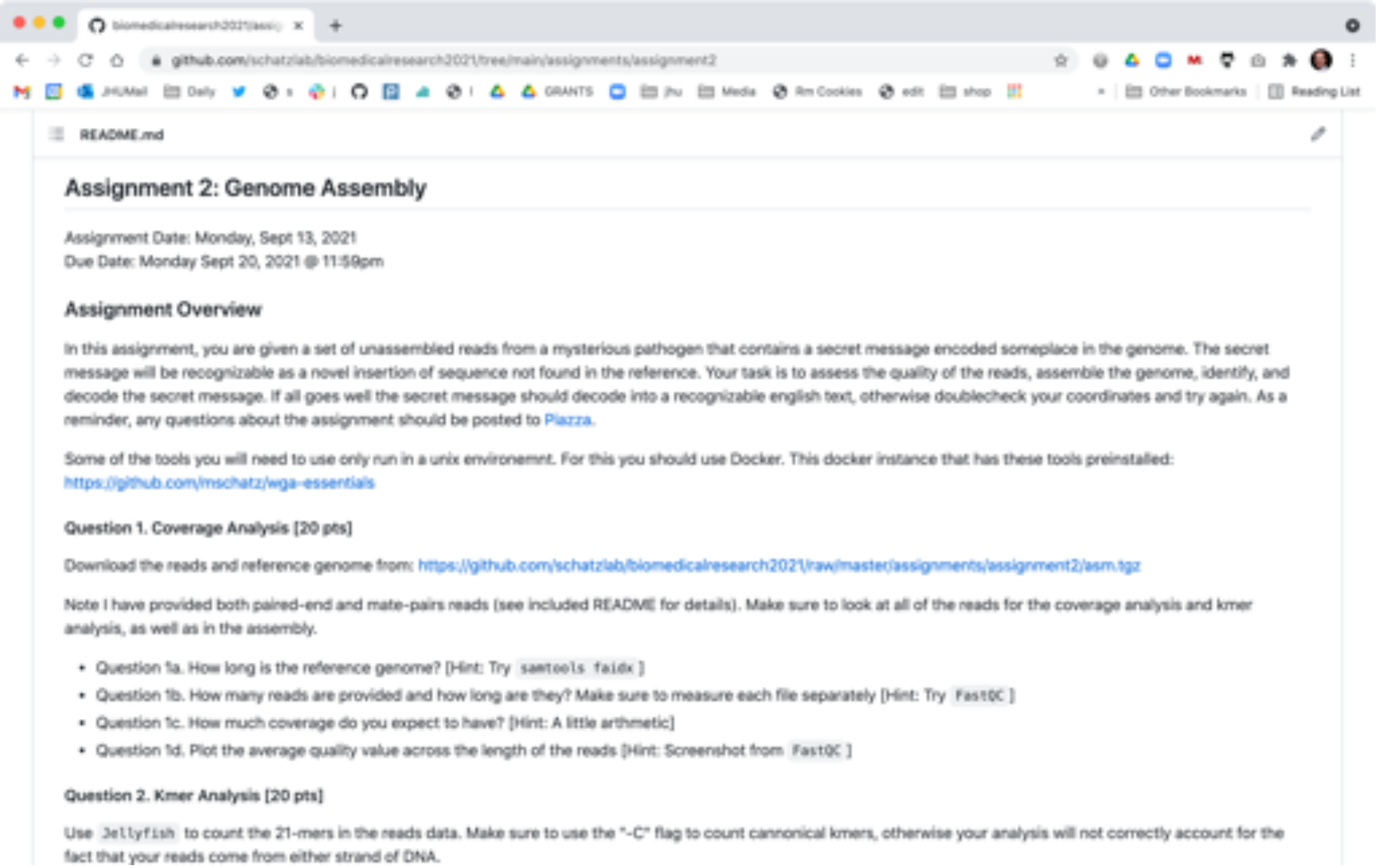
Read Mapping

Michael Schatz

September 22, 2021

Lecture 7. Computational Biomedical Research

Assignment 2: Genome Assembly
Due Monday Sept 20 @ 11:59pm


biomedicalresearch2021/assig
github.com/schatzlab/biomedicalresearch2021/tree/main/assignments/assignment2
JHUMail
Daily
GRANTS
jhu
Media
Rm Cookies
edit
shop
Other Bookmarks
Reading List
README.md

Assignment 2: Genome Assembly

Assignment Date: Monday, Sept 13, 2021
Due Date: Monday Sept 20, 2021 @ 11:59pm

Assignment Overview

In this assignment, you are given a set of unassembled reads from a mysterious pathogen that contains a secret message encoded someplace in the genome. The secret message will be recognizable as a novel insertion of sequence not found in the reference. Your task is to assess the quality of the reads, assemble the genome, identify, and decode the secret message. If all goes well the secret message should decode into a recognizable english text, otherwise doublecheck your coordinates and try again. As a reminder, any questions about the assignment should be posted to Piazza.

Some of the tools you will need to use only run in a unix environemnt. For this you should use Docker. This docker instance that has these tools preinstalled: https://github.com/mschatz/wga-essentials

Question 1. Coverage Analysis [20 pts]

Download the reads and reference genome from: https://github.com/schatzlab/biomedicalresearch2021/raw/master/assignments/assignment2/asm.tgz

Note I have provided both paired-end and mate-pairs reads (see included README for details). Make sure to look at all of the reads for the coverage analysis and kmer analysis, as well as in the assembly.

- Question 1a. How long is the reference genome? [Hint: Try `samtools faidx`]
- Question 1b. How many reads are provided and how long are they? Make sure to measure each file separately [Hint: Try `FastQC`]
- Question 1c. How much coverage do you expect to have? [Hint: A little arthmetic]
- Question 1d. Plot the average quality value across the length of the reads [Hint: Screenshot from `FastQC`]

Question 2. Kmer Analysis [20 pts]

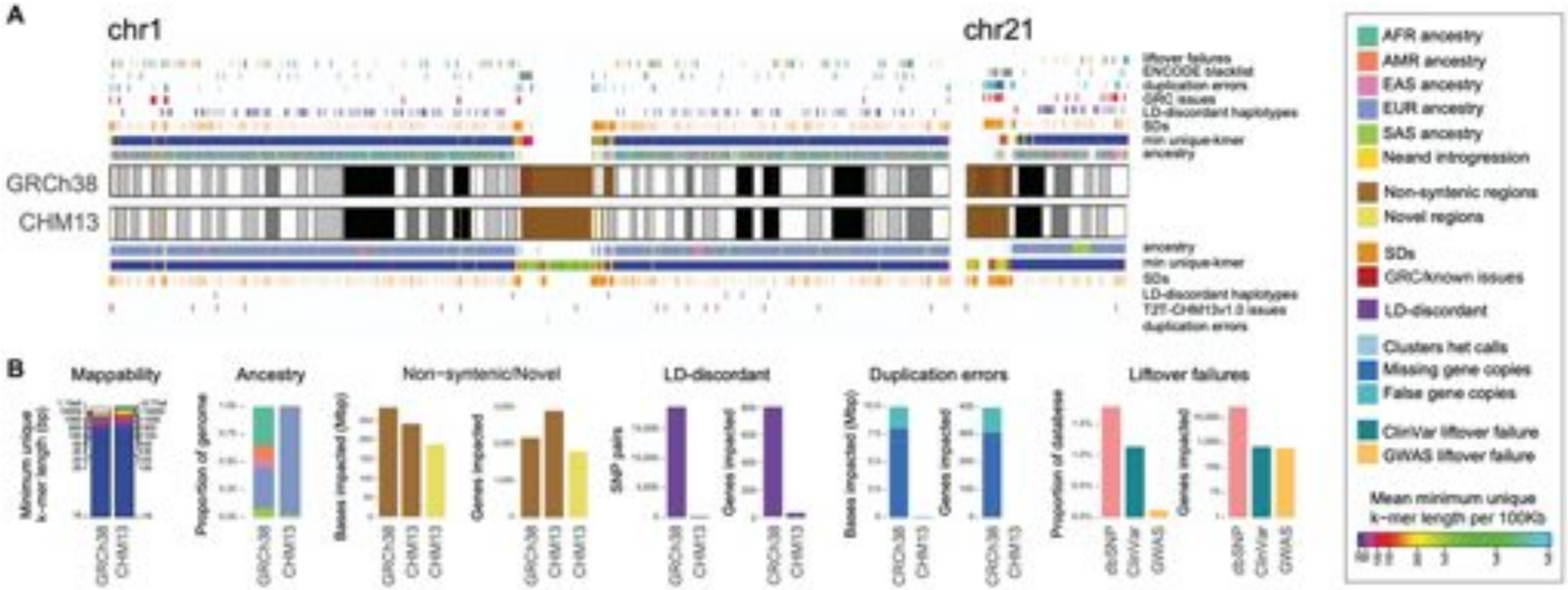
Use `Jellyfish` to count the 21-mers in the reads data. Make sure to use the "-C" flag to count cannonical kmers, otherwise your analysis will not correctly account for the fact that your reads come from either strand of DNA.

https://github.com/schatzlab/biomedicalresearch2021

Assignment 3:Variant Calling

Postponed for 1 week!

The *complete* sequence of a human genome

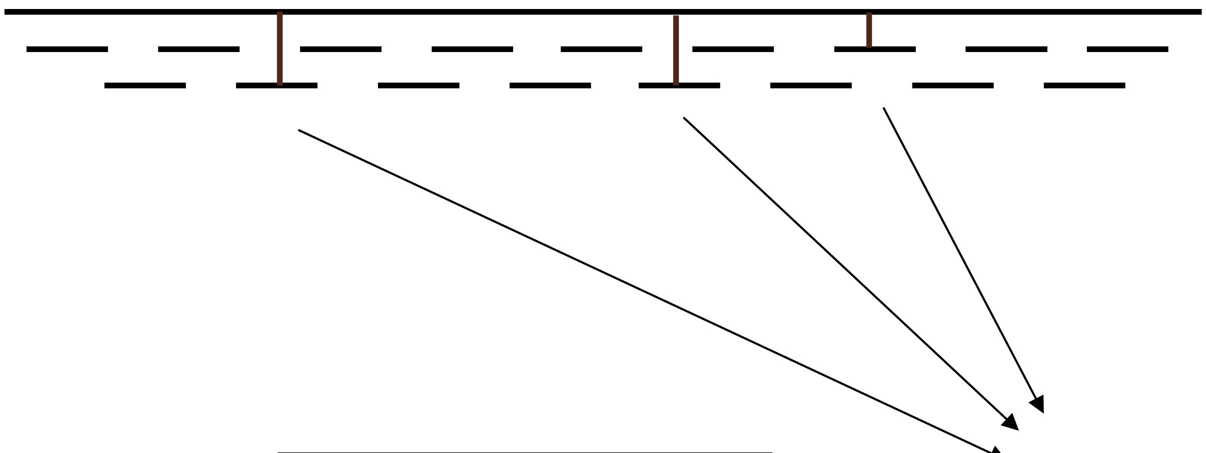


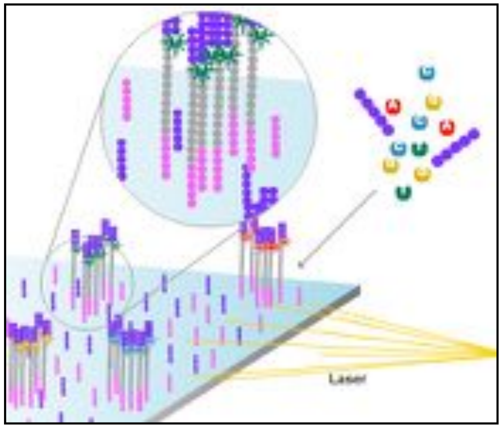
CHM13v1.1 genome size is **3.057 Gbp with zero Ns**
Every chromosome is telomere-to-telomere, quality estimated >Q70
~190 Mbp (3–6%) of new sequence vs. GRCh38, fixes thousands of errors

A complete reference genome improves analysis of human genetic variation
Aganezov, S*, Yan, SM*, Soto, DC*, Kirsche, M*, Zarate, S*, *et al.* (2021) bioRxiv. doi: https://doi.org/10.1101/2021.07.12.452063

Personal Genomics

How does your genome compare to the reference?





Heart Disease

Cancer

Presidential Smile

Searching for GATTACA

- Where is GATTACA in the human genome?

- Strategy 1: Brute Force

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
G	A	T	T	A	C	A									

No match at offset 1

Searching for GATTACA

- Where is GATTACA in the human genome?

- Strategy 1: Brute Force

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
	G	A	T	T	A	C	A								

Match at offset 2

Searching for GATTACA

- Where is GATTACA in the human genome?

- Strategy 1: Brute Force

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
		G	A	T	T	A	C	A	…						

No match at offset 3…

Searching for GATTACA

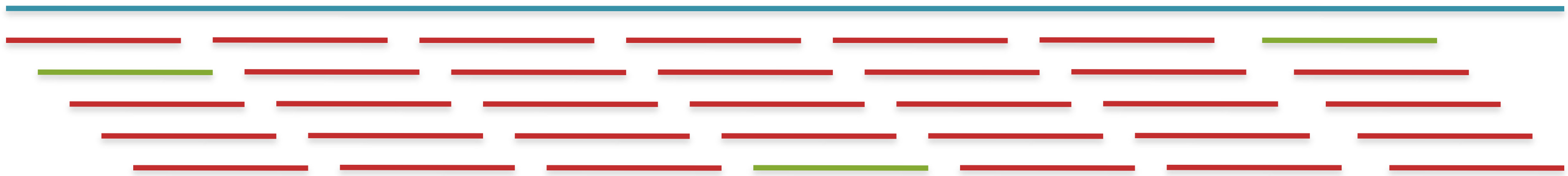
- Where is GATTACA in the human genome?

- Strategy 1: Brute Force

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
								G	A	T	T	A	C	A	

No match at offset 9 <- Checking each possible position takes time

Brute Force Analysis



- Brute Force:
  - At every possible offset in the genome:
    - Do all of the characters of the query match?

- Analysis
  - Simple, easy to understand
  - Genome length = n [3B]
  - Query length = m [7]
  - Comparisons: (n-m+1) * m [21B]

- Overall runtime: O(nm)

[How long would it take if we double the genome size, read length?]

[How long would it take if we double both?]

Brute Force Reflections

Why check every position?

- GATTACA can't possibly start at position 15 [WHY?]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
								G	A	T	T	A	C	A	

- Improve runtime to O(n + m) [3B + 7]
  - If we double both, it just takes twice as long
  - Knuth-Morris-Pratt, 1977
  - Boyer-Moyer, 1977, 1991

- For one-off scans, this is the best we can do (optimal performance)
  - We have to read every character of the genome, and every character of the query
  - For short queries, runtime is dominated by the length of the genome

How can we make this go faster?

Lets play the hi/lo game

Hi/Lo Game

Im thinking of a secret number …

Call out a number, and I'll tell you if you guessed it, or are too hi or too low

Just like grade school ….

Except my secret number is between 1 and 1,000,000 ☺

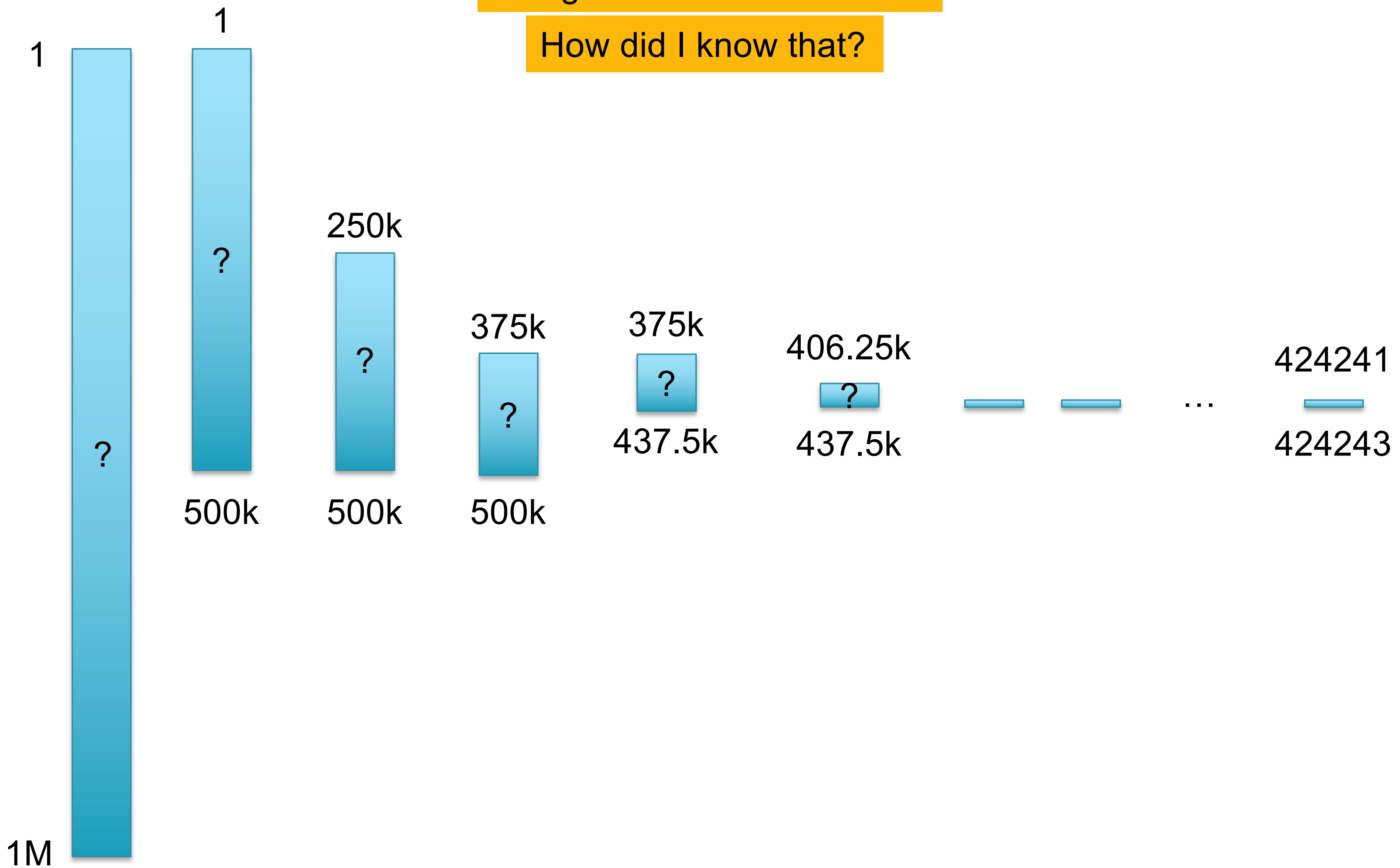
Possibilities: 1
2
3
…
…
…
…
…
…
999,998
999,999
1,000,000

Results???

Hi/Lo Game

<20 guesses to find 424242

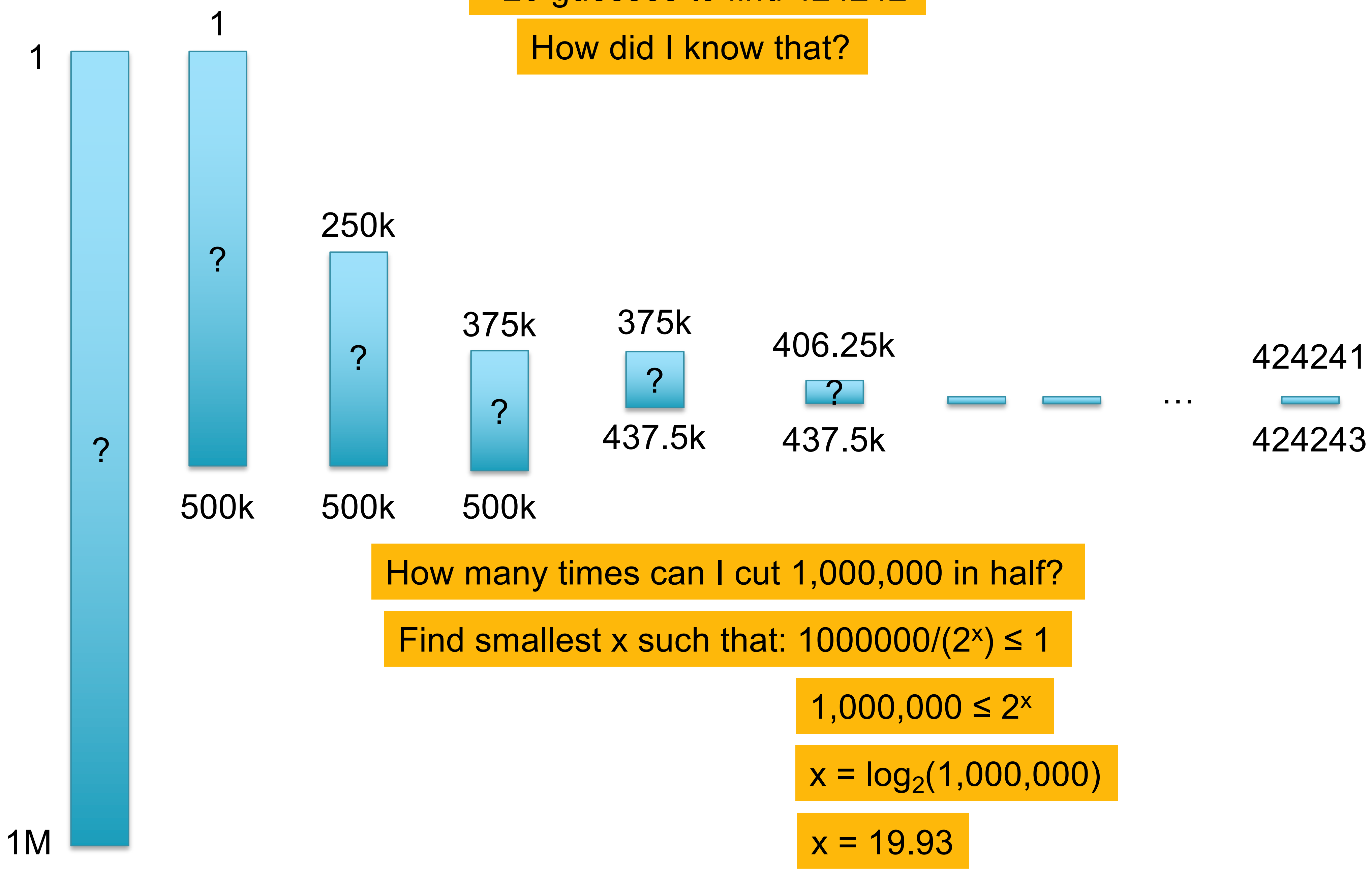
How did I know that?



Hi/Lo Game

<20 guesses to find 424242

How did I know that?



How many times can I cut 1,000,000 in half?

Find smallest x such that: $1000000/(2^x) \leq 1$

$1{,}000{,}000 \leq 2^x$

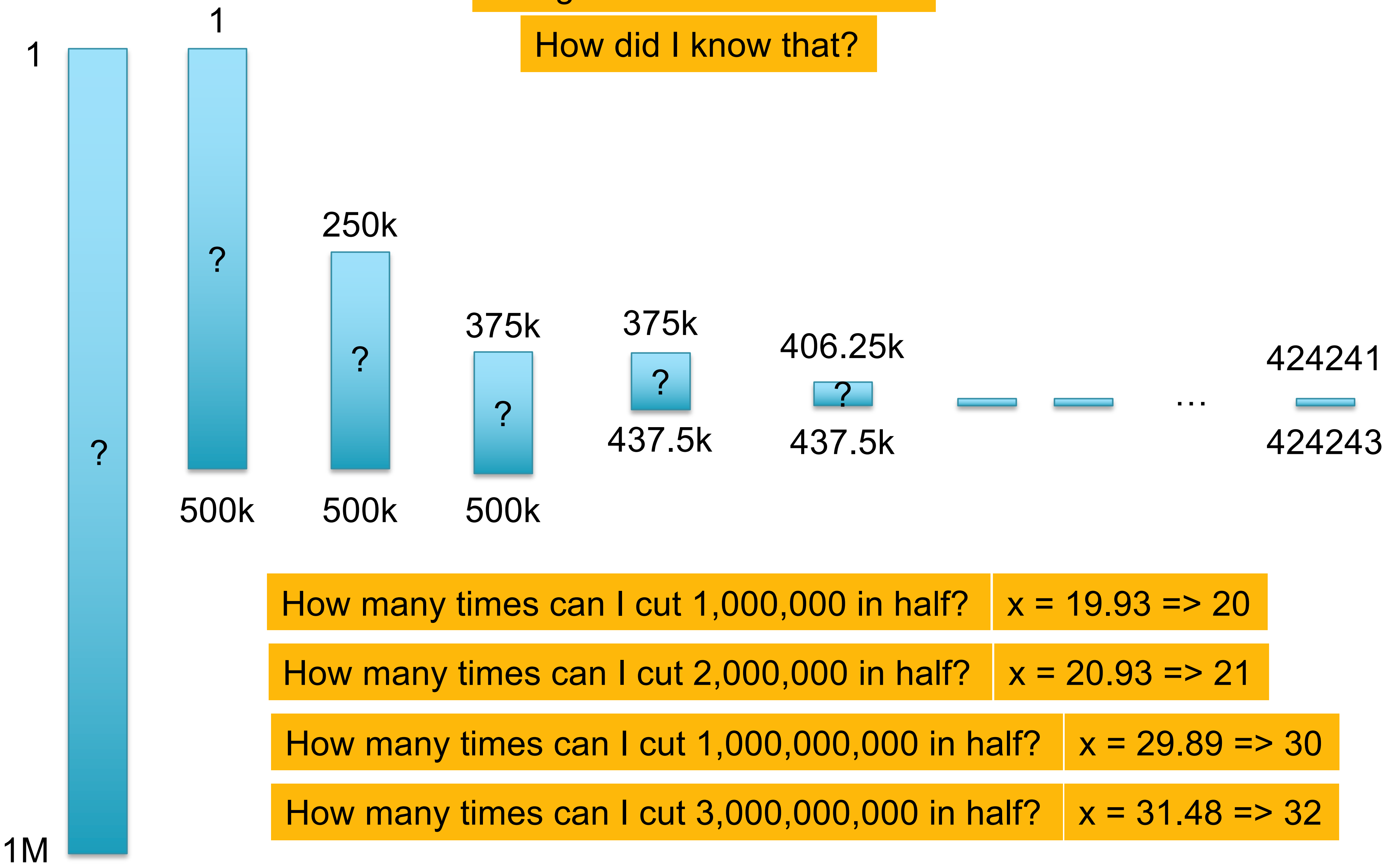
$x = \log_2(1{,}000{,}000)$

$x = 19.93$

Hi/Lo Game

<20 guesses to find 424242

How did I know that?



How many times can I cut 1,000,000 in half?	x = 19.93 => 20
How many times can I cut 2,000,000 in half?	x = 20.93 => 21
How many times can I cut 1,000,000,000 in half?	x = 29.89 => 30
How many times can I cut 3,000,000,000 in half?	x = 31.48 => 32

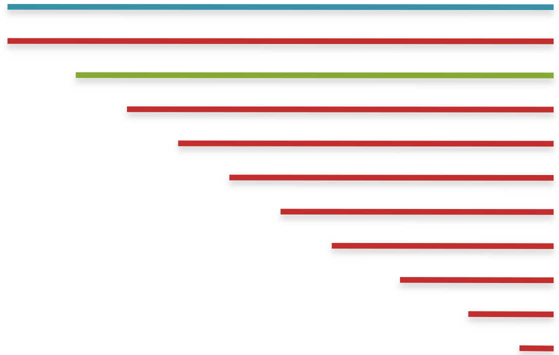
Searching the Phone Book

- What if we need to check many queries?
  - We don't need to check every page of the phone book to find 'Schatz'
  - Sorting alphabetically lets us immediately skip 96% (25/26) of the book *without any loss in accuracy*

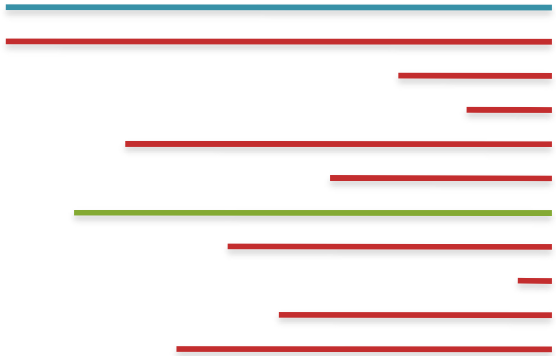
Searching the Phone Book

- What if we need to check many queries?
  - We don't need to check every page of the phone book to find 'Schatz'
  - Sorting alphabetically lets us immediately skip 96% (25/26) of the book *without any loss in accuracy*

- Sorting the genome: Suffix Array (Manber & Myers, 1991)
  - Sort every suffix of the genome



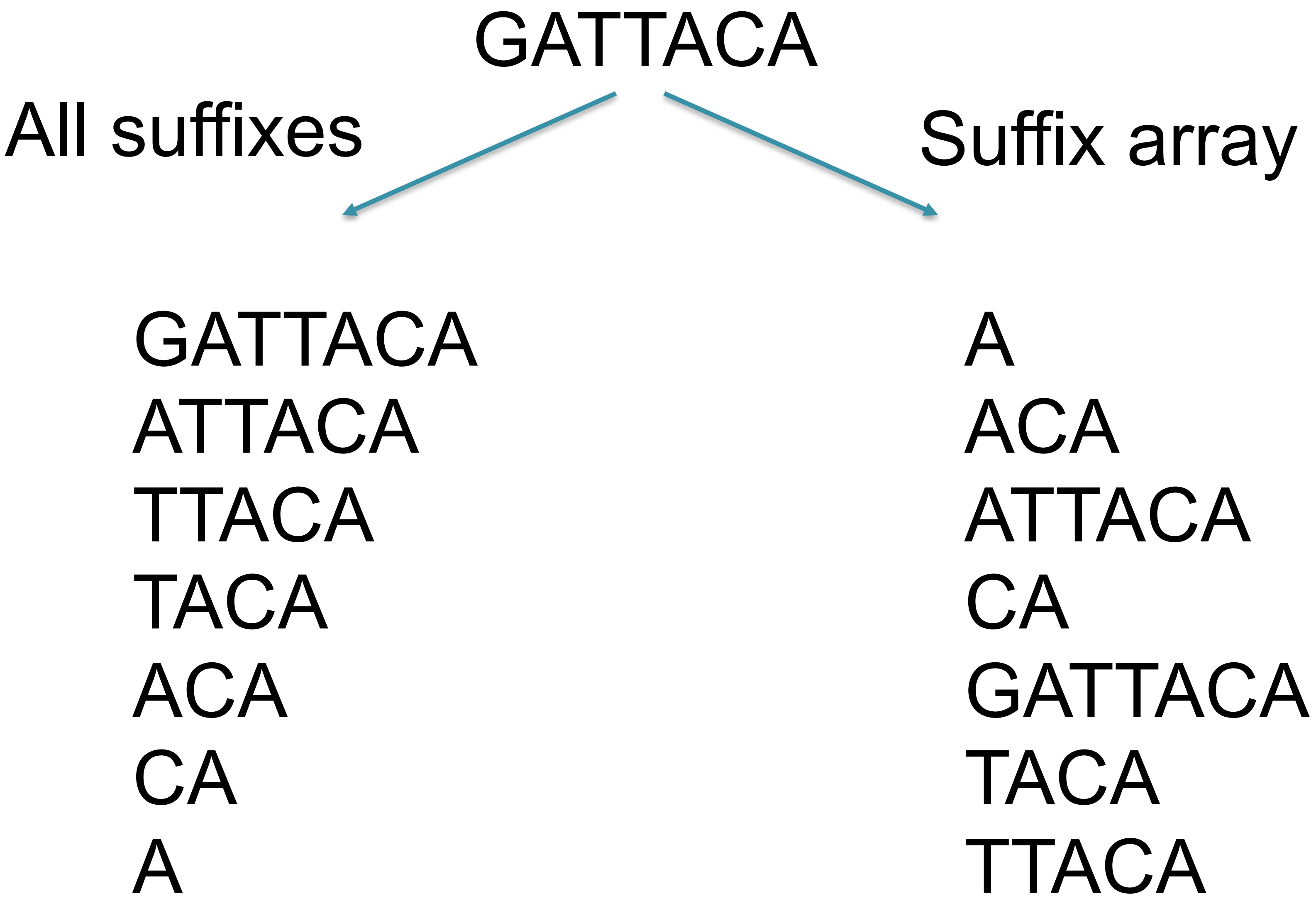
Split into n suffixes



Sort suffixes alphabetically

[Challenge Question: How else could we split the genome?]

Suffix Arrays: Searching the Phone Book



GATTACA

All suffixes

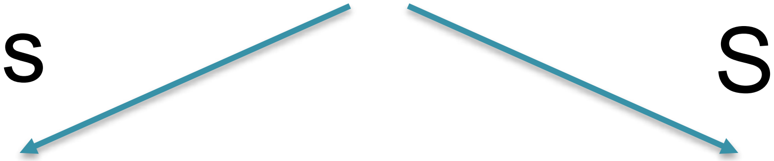
GATTACA
ATTACA
TTACA
TACA
ACA
CA
A

Suffix array

A
ACA
ATTACA
CA
GATTACA
TACA
TTACA

Suffix Arrays: Searching the Phone Book

GATTACA



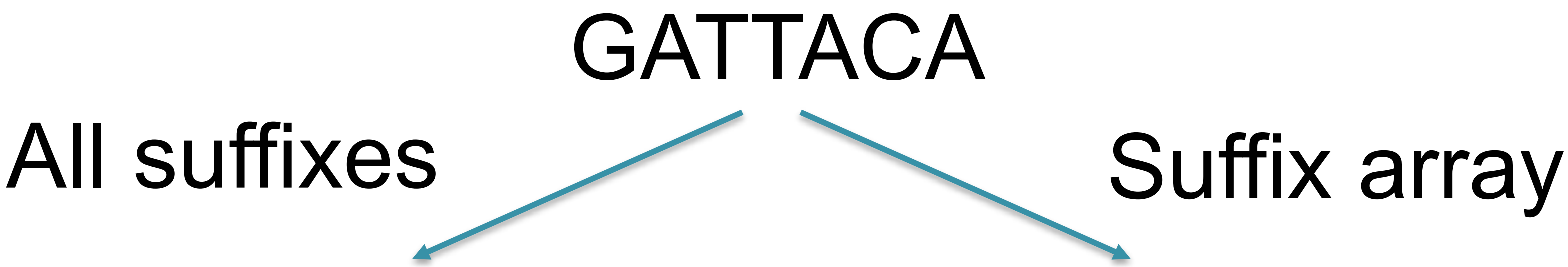
All suffixes

GATTACA
ATTACA
TTACA
TACA
ACA
CA
A

Suffix array

#	p	
1	7	A
2	5	ACA
3	3	ATTACA
4	6	CA
5	1	GATTACA
6	4	TACA
7	3	TTACA

Suffix Arrays: Searching the Phone Book



GATTACA

All suffixes

Suffix array

	#	p
GATTACA	1	7
ATTACA	2	5
TTACA	3	3
TACA	4	6
ACA	5	1
CA	6	4
A	7	3

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15;

Lo →

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Hi →

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC

Lo →

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Hi →

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC

    => Higher: Lo = Mid + 1

Lo →

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Hi →

Searching the Index

- Strategy 2: Binary search
    - Compare to the middle, refine as higher or lower

- Searching for GATTACA
    - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
    - Middle = Suffix[8] = CC

      => Higher: Lo = Mid + 1

    - Lo = 9; Hi = 15;

	#	Sequence	Pos
	1	ACAGATTACC…	6
	2	ACC…	13
	3	AGATTACC…	8
	4	ATTACAGATTACC…	3
	5	ATTACC…	10
	6	C…	15
	7	CAGATTACC…	7
	8	CC…	14
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
Hi →	15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC

	#	Sequence	Pos
	1	ACAGATTACC…	6
	2	ACC…	13
	3	AGATTACC…	8
	4	ATTACAGATTACC…	3
	5	ATTACC…	10
	6	C…	15
	7	CAGATTACC…	7
	8	CC…	14
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
Hi →	15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11;

Lo →

Hi →

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

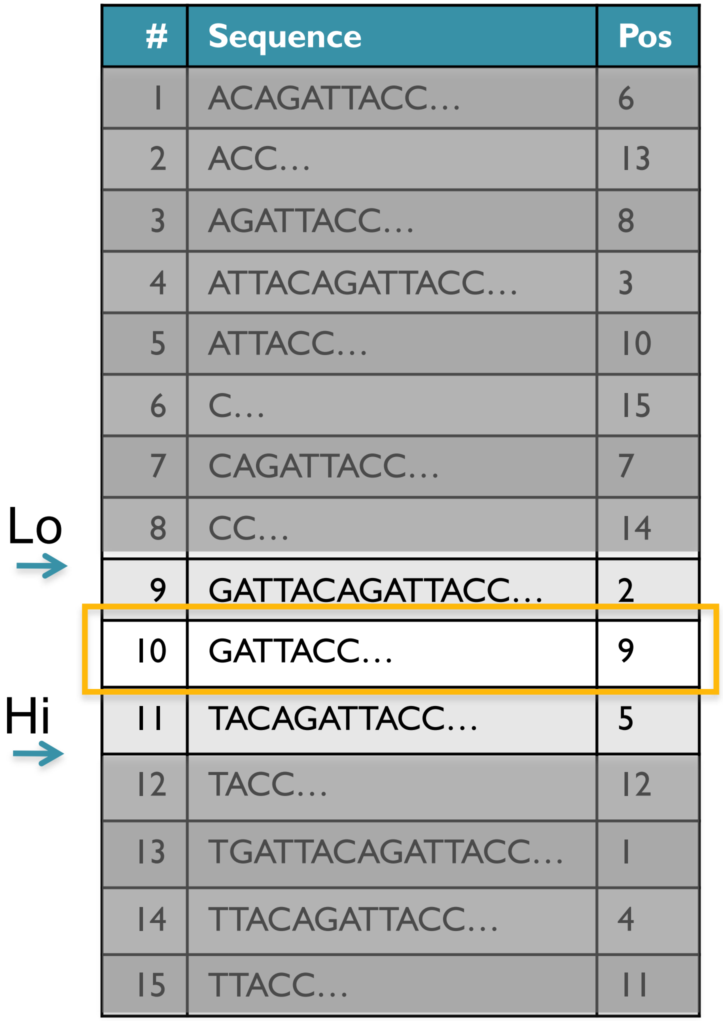
- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC

    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC

    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11; Mid = (9+11)/2 = 10
  - Middle = Suffix[10] = GATTACC



#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11; Mid = (9+11)/2 = 10
  - Middle = Suffix[10] = GATTACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 9;

Lo
Hi

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC

    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC

    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11; Mid = (9+11)/2 = 10
  - Middle = Suffix[10] = GATTACC

    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 9; Mid = (9+9)/2 = 9
  - Middle = Suffix[9] = GATTACA…

    => Match at position 2!

Lo
Hi

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Binary Search Analysis

- Binary Search

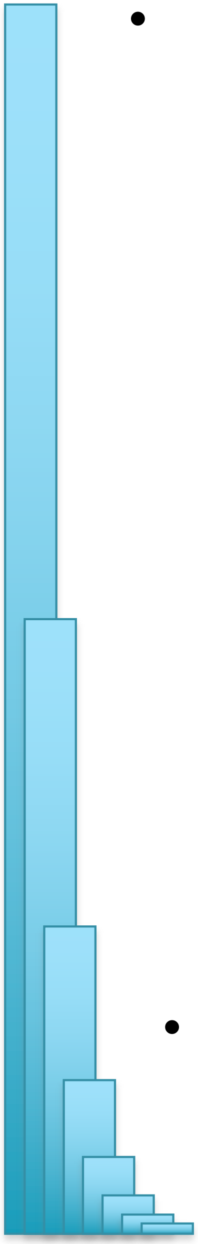
```
Initialize search range to entire list
    mid = (hi+lo)/2; middle = suffix[mid]
    if query matches middle: done
    else if query < middle: pick low range
    else if query > middle: pick hi range
Repeat until done or empty range                    [WHEN?]
```

- Analysis
  - More complicated method
  - How many times do we repeat?
    - How many times can it cut the range in half?
    - Find smallest x such that: $n/(2^x) \le 1$; $x = \lg_2(n)$ [32]

- Total Runtime: O(m lg n)
  - More complicated, but much faster!
  - Looking up a query loops 32 times instead of 3B

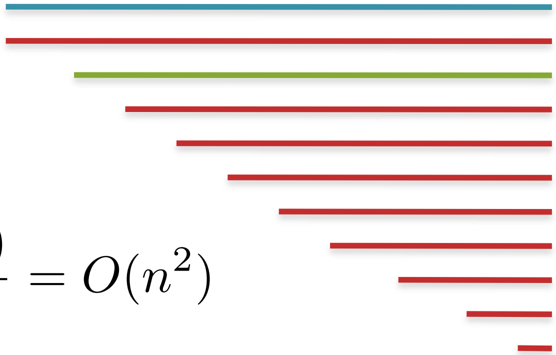
[How long does it take to search 6B or 24B nucleotides?]

Suffix Array Construction



- How can we store the suffix array?

  [How many characters are in all suffixes combined?]



$$S = 1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

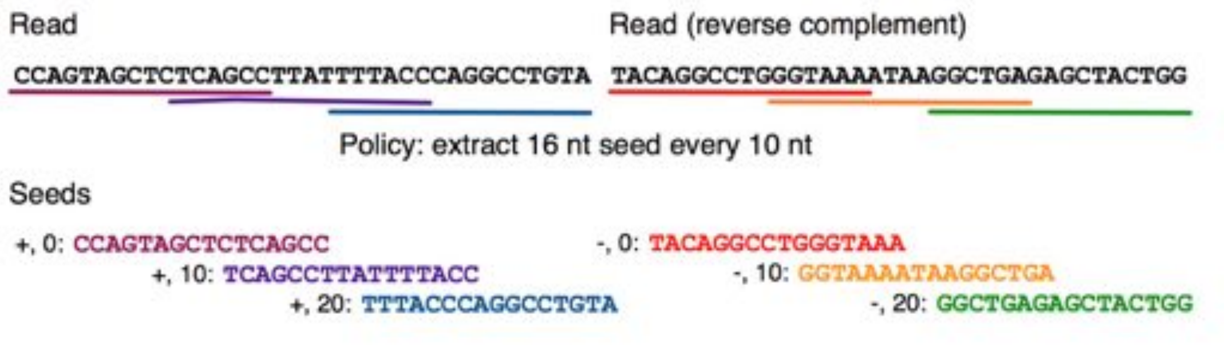
  - Hopeless to explicitly store 4.5 billion billion characters
  - Instead use implicit representation
    - Keep 1 copy of the genome, and a list of sorted offsets
    - Storing 3 billion offsets fits on a server (12GB)
- Searching the array is very fast, but it takes time to construct
  - This time will be amortized over many, many searches
  - Run it once "overnight" and save it away for all future queries

Pos
6
13
8
3
10
15
7
14
2
9
5
12
1
4
11

TGATTACAGATTACC

Bowtie2 Algorithm

1. Split read into segments



Read

CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA

Read (reverse complement)

TACAGGCCTGGGTAAAATAAGGCTGAGAGCTACTGG

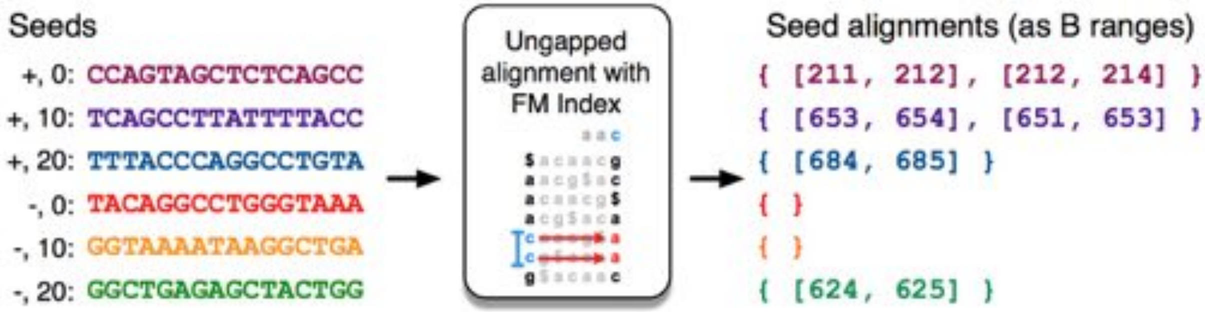
Policy: extract 16 nt seed every 10 nt

Seeds

+, 0: CCAGTAGCTCTCAGCC
+, 10: TCAGCCTTATTTTACC
+, 20: TTTACCCAGGCCTGTA

-, 0: TACAGGCCTGGGTAAA
-, 10: GGTAAAATAAGGCTGA
-, 20: GGCTGAGAGCTACTGG

2. Lookup each segment and prioritize



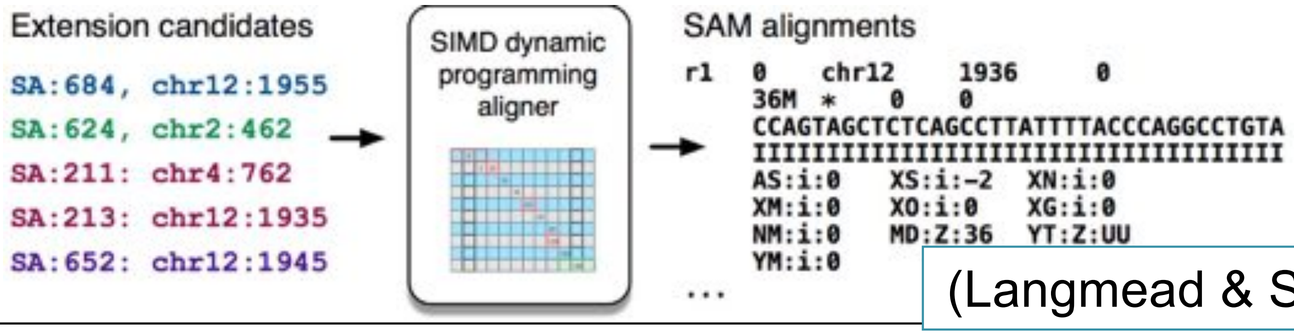
Seeds

+, 0: CCAGTAGCTCTCAGCC
+, 10: TCAGCCTTATTTTACC
+, 20: TTTACCCAGGCCTGTA
-, 0: TACAGGCCTGGGTAAA
-, 10: GGTAAAATAAGGCTGA
-, 20: GGCTGAGAGCTACTGG

Seed alignments (as B ranges)

```
{ [211, 212], [212, 214] }
{ [653, 654], [651, 653] }
{ [684, 685] }
{ }
{ }
{ [624, 625] }
```

3. Evaluate end-to-end match



Extension candidates

```
SA:684, chr12:1955
SA:624, chr2:462
SA:211: chr4:762
SA:213: chr12:1935
SA:652: chr12:1945
```

SAM alignments

```
r1   0    chr12    1936     0
     36M  *    0    0
     CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
     IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
     AS:i:0    XS:i:-2   XN:i:0
     XM:i:0    XO:i:0    XG:i:0
     NM:i:0    MD:Z:36   YT:Z:UU
     YM:i:0
...
```

(Langmead & Salzberg, 2012)

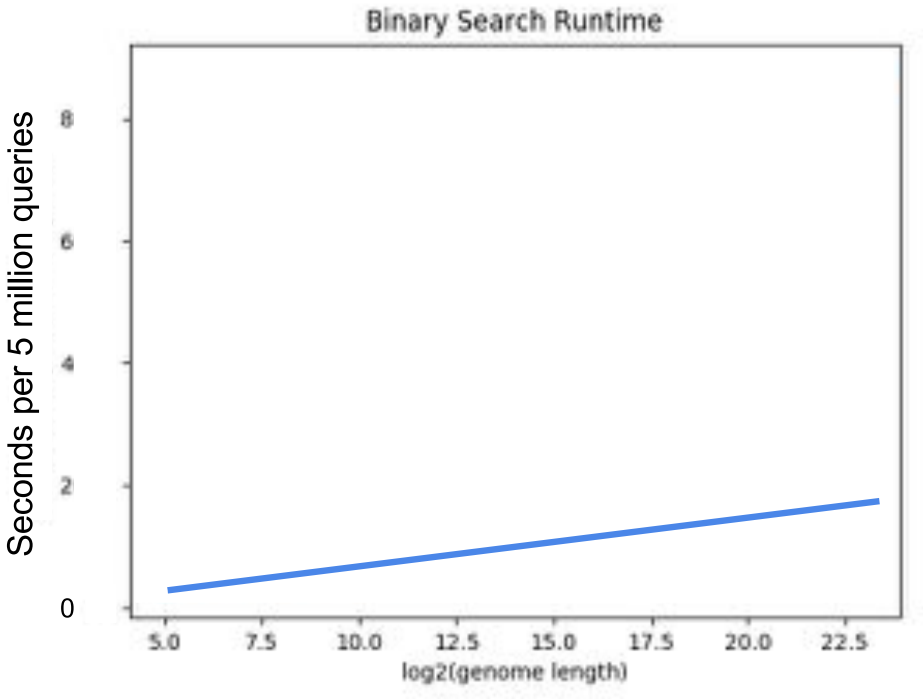
Sapling: Accelerating Suffix Array Queries with Learned Data Models

Melanie Kirsche

Arun Das

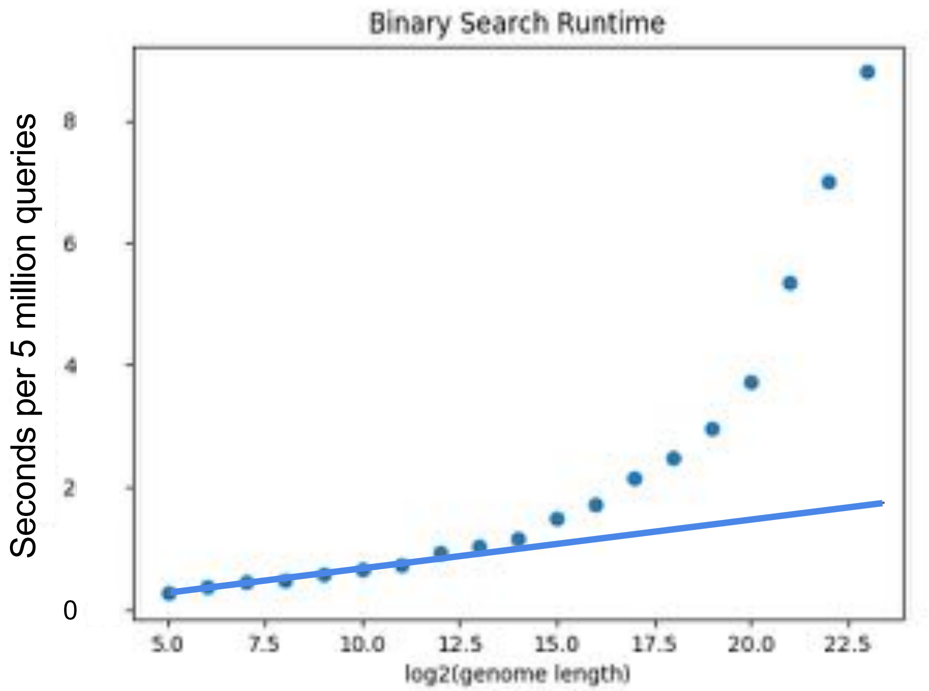
https://github.com/mkirsche/sapling

Caching and Binary Search



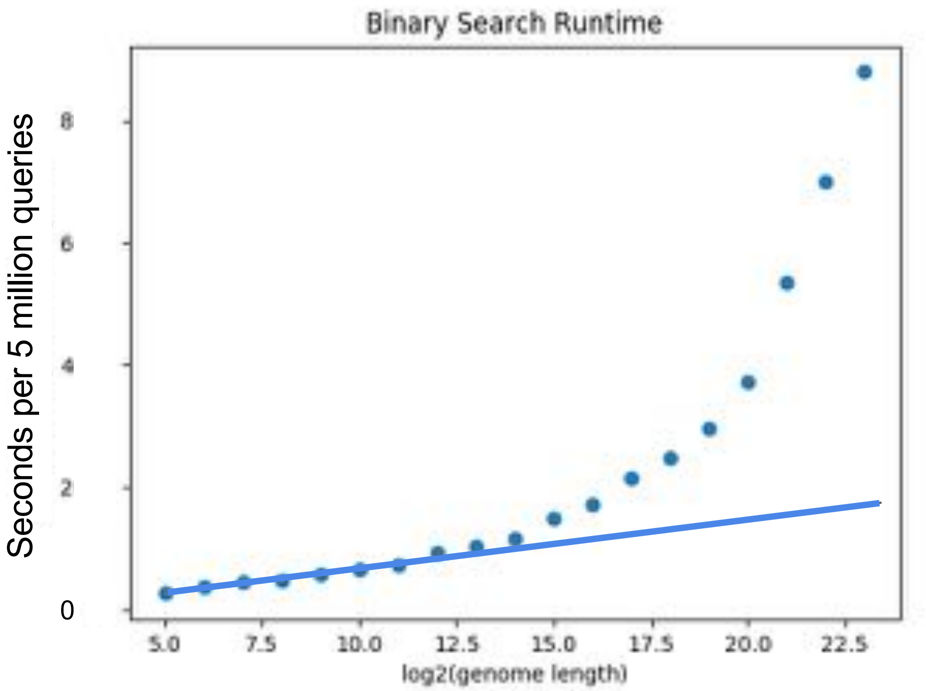
In theory, searching should scale linearly with log2 of the genome size

Caching and Binary Search



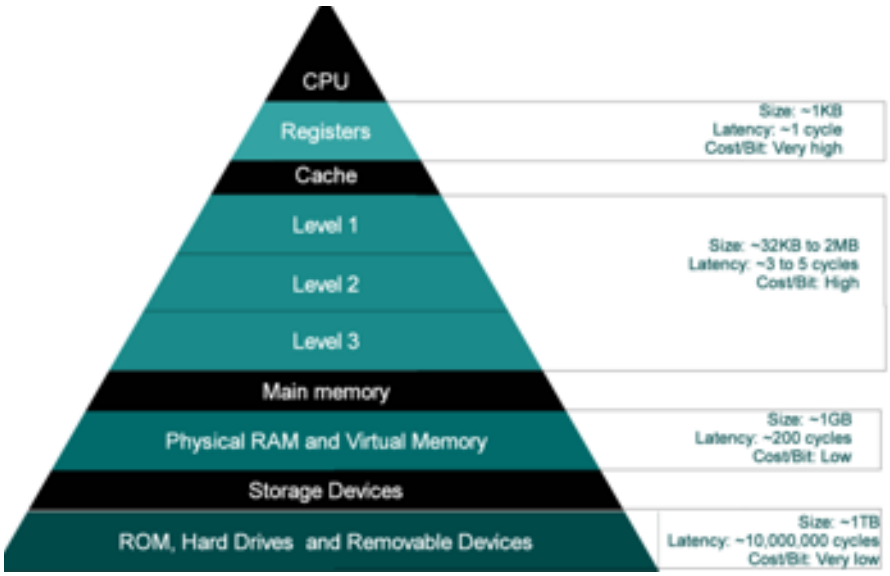
In practice, searching is much slower for large genome sizes

Caching and Binary Search



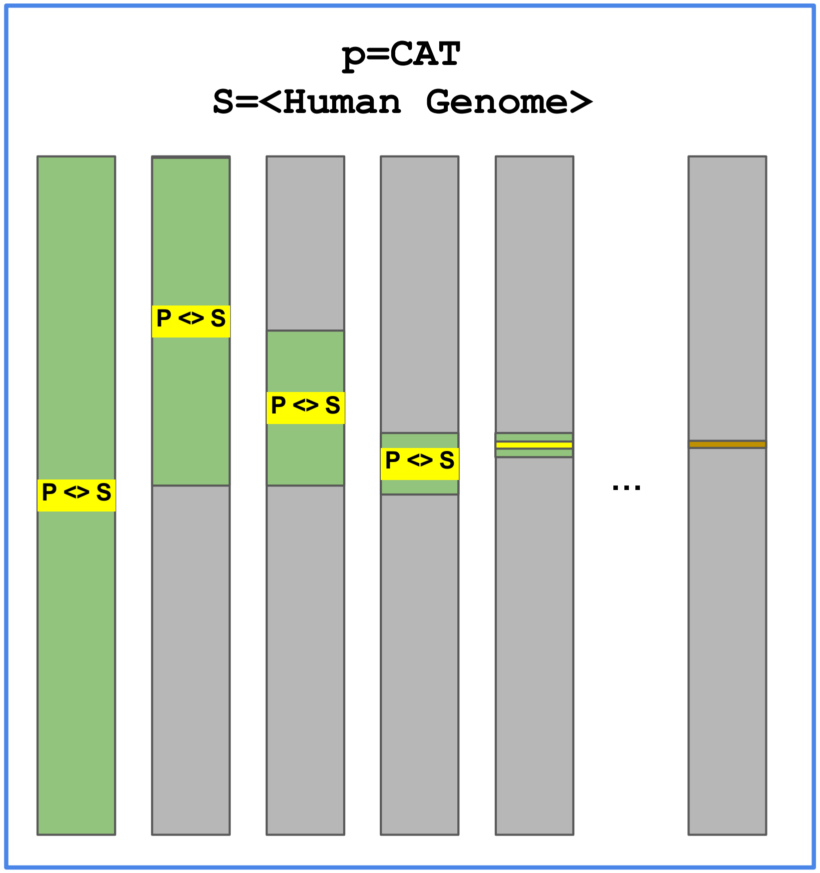
In practice, searching is much slower for large genome sizes

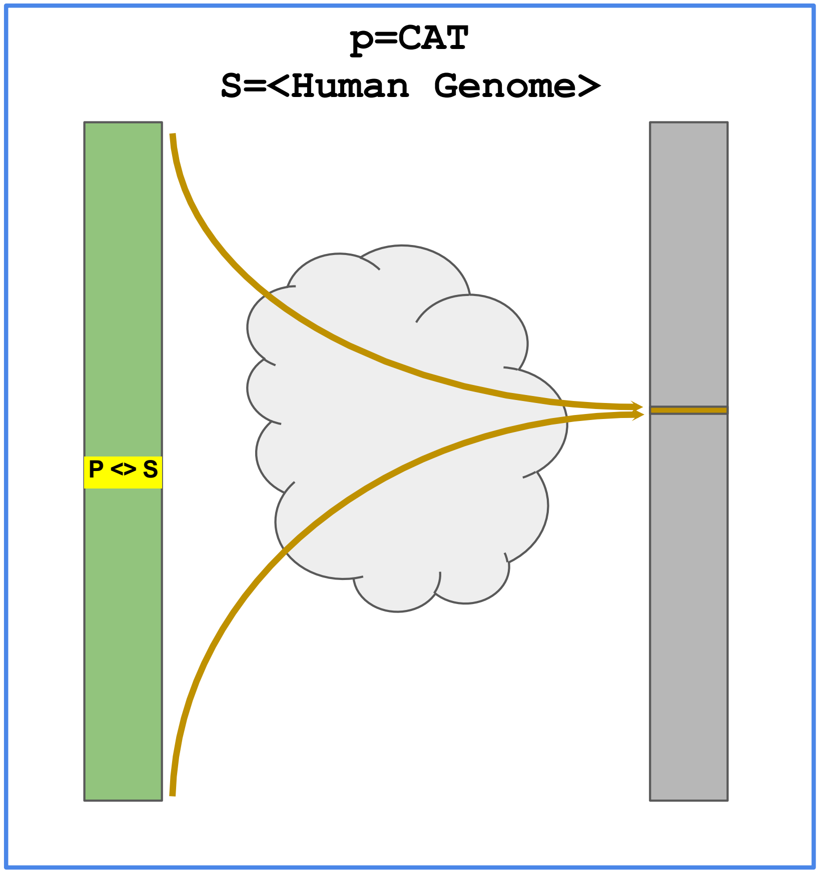
Memory Hierarchy



Binary search suffers from poor locality causing many lookups in main memory

Suffix Array Prediction





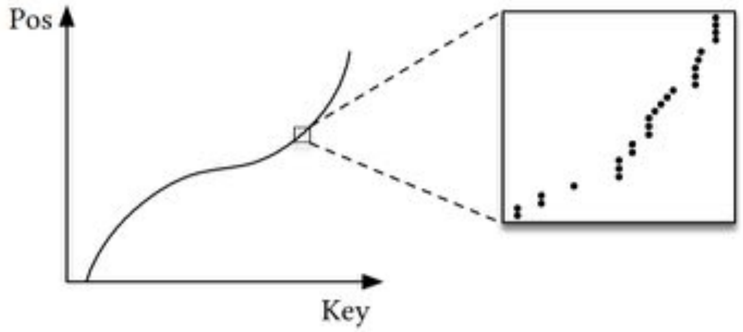
What if instead of a slow algorithmic approach to find the correct rows, we could somehow quickly guess/predict the correct rows?

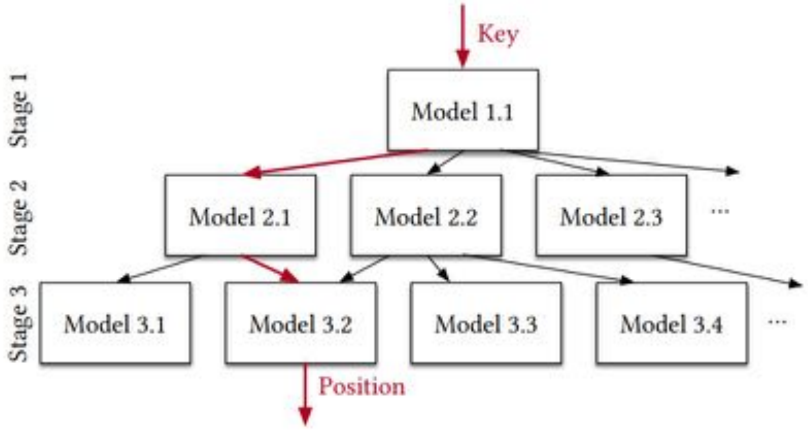
Learned Index Structures

Researchers at Google using neural networks to replace classical data structures such as B-Trees, HashMaps, and Bloom Filters

Train network to predict position of a data point in the structure given its value

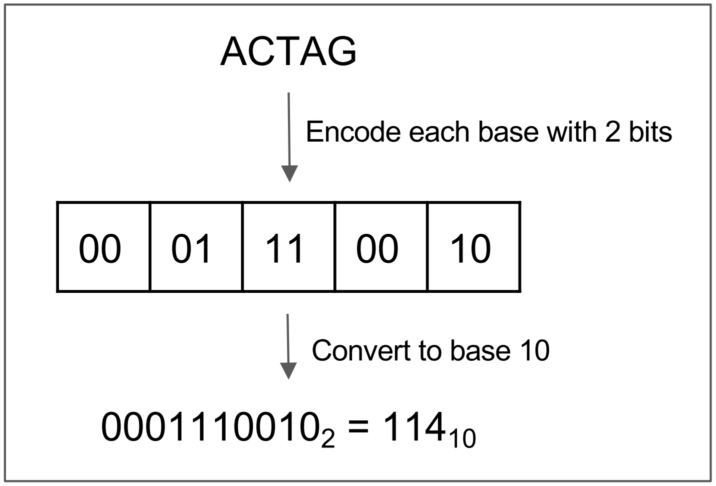
Compute the maximum error E = |predicted position - actual position| among all points in data structure. Then, narrow search to within E of predicted value.





Kraska et. al. “The Case for Learned Index Structures”. SIGMOD 2018.

Suffix Array Search as a Prediction Task



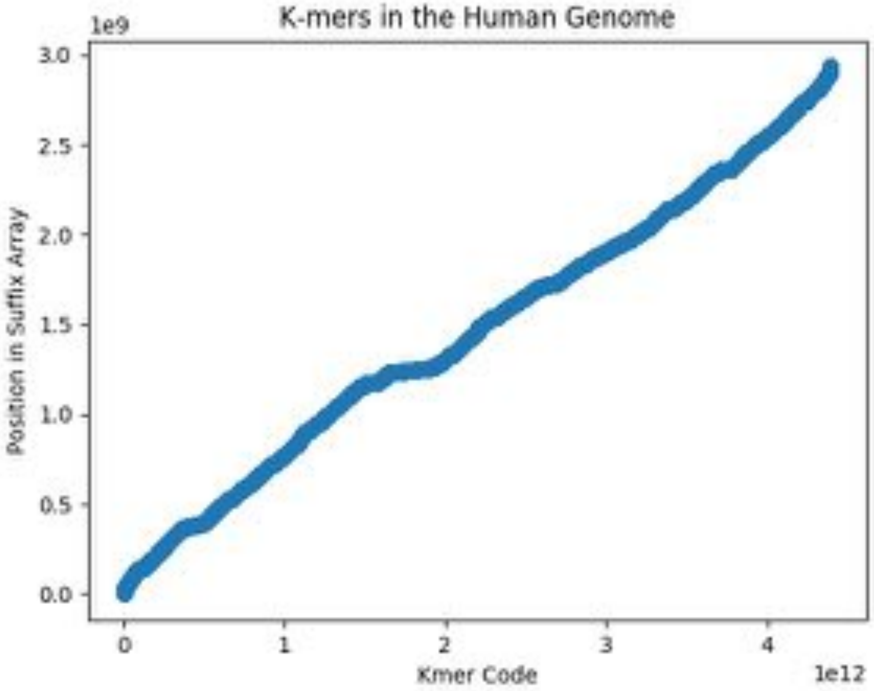
ACTAG

Encode each base with 2 bits

00	01	11	00	10

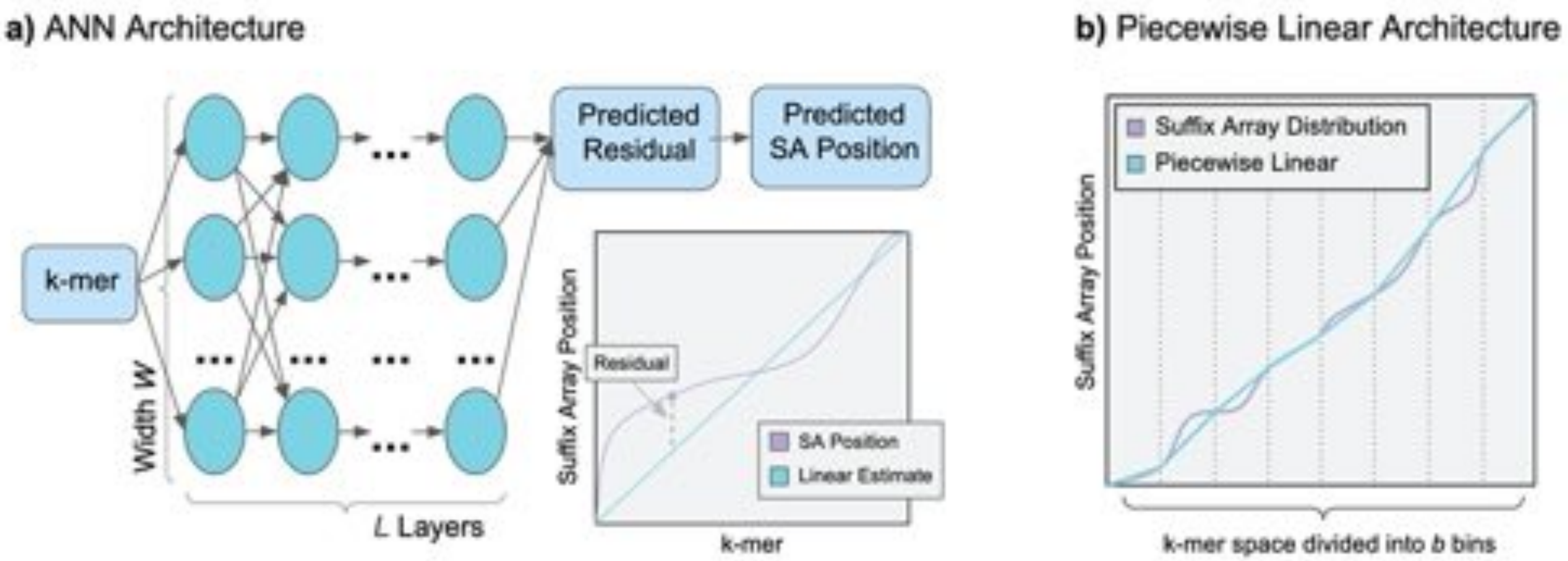
Convert to base 10

$0001110010_2 = 114_{10}$



Goal: Given a query string and a suffix array, predict the suffix array position where the suffix begins with that query string

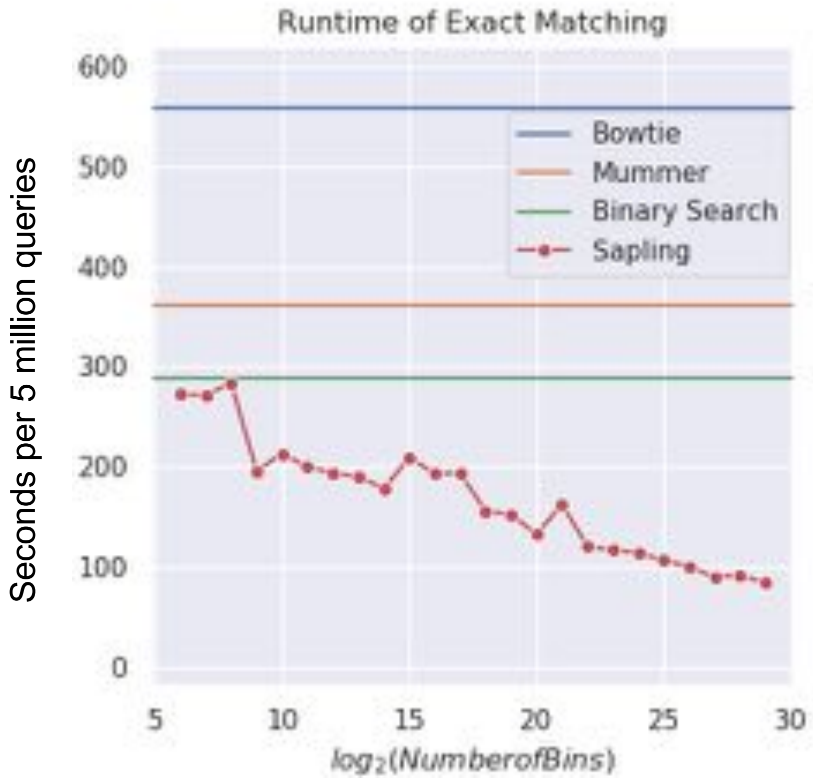
Prediction Schemes



a) ANN Architecture

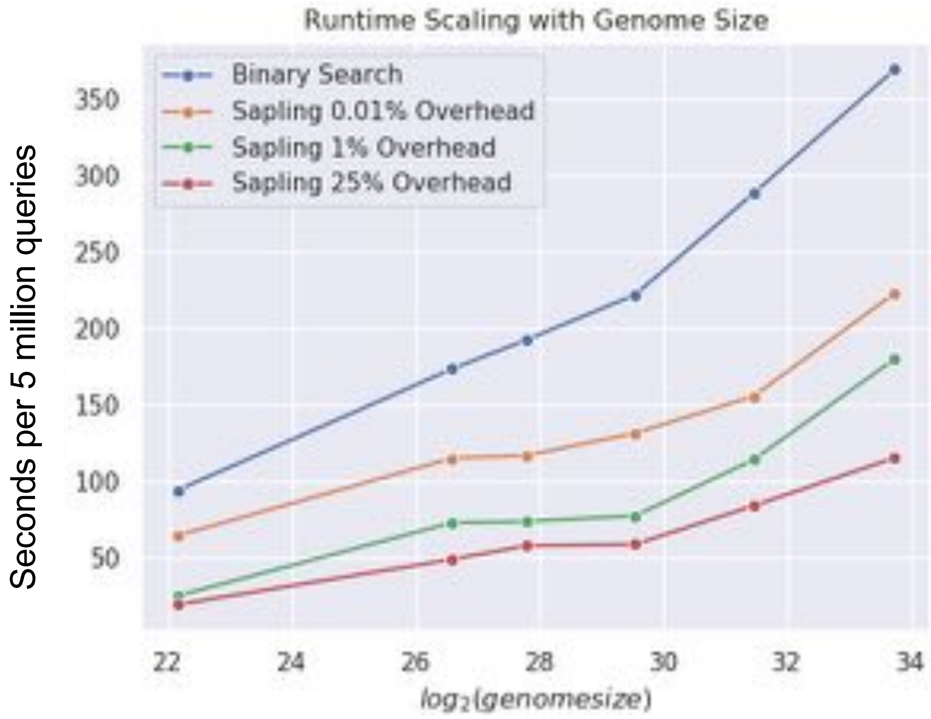
b) Piecewise Linear Architecture

Performance Results



0.1% space overhead

10% space overhead

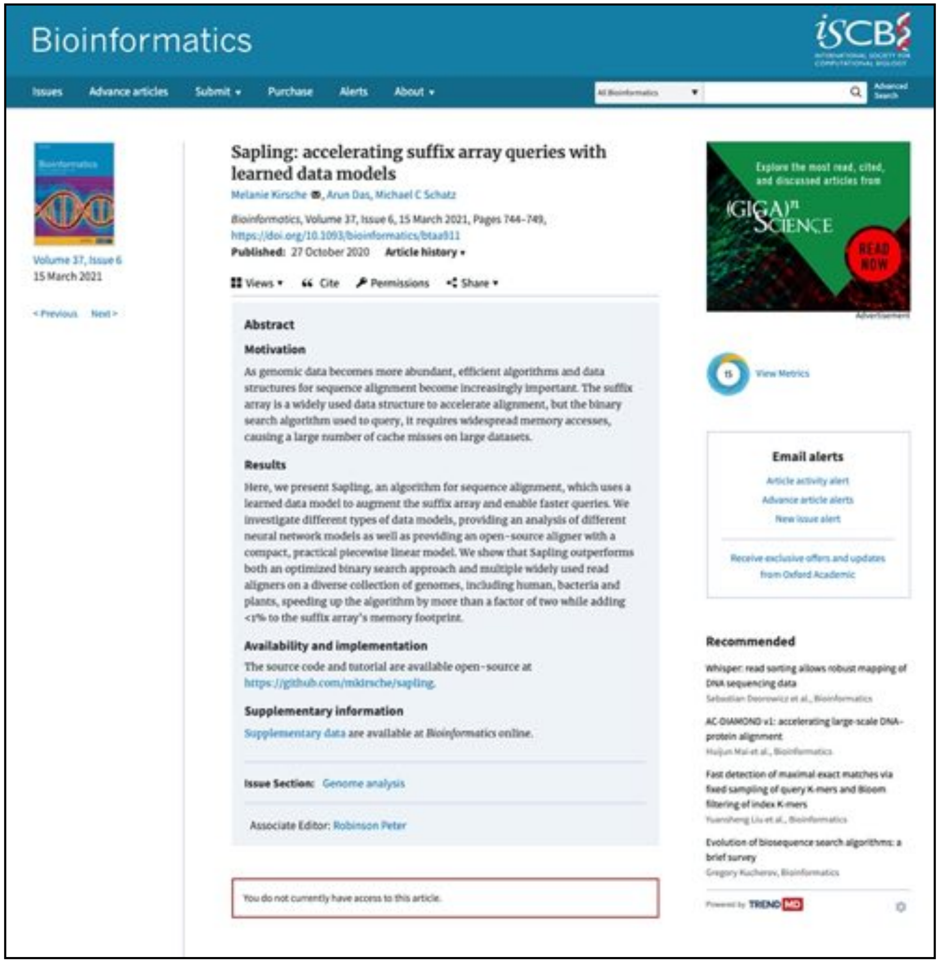


Conclusion

SAPLING allows faster string searching than existing approaches

- Scales well with genome length and could be used for searching large collections of genomes, e.g. metagenomics search

- Technique of treating data structure lookups as predictive function evaluation has the potential to speed up many other genomic data structures



Sapling: accelerating suffix array queries with learned data models

Melanie Kirsche, Arun Das, Michael C Schatz

Bioinformatics, Volume 37, Issue 6, 15 March 2021, Pages 744–749, https://doi.org/10.1093/bioinformatics/btaa911

Published: 27 October 2020 **Article history** ▾

Abstract

Motivation

As genomic data becomes more abundant, efficient algorithms and data structures for sequence alignment become increasingly important. The suffix array is a widely used data structure to accelerate alignment, but the binary search algorithm used to query, it requires widespread memory accesses, causing a large number of cache misses on large datasets.

Results

Here, we present Sapling, an algorithm for sequence alignment, which uses a learned data model to augment the suffix array and enable faster queries. We investigate different types of data models, providing an analysis of different neural network models as well as providing an open-source aligner with a compact, practical piecewise linear model. We show that Sapling outperforms both an optimized binary search approach and multiple widely used read aligners on a diverse collection of genomes, including human, bacteria and plants, speeding up the algorithm by more than a factor of two while adding <1% to the suffix array's memory footprint.

Availability and implementation

The source code and tutorial are available open-source at https://github.com/mkirsche/sapling.

Supplementary information

Supplementary data are available at *Bioinformatics* online.

Issue Section: Genome analysis

Associate Editor: Robinson Peter

You do not currently have access to this article.

Recommended

Whisper: read sorting allows robust mapping of DNA sequencing data
Sebastian Deorowicz et al., Bioinformatics

AC-DIAMOND v1: accelerating large-scale DNA-protein alignment
Huijun Mai et al., Bioinformatics

Fast detection of maximal exact matches via fixed sampling of query K-mers and Bloom filtering of index K-mers
Yuansheng Liu et al., Bioinformatics

Evolution of biosequence search algorithms: a brief survey
Gregory Kucherov, Bioinformatics

https://github.com/mkirsche/sapling