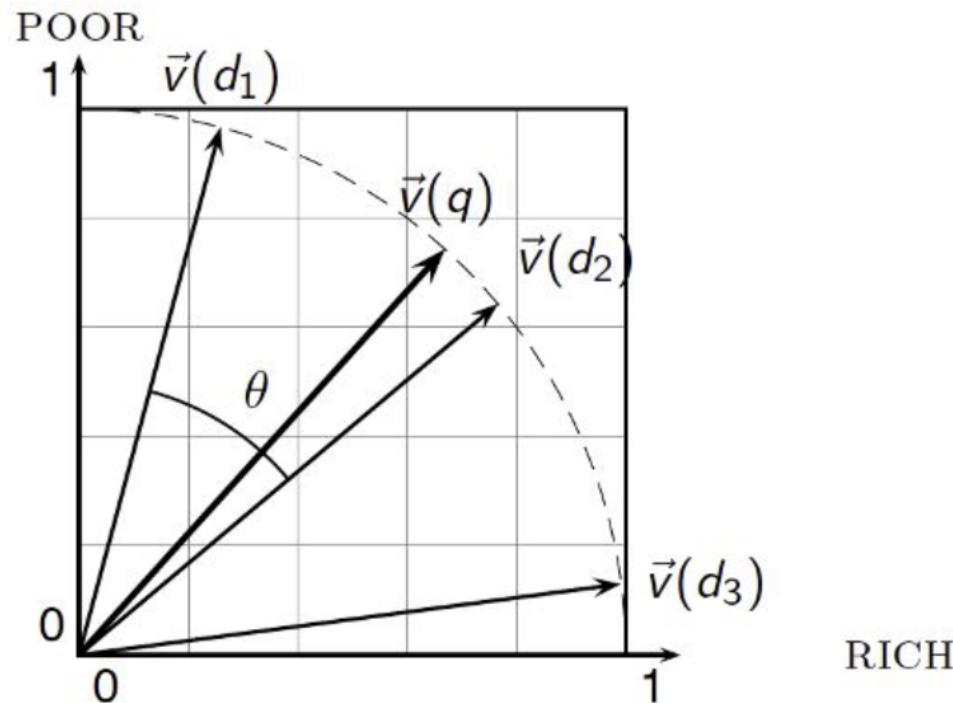
# Cosine Similarity



- Plot term vectors, vector length is affected by TF
  - e.g., document with just “be be be be be be”
- Distance between query and document is problematic
- Rather than vector magnitude, use angle between
  - Smaller angle → more similar documents
- To measure how close/similar they are:
  - The Euclidean distance between the two documents can be quite large
  - The angle between the two documents is 0, corresponding to maximal similarity. (Cosine = 1)
- Key idea: Rank documents according to angle with query.



# Cosine Similarity Illustrated



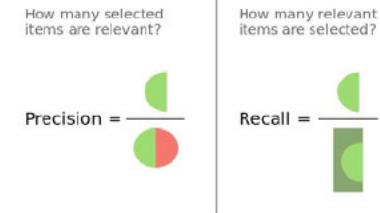
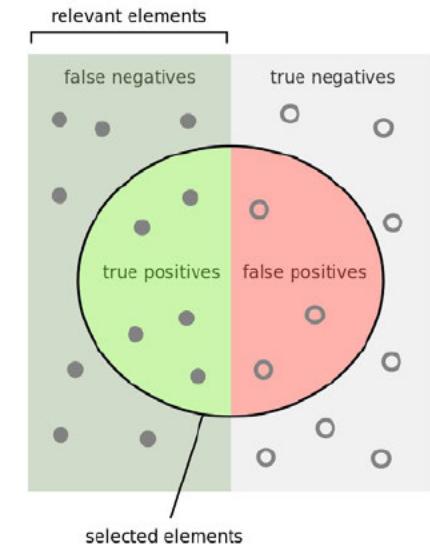
# Vector Space Ranking

Using TF-IDF and Cosine Similarity together

- Represent the query as weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return top K (e.g., K=10) to the user

# Precision & Recall

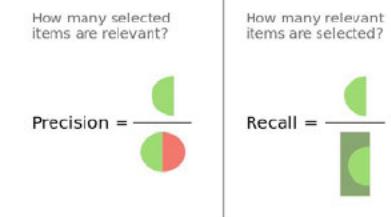
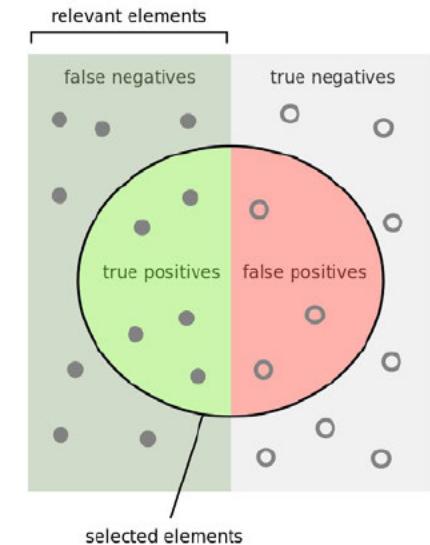
- **True Positive**: Things we got and should have gotten
- **False Positive**: Things we got and should not have gotten
- **False Negative**: Things we did not get and should have gotten
- **True Negative**: Things we did not get and should not have gotten (true negative)



[CC-BY-SA](#) by Walbur

# Precision & Recall

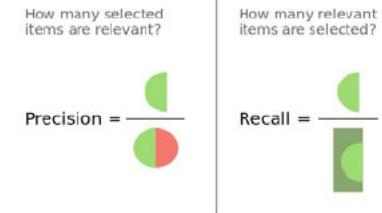
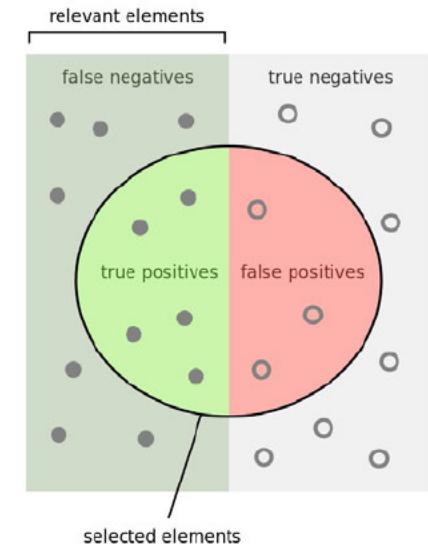
- Precision & Recall are a means of evaluating retrieval qual of 0.0 to 1.0)
  - What this entails may be domain-specific
  - Generally are inversely proportional
- **Precision:** # True positives over # results returned:
  - $TP/(TP+FP)$
- **Recall:** True positives over all positives
  - $TP/(TP+FN)$
- Neither metric is sufficient alone, can be gamed
  - High precision/ low recall: very few, but correct result (**P**: 1.0)
  - Low precision / high recall: return everything (**R**:1.0)



[CC-BY-SA](#) by Walbur

# Precision & Recall

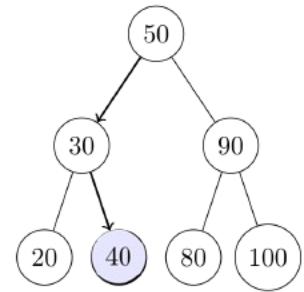
- Key is to find best balance between **Precision/Recall**
- F-Measure:
  - $$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$
- Measures *accuracy* of the retrieval/classification method
- Range from 0.0 (e.g., recall=1.0) to 1.0 (balance **P/R**)



[CC-BY-SA](#) by Walbur

# Time/Space Complexity

- Big-O notation
  - Search algorithms
  - Sorting algorithms
- Large data sets require efficient processing
- Sorting preferred, makes searching more efficient

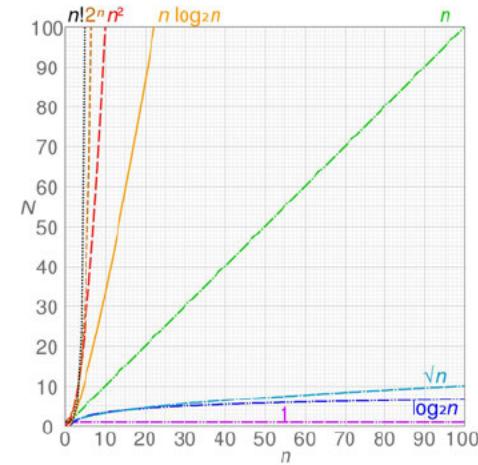
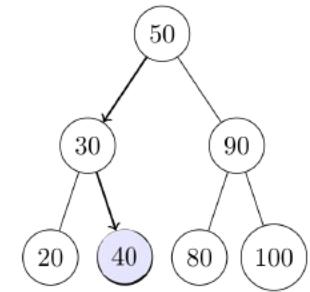


[CC BY-SA](#) Cmglee

# Time/Space Complexity

- Big-O notation
  - Search algorithms
  - Sorting algorithms
- Large data sets require efficient processing
- Sorting preferred, makes searching more efficient
- **O(n)** → linear → One pass, max look through all

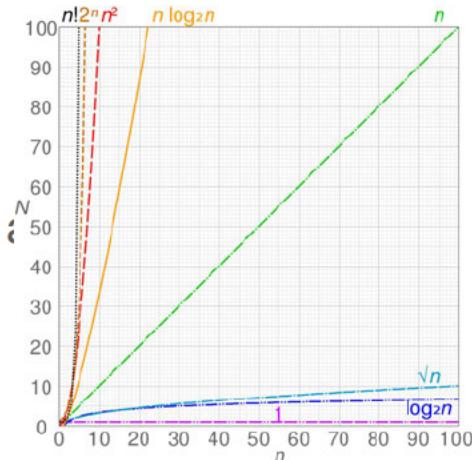
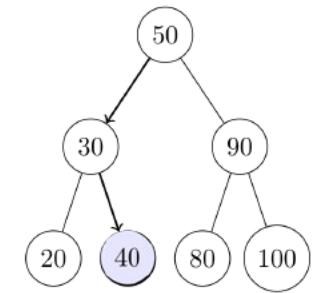
Finding an item in an ordered list



[CC BY-SA Cmglee](#)

# Time/Space Complexity

- Big-O notation
  - Search algorithms
  - Sorting algorithms
- Large data sets require efficient processing
- Sorting preferred, makes searching more efficient
- $O(n)$  → linear → One pass, max look through all
- **$O(\log n)$**  → log → More efficient than linear, requires divide & conquer

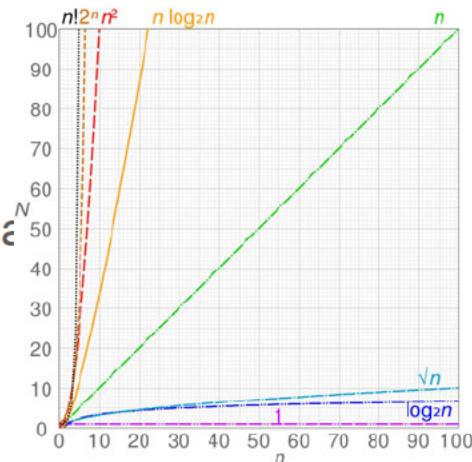
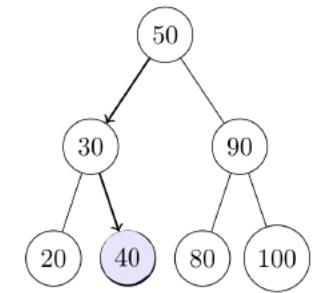


Finding an item a balanced tree structure

[CC BY-SA](#) Cmglee

# Time/Space Complexity

- Big-O notation
  - Search algorithms
  - Sorting algorithms
- Large data sets require efficient processing
- Sorting preferred, makes searching more efficient
- $O(n)$  → linear → One pass, max look through all
- $O(\log n)$  → log → More efficient than linear, requires divide & conquer
- **$O(n^k)$**  → polynomial → Very inefficient, e.g., nested loops



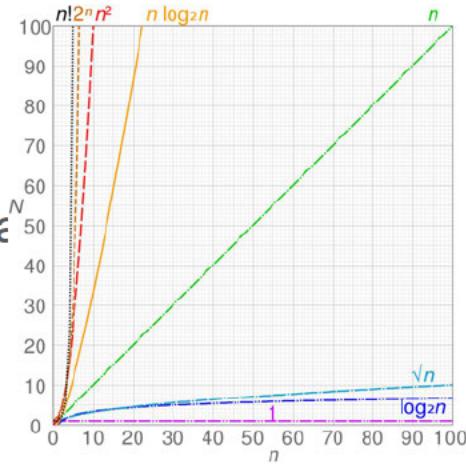
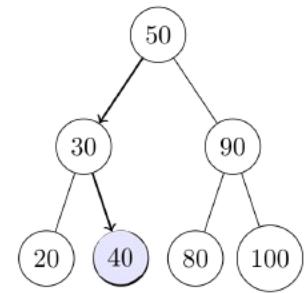
## Nested loops

[CC BY-SA](#) Cmglee

# Time/Space Complexity

- Big-O notation
  - Search algorithms
  - Sorting algorithms
- Large data sets require efficient processing
- Sorting preferred, makes searching more efficient
- $O(n)$  → linear → One pass, max look through all
- $O(\log n)$  → log → More efficient than linear, requires divide & conquer
- $O(n^k)$  → polynomial → Very inefficient, e.g., nested loops
- **$O(n \log n)$**  →  $n \cdot \log n$  → More efficient than polynomial, less than linear

Various sorting algorithms: Heap, Merge, Quick



CC BY-SA Cmglee

# Parallelization

- Means of optimizing execution
- Multiprocessing
  - Execute tasks simultaneously using multiple processors.
  - `multiprocessing.Pool` and `multiprocessing.Process`
- Asynchronous vs. Synchronous execution
  - Sync: retains order, uses locking
  - Async: mixed order, but may finish quicker



# Parallelization

- Identifying bottlenecks, programmatic dependencies
  - e.g., corpus metrics cannot be calculated until processing completed on all documents
- Works well for processes that can be chunked or sub-processes executed independently
  - e.g., classification, numerical calculation, relevant text extraction

# In closing, topics covered

- Normalization
- Regular expressions
- Tokenization
- Stopwords
- Stemming
- N-grams
- TF-IDF
- Cosine Similarity
- Precision/Recall
- Complexity
- Parallelization