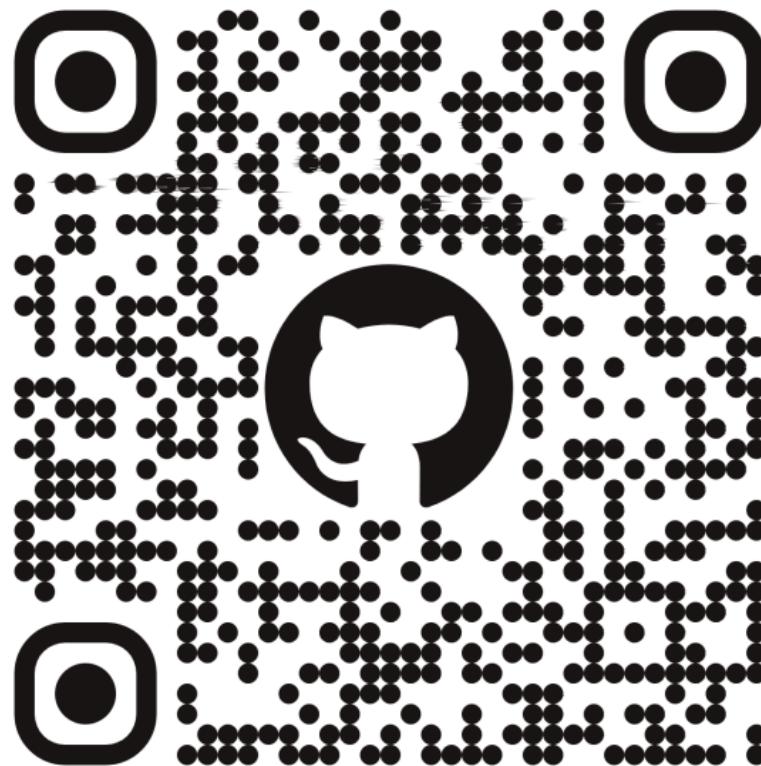


Putting the FUN back in Fundamentals:

Data Structures, Algorithms, and More!

July 19, 2024

Download the PDF

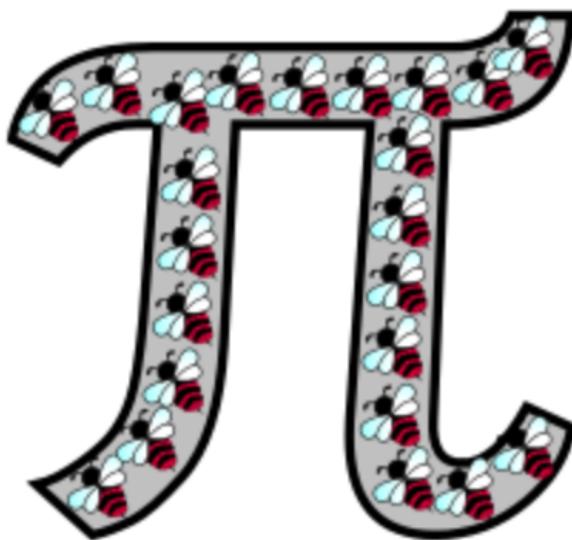


Matt “Kelly” Williams



Matt “Kelly” Williams

Background aka Why am I Up Here?



- Raspberry Pi Addict
- Search for “real” work for Pis
- In the process I discovered lots of new and nifty algorithms

Disclaimers

1. I am not a Mathematician
2. I am not a Chemist (this will make sense later)
3. I am not an expert
4. I have become passionate about this topic

One Data Structure to Rule them All

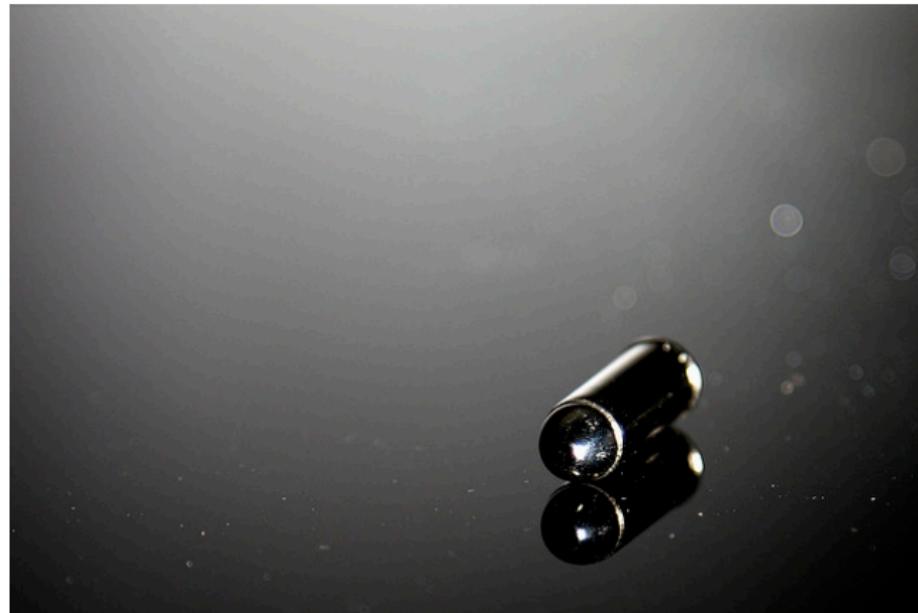
Hint:

It is used by every program in every digital computer

Good Algorithms Matter

The real reason behind choosing good algorithms to solve a problem is that the good ones either break the problem into smaller problems or are myopic; focusing only on the steps needed to reach an answer. All else is superfluous.

No Silver Bullet



flickr photo shared by [eschipul](#) under a [Creative Commons \(BY-SA \)](#) license

When all you have is a hammer.....



Real World™ Considerations

Each algorithm has its “sweet spot”, but the considerations tend to be among the following:

- Space
- Runtime
- Data Access



Know thy data — “Not Socrates”



Like sand through the hourglass, so is the data of our lives — Also “Not Socrates”

The Big O

The Big $O()$

- $O()$ refers to the complexity of an algorithm.
- You can also think of it in terms of the number of cycles/comparisons/steps to complete its task.

$O(n)$

Linear Time

Size	Cycles
1	1
2	2
10	10
100	100
1000	1000

$O(n^2)$

Quadratic Time

Size	Cycles
1	1
2	4
10	100
100	10000
1000	1000000

$O(n \log(n))$

Size	Cycles
1	0
2	1
10	23
100	460
1000	6907

$O(\log(n))$

Logarithmic Time

Size	Cycles
1	0
2	1
10	3
100	7
1000	10

$O(\log(i))$

Logarithmic Time

Size	Cycles
1	1
2	1
10	1
100	2
1000	3

$O(\log(\log(n)))$

“Logalog”



$O(1)$

Constant Time

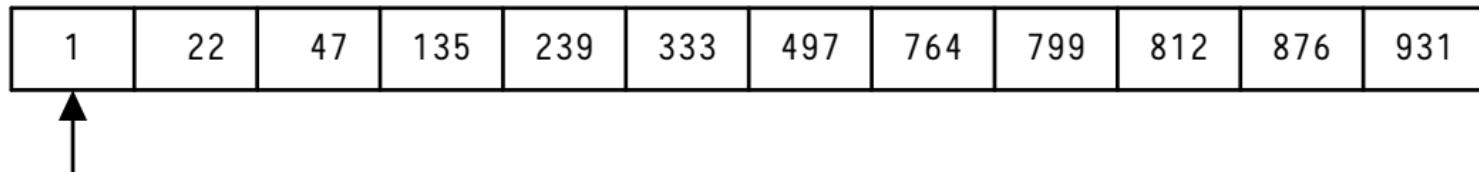
Size	Cycles
1	1
2	1
10	1
100	1
1000	1

Searches

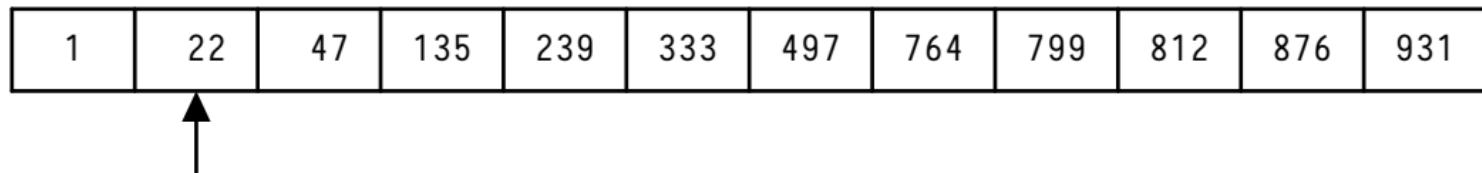
Looking for 812...

1	22	47	135	239	333	497	764	799	812	876	931
---	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----

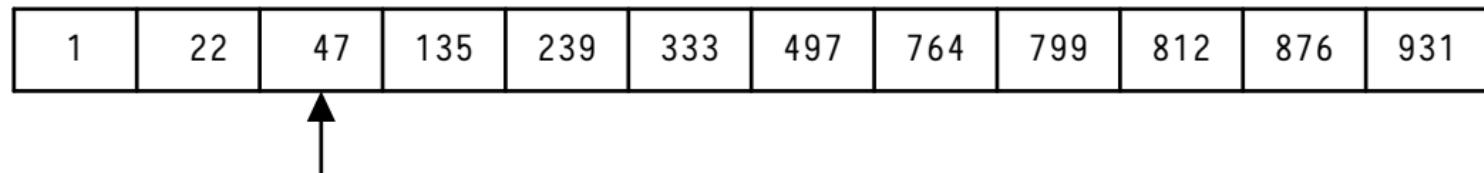
Linear



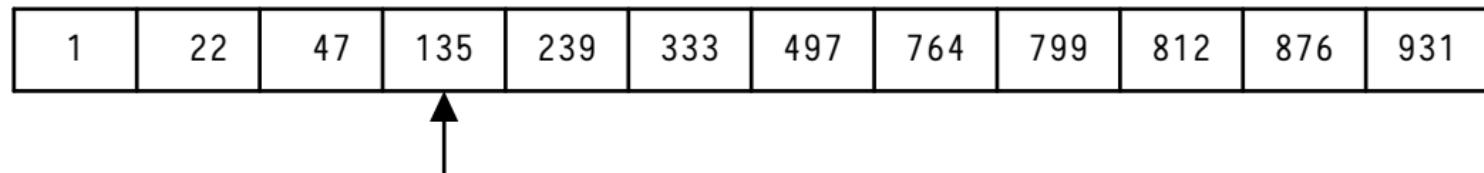
Linear



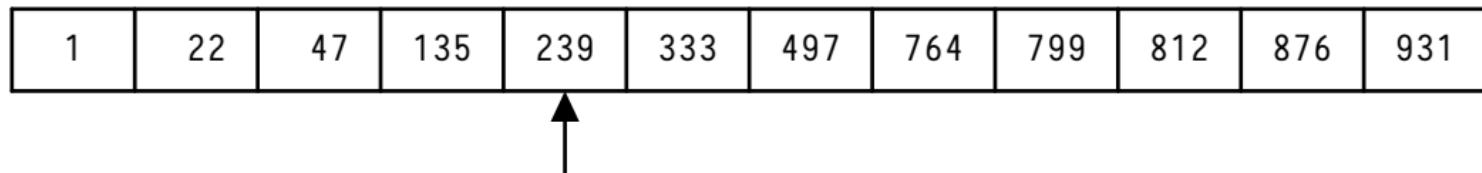
Linear



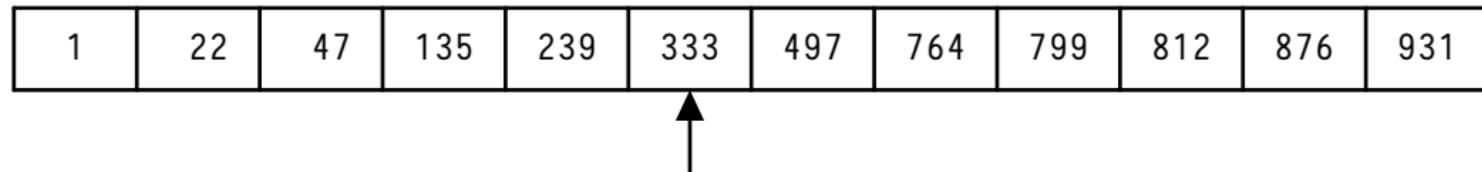
Linear



Linear



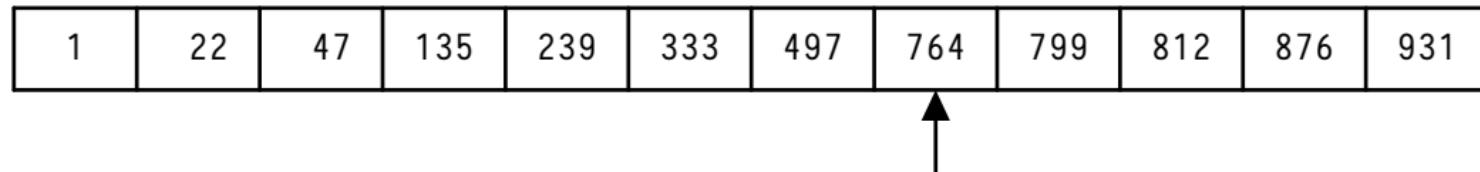
Linear



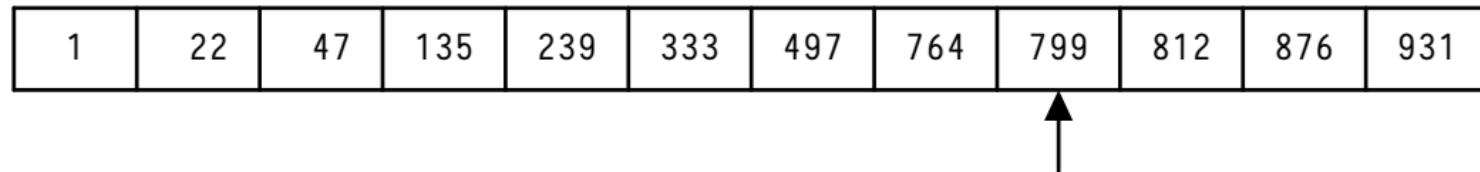
Linear



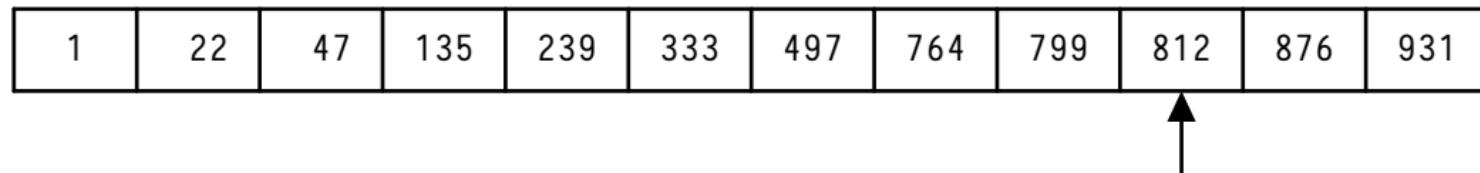
Linear



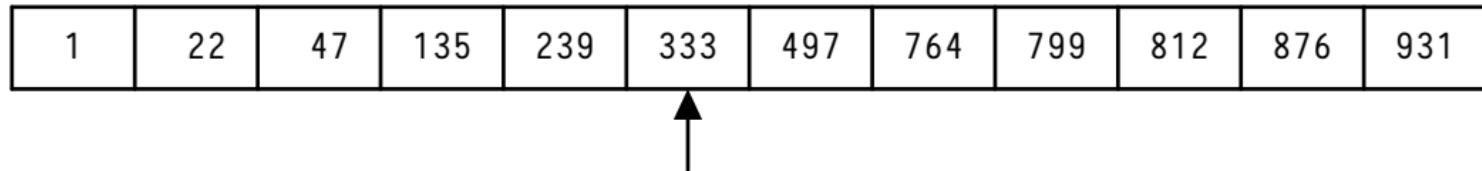
Linear



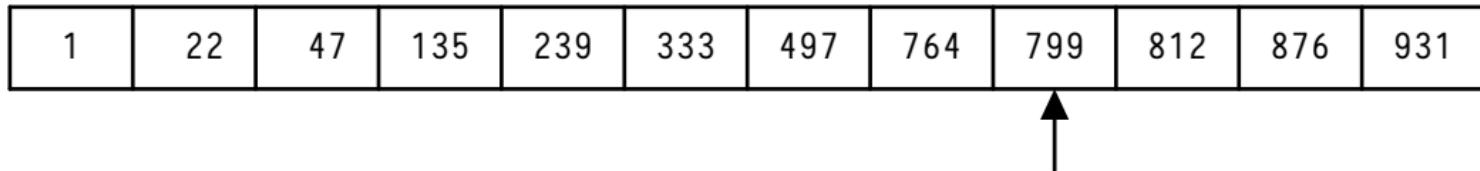
Linear



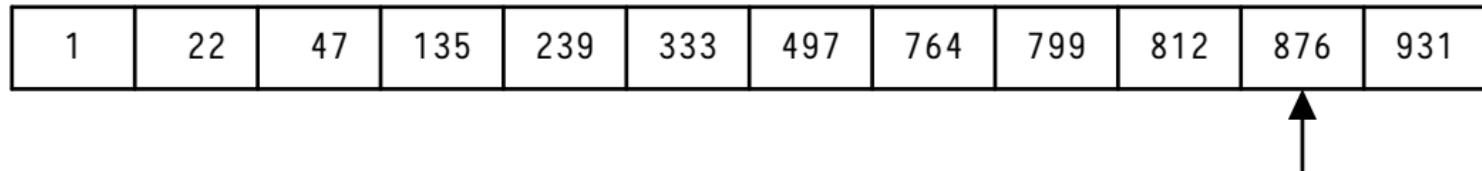
Binary



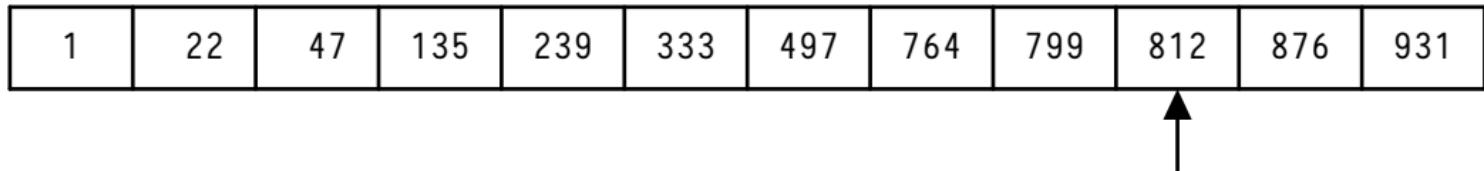
Binary



Binary



Binary

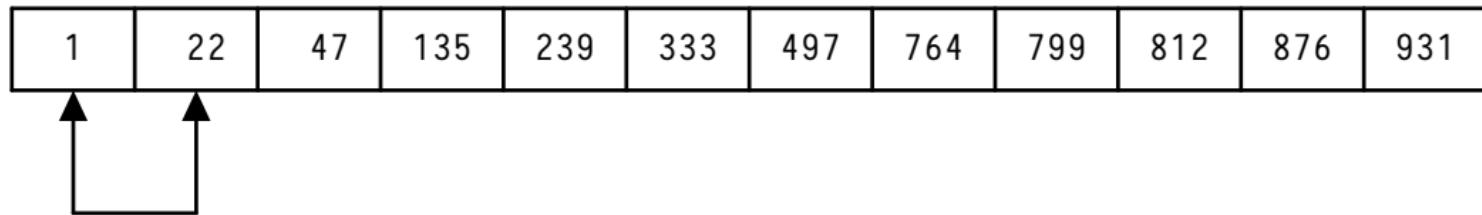


Galloping Search

- First published in 1976!!!!
- Sliding Window
- Good for cases where
- The target is at beginning or end
- The bounds of the search space are not closed (like log file)
- Approaches $O(\log(i))$

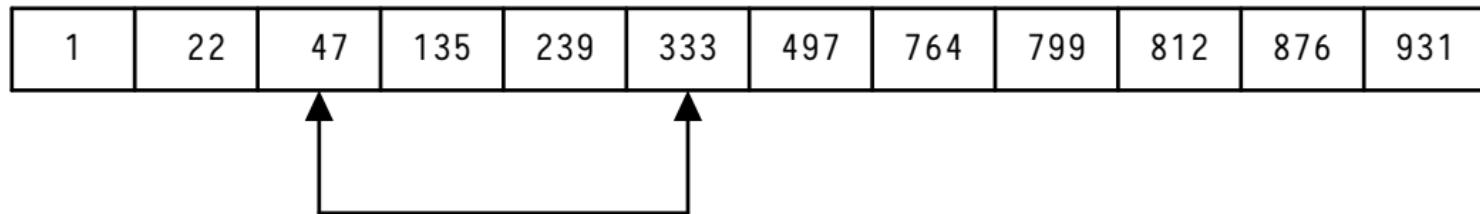
Galloping Search

First window is 2^1 long.



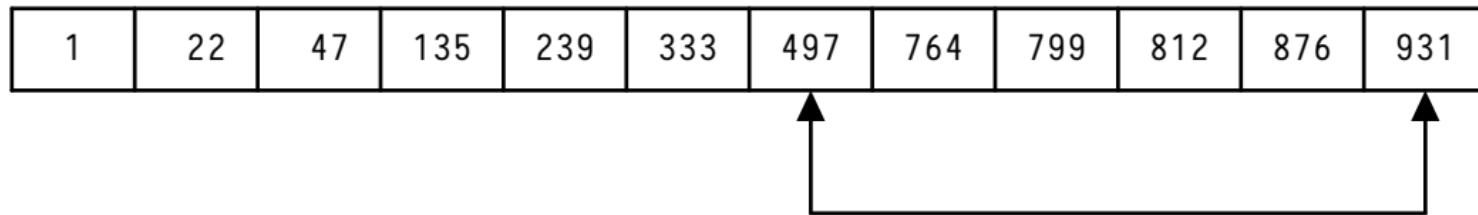
Galloping Search

Second window is 2^2 long.



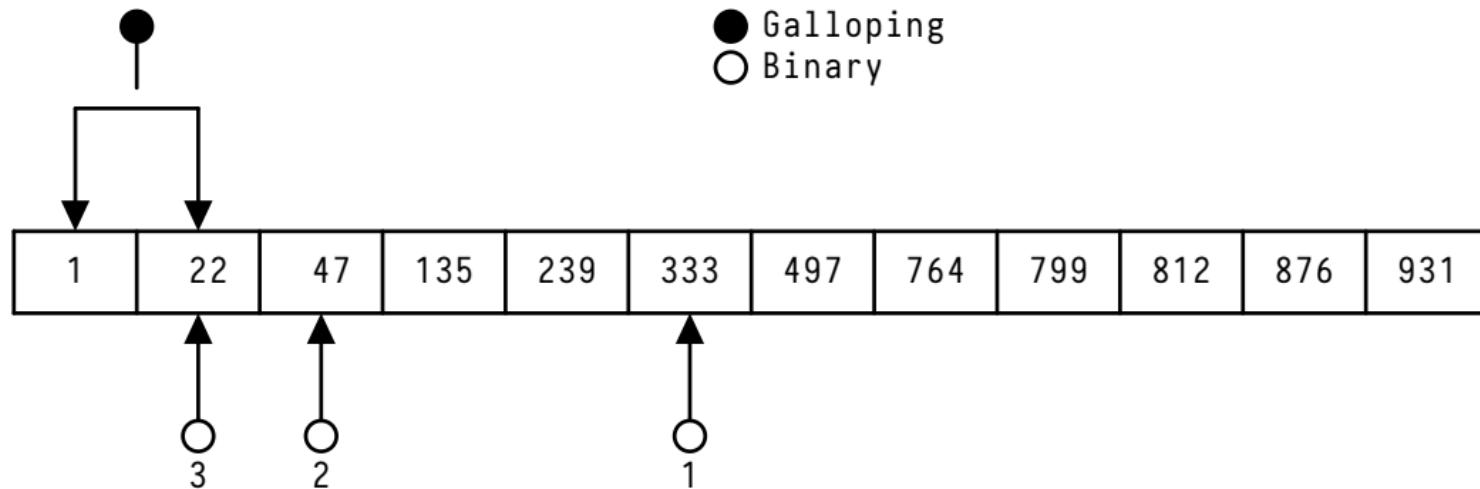
Galloping Search

Third window is 2^3 (or to the end of the array) wide. When the proper window is found, then do a binary search on the window.



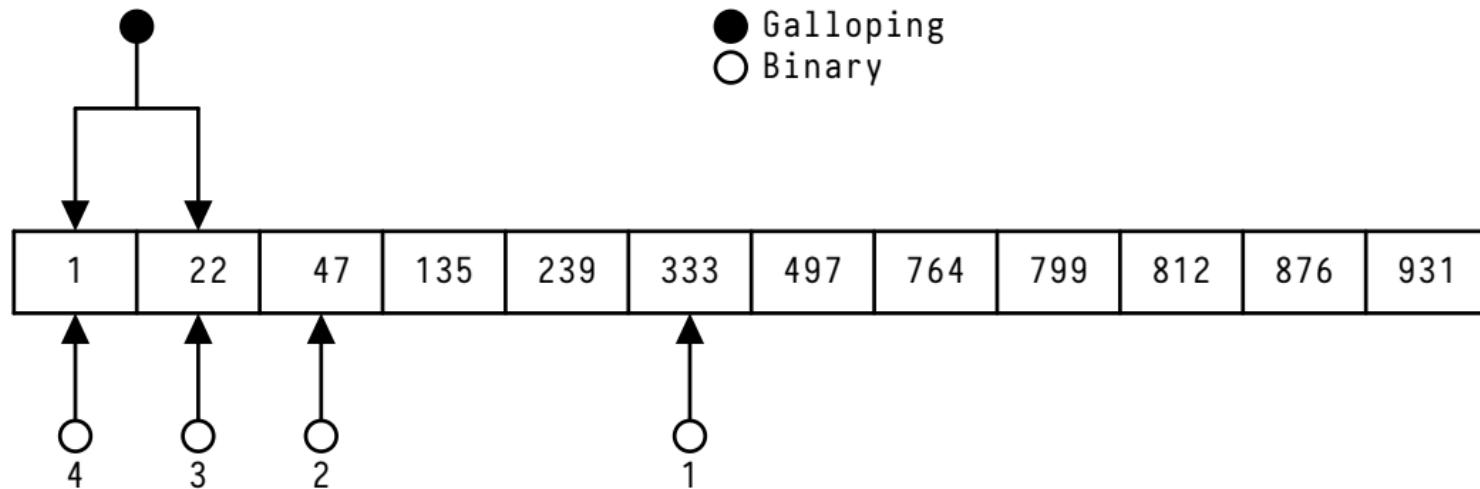
Where it Beats Binary Search

- If 22 is the target, binary search takes 3 checks. Galloping search finds it on a window boundary.



Where it Beats Binary Search

- If looking for 13, then binary search takes 4 checks to find it isn't a match. Galloping finds it isn't in the array immediately.



Variants on Galloping Search

- Start at other end
 - Trot — step grows linearly (1,2,3,...) so that the indexes are (1,3,6,10,15,...). Once the value is overshot, then hone in with binary search.
- $O(\sqrt{n})$

Interpolation Search



Interpolation Search

- Similar to how we look up a word in a dictionary.
- Instead of splitting in $1/2$ each time, make a guess where it should be in range of values
- Assumes an uniform distribution of points
- Approaches $O(\log(\log(n)))$

Interpolation Search

Predicted Location

$$(seeking / max) / size$$

Interpolation Search

First check point: $(812/931) * 12 = 10$

1	22	47	135	239	333	497	764	799	812	876	931
									↑		

Search Summary

Searching for 812

Algorithm	# of Iterations
Linear	10
Binary	4
Galloping	4
Interpolation	1

Data Structures

Data Structures not Appearing Today

- Queues
- (Americans Don't Know How to Queue)
- Stacks
- (These are Queues, too)

Arrays

1	22	47	135	239	333	497	764	799	812	876	931
---	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----

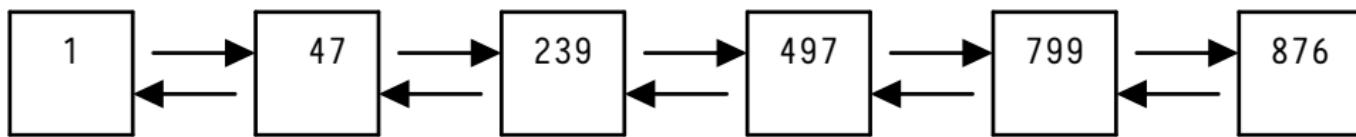
Linked Lists



Linked Lists

- Inserts tend to be easier than an array
- Appending is more expensive than an array
- Traversal is linear; $O(n)$
- “One Way Street”

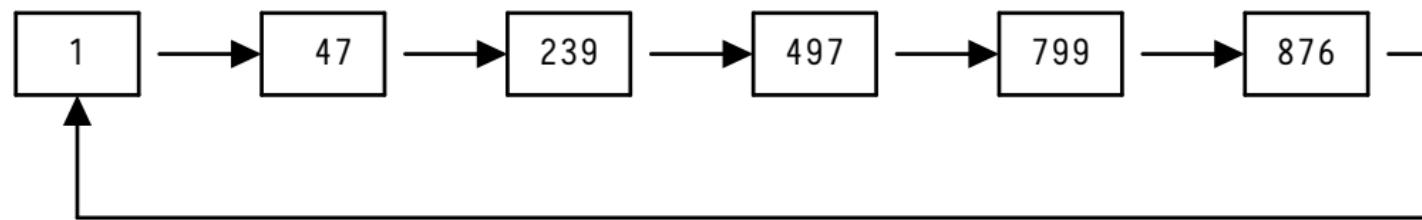
Doubly Linked Lists



Doubly Linked Lists

- Inserts are less complex than an array, but more complex than Linked List
- Appending is more complex
- Traversal is linear; $O(n)$
- “Two Way Street”

Ring Lists



Ring Lists

- Often implemented as a fixed size array; if so they are generally used as buffers
- “Now Serving....”

Trees

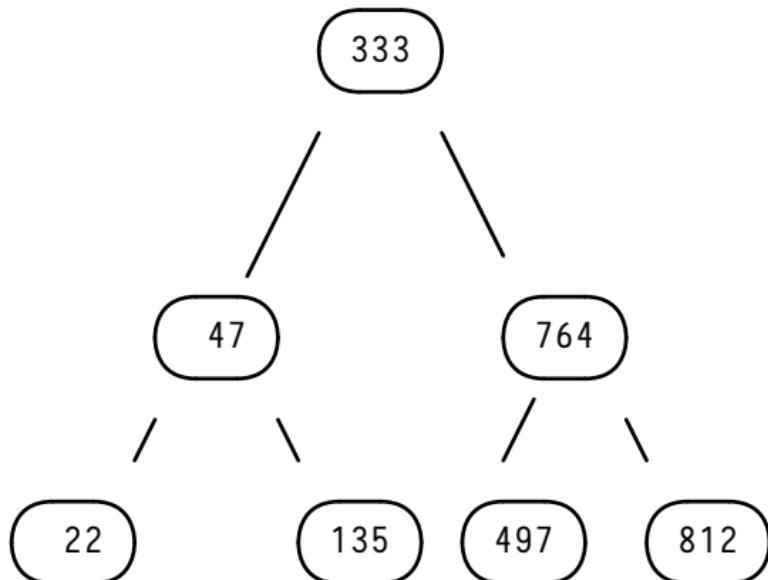
“ Programs are made by fools like me,

But only God can make a tree.

— Not Joyce Kilmer

”

Trees



Mini Quiz Break

> “What the (...) is this monstrosity? And why the (...) did he name it corned__beef?” >> “Jerry says the name is probably some kind of rotten pun. What does it do?” >> “Basically it takes the value of the characters of a demon's name, > multiplies them by a number, adds another number and then divides > the result by 65,353. Then it uses that result as a subscript in > some kind of an array.” She shook her head again. “Why 65,353? > (...)! You know, if this guy doesn't come back we may never > understand some of this stuff.” > [The Wizardry Compiled by Rick Cook](#)

Hash plural hashes

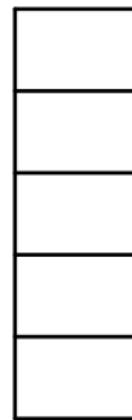
1. Food, especially meat and potatoes, chopped and mixed together.
2. A confused mess.
3. The # symbol (octothorpe, pound).
4. (computing) The result generated by a hash function.



Hash

- Used for sparsely populated universes
- Quick lookup of values
- Often implemented with a space the size of a (largeish) prime. 65353 is a large prime near the maximum size of a 16 bit unsigned Integer.

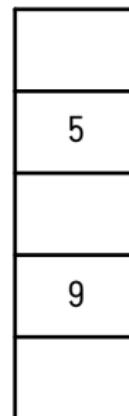
Hash



Hash

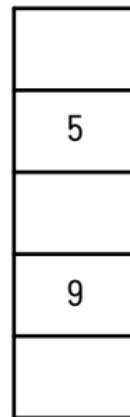
Hashing function maps entity to limited space in hash

$$(CRC(entity) \bmod ARRAYLENGTH)$$



Hash Collision

Try to insert 23; but it collides with 5.



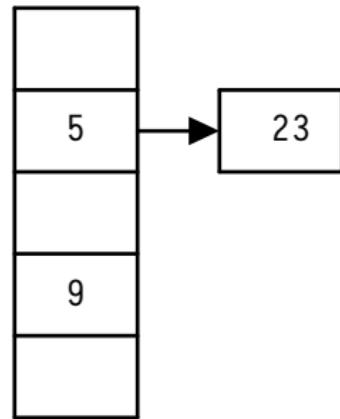


flickr photo shared by [Dino Kuznik](#) under a Creative Commons (BY) license

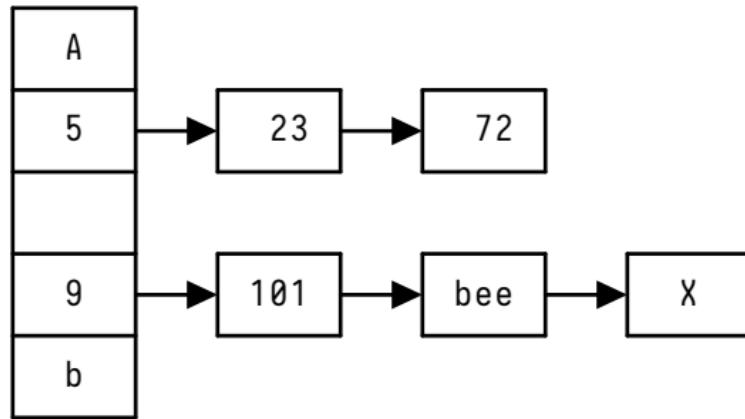
Handling Collisions

- Rehash with larger hash table
- Buckets

Hash: Bucket Hash



Hash: Bucket Hash





**MY BUKKIT IS
FULL!**

memegenerator.net

Cuckoo Hash



A common cuckoo chick pushes one of its host's eggs out of the nest. Detail of figure 1 from [Anderson et al. \(2009\)](#)

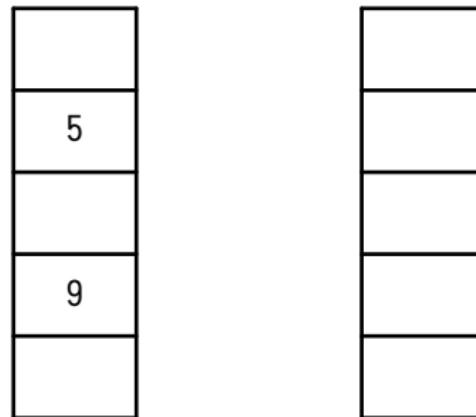
Cuckoo Hash



A common cuckoo chick pushes one of its host's eggs out of the nest. Detail of figure 1 from [Anderson et al. \(2009\)](#)

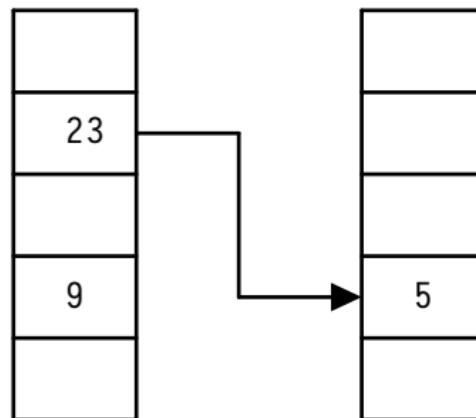
Cuckoo Hash

Trying to insert 23 again, it collides with 5

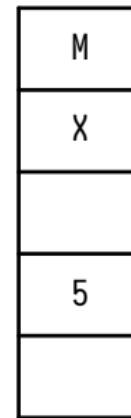
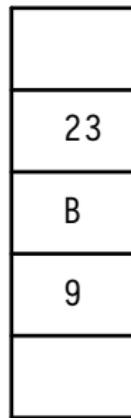


Evict 5

Each hash has a separate hashing function

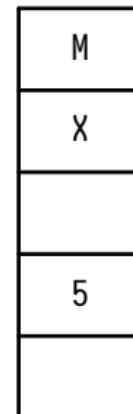
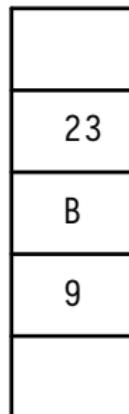


Time Passes

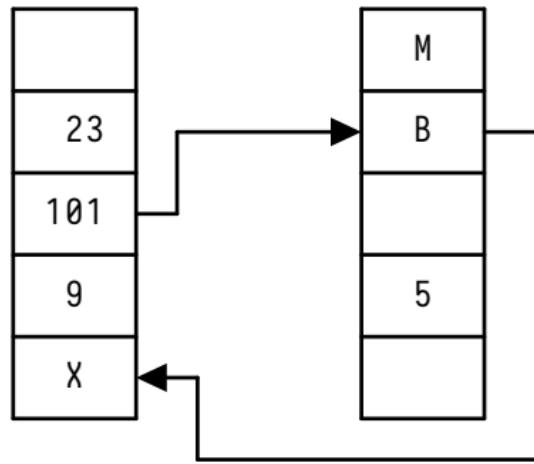


Multiple Evictions

Adding 101, collides with B & B collides with X



Multiple Evictions



Advantages of Cuckoo Hash

- Approaches $O(1)$ for inserts
- $O(1)$ for lookups

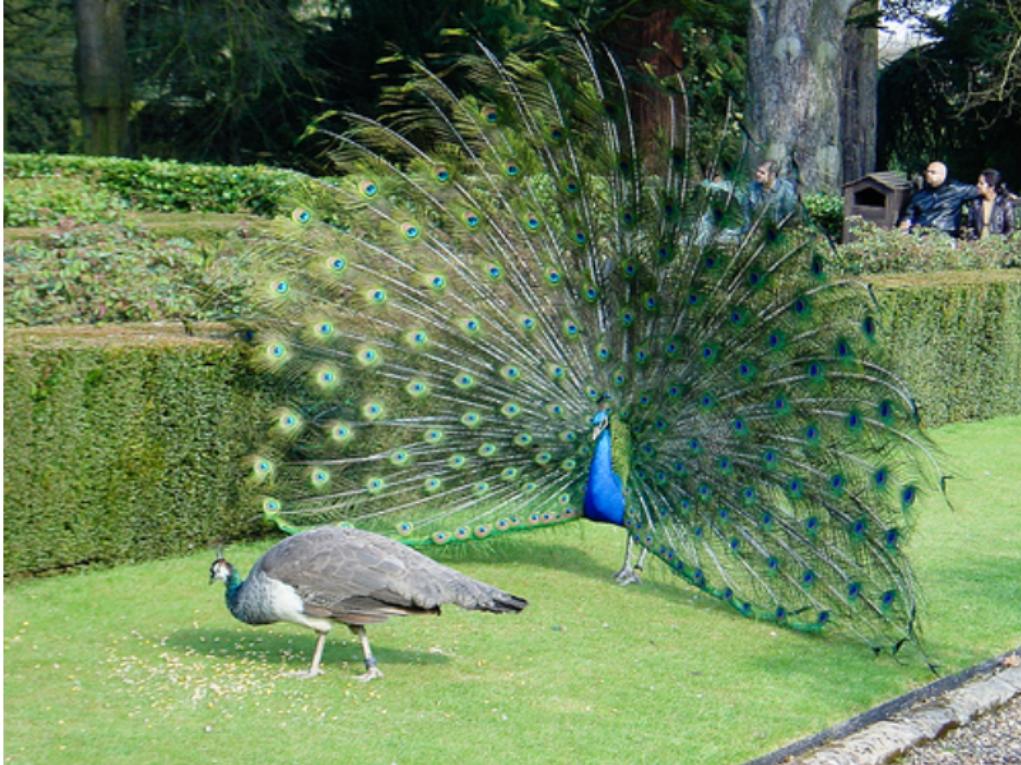
Limitations of Cuckoo Hash

- Needs a certain amount of empty space; if not an eviction cycle can occur
- May need to resize & rehash to keep evictions from looping



Obi-Wan: “That Hash is our only hope.”

Yoda: “No.... there is another.”

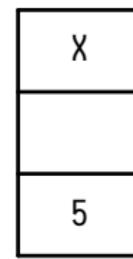
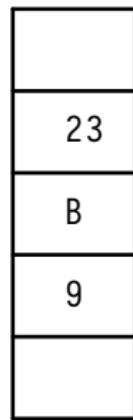


flickr photo shared by [ToastyKen](#) under a [Creative Commons \(BY \)](#) license

Peacock Hash

- Handles collisions by adding extra smaller hashes
- Originally devised for network routers to store IP/Mac Addresses
- Realtime, constant time

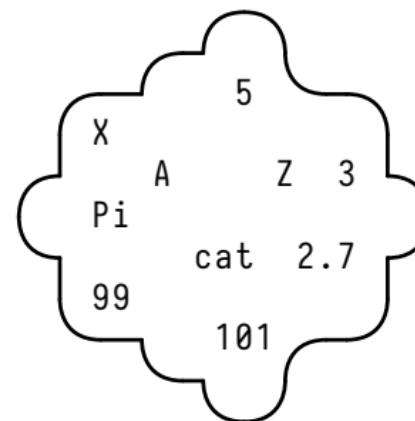
Peacock Hash



Sets

Unordered Set

- Can be implemented as a list or as a hash
- Inserts are cheap; retrievals may not be





THERE CAN BE

ONLY ONE

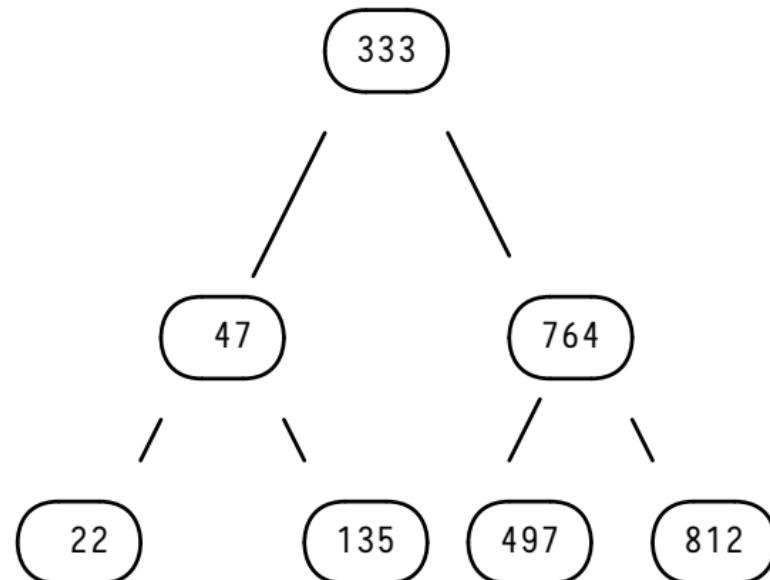
PICTUREQUOTES.COM

Ordered Sets

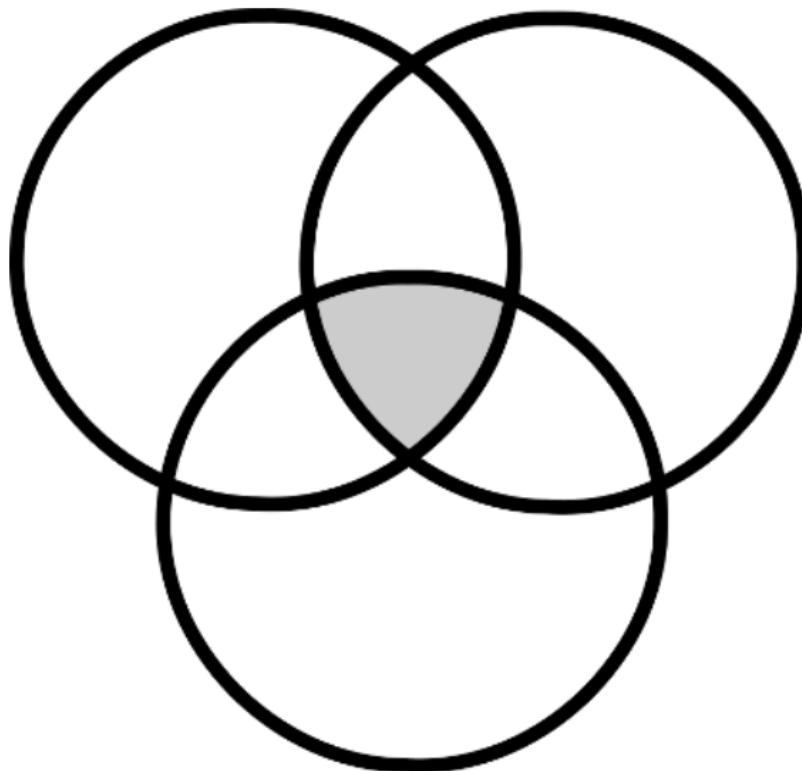
- Elements stored sorted; inserts can be expensive

1	22	47	135	239	333	497	764	799	812	876	931
---	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Stored as Tree



Set Intersection



Set Intersection

- Surveyed implementations are $O(n)$ for intersection
- Ruby
- Java
- Go
- Python

---- **THINK !**
*is there a better
way to do it ?*



Plant Efficiency starts with your ability

Galloping Intersection

Comparisons of Set Intersections

Excerpted from [Faster Adaptive Set Intersections for Text Searching](#)

Algorithm	# of comparisons
Sequential	119479075
Adaptive	83326341
Small Adaptive	68706234
Interpolation Sequential	55275738
Interpolation Adaptive	58558408

Comparisons of Set Intersections

Excerpted from [Faster Adaptive Set Intersections for Text Searching](#)

Algorithm	# of comparisons
Sequential	119479075
Interpolation Small Adaptive	44525318
Extrapolation Small Adaptive	50018852
Extrapolate Many Small Adaptive	44087712
Extrapolate Ahead Small Adaptive	43930174

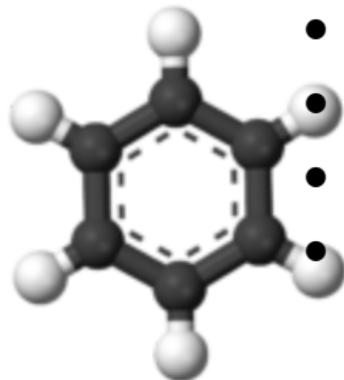
Case Study

- Search for Real Work™ for my Pi Cluster
- Pi 2B
 - 4 Cores / 900 Mhz
 - 1G Ram
 - I/O on USB 2 Bus (480 Gb/s)

Case Study

- Substructure Search
- PubChem from NIH
 - 150GB compressed download
- Uncompressed, 25K substances is 120MB; whole ~3TB
 - ~69 Million substances

Smiles



- Text Description of Chemical Structures
- C1CCCCC1 is a benzene ring
- Typically Hydrogen is left out
- *Can be more than one way to describe the same structure*

"Benzene-aromatic-3D-balls" by Benjah-bmm27 - Own work. Licensed under Public Domain via Commons.

Naive Solution

SMILES is text. What do we use to search text?

NEW!

GREP FIGURES



WITH REAL GREP ACTION!

Naive Solution

- 5 Pi 2B, searching a subset of data
 - ID, Smiles, Name
- Docker Swarm to distribute workload

Naive Solution

Structure	Results	Time
C1CCCCC1=0	~2500	5s
C1CCCCC10	~17500	15s

Fingerprints

- Pubchem has a defined bitmap to describe a molecule
 - 880 bits wide
 - Examples:
 - >2 Carbon
 - >4 Carbon
 - Boron
 - A ring

Lightbulb

- SMILES can be converted to Fingerprint with Chemistry Dev Kit
- If I create sets of substances which have a particular bit set, I can take the intersections of those sets to find the substances which match the fingerprint.

Sets

- 810 sets, one for each bit in the bitmap
- One file per bit in bitmap containing a list of substances which have the bit set.
- Using unsigned 32bit ints, the sets are anywhere from 4 bytes to 225MB in size.
 - Median 9.3M (2326026 substances)
 - Average 40M
- 33GB total

Optimizations

Memoization FTW! If I've solved it once, why solve it again?

Optimizations

- Start from the smallest set and compare it with other sets
- In cases where the majority of the substances have this property, consider inverting the set — checking against the substances which don't have the property

If it's a bitmap... why not use bitmap tools

- In order to fit the universe, 10000×10000 1 bit PNG
- 12K to 6.8M
 - Median 1.7M
 - Average 2.4M

Bitmaps



- ImageMagick on a 3.5 GHz box requires 10s/image intersection using a single core.
 - `convert FILE1 FILE2 -compose Darken -composite out.png`
- Using 5 cores can do 30/minute
- Pre-Calculating all of the paths will take ~20 years.

One Data Structure to Rule them All

???

Array

Credits

- Presentation created with [markdeep-slides](#)
- Markdown and Diagram rendering via [Markdeep](#)

Resources: General Topics

- Big O notation
- SIMD

Resources: Bitmaps

- Better bitmap performance with Roaring bitmaps
- A General SIMD-based Approach to Accelerating Compression Algorithms
- Roaring Bitmap

Resources: Cuckoos

- [Cuckoo Hashing Visualization](#)
- [Cuckoo Hashing](#) (the original paper)
- [cuckoo - GoDoc](#) (go library)
- [garethm/cuckoo](#) (ruby)

Resources: Searches

- Binary search
- Linear search
- Prune and search

Resources: Galloping Search

- Beating the binary search algorithm – interpolation search, galloping search
- AN ALMOST OPTIMAL ALGORITHM FOR UNBOUNDED SEARCHING

Resources: Interpolation

- [Interpolation search](#)

Resources: Peacock Hashes

- Approximately-Perfect Hashing: Improving Network Throughput through Efficient Off-chip Routing Table Lookup
- Peacock Hashing: Deterministic and Updatable Hashing for High Performance Networking

Resources: Sets

- A Fast Set Intersection Algorithm for Sorted Sequences
- Experimental Analysis of a Fast Intersection Algorithm for Sorted Sequences
- Experimental Comparison of Set Intersection Algorithms for Inverted Indexing
- Fast Set Intersection in Memory
- Faster Adaptive Set Intersections for Text Searching
- Faster Set Intersection with SIMD instructions by Reducing Branch Mispredictions
- SIMD Compression and the Intersection of Sorted Integers

Resources: Cheminformatics

- The PubChem Project
- SDF Toolkit (perl)
- Chemistry Development Kit (java)
- Simplified molecular-input line-entry system (SMILES)

Resources: Raspberry Pi

- <http://raspberrypi.org>
- <http://blog.hypriot.org>
- <http://matthewkwilliams.com>

Thank You!

