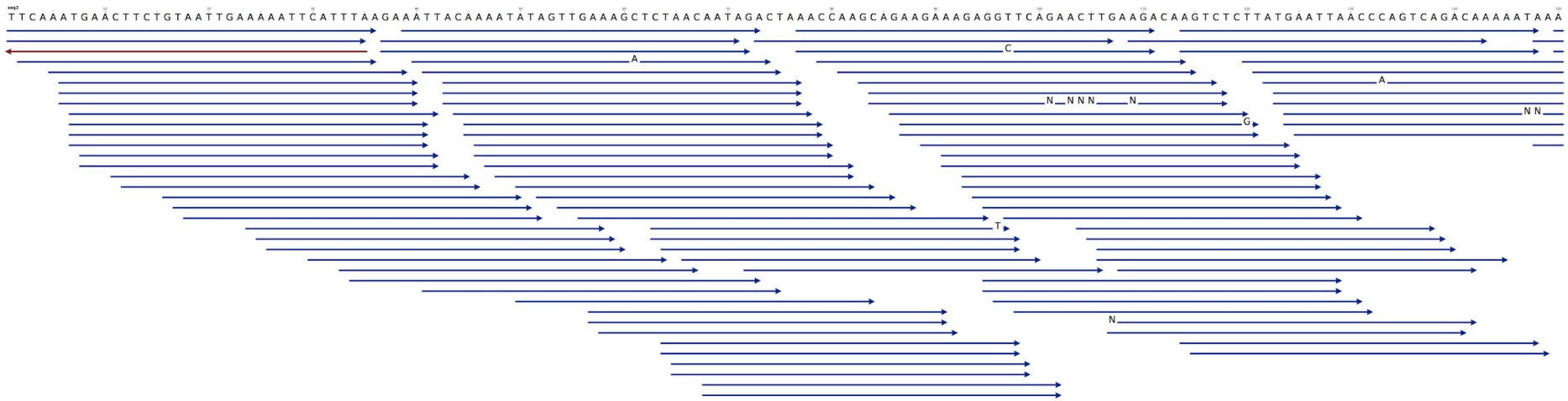
Data Structures for Genomic Read Alignment

Melanie Kirsche

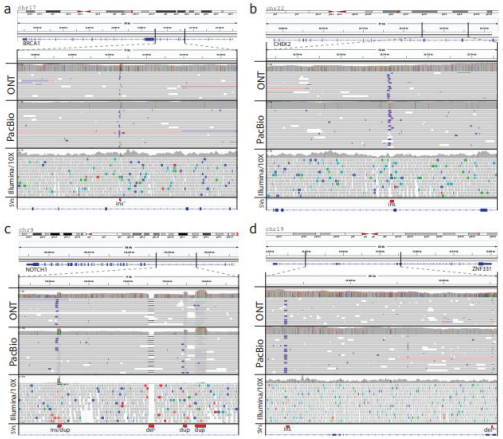
February 26, 2020

What is Read Alignment?

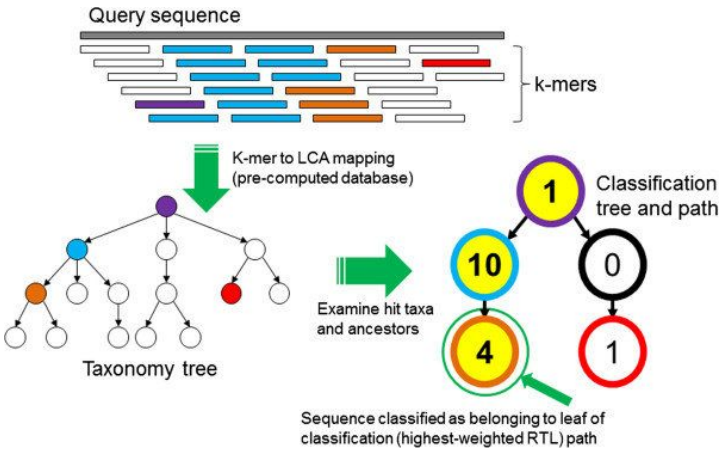


Why align reads?

- ➔ Variant calling
- ➔ Zoom in on target regions
- ➔ Sequence classification
- ➔ Assembly validation



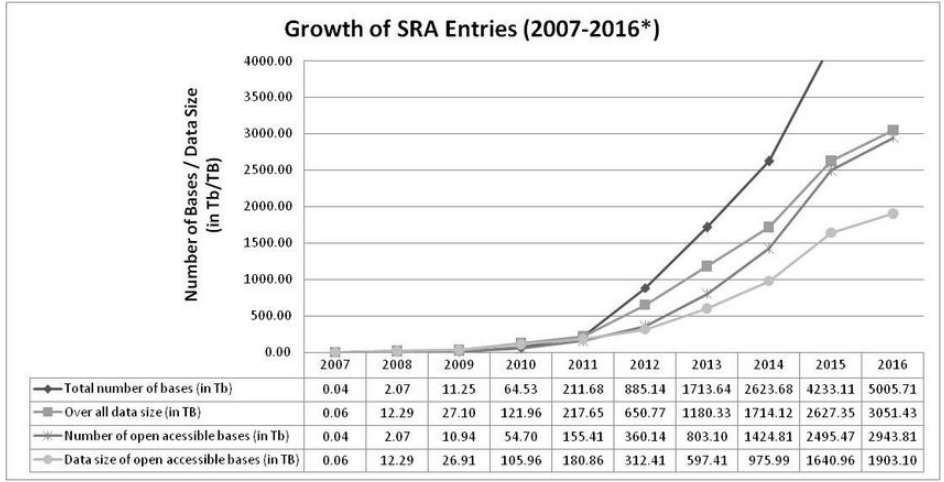
Comprehensive analysis of structural variants in breast cancer genomes using single molecule sequencing. (Aganezov et. al., 2019)



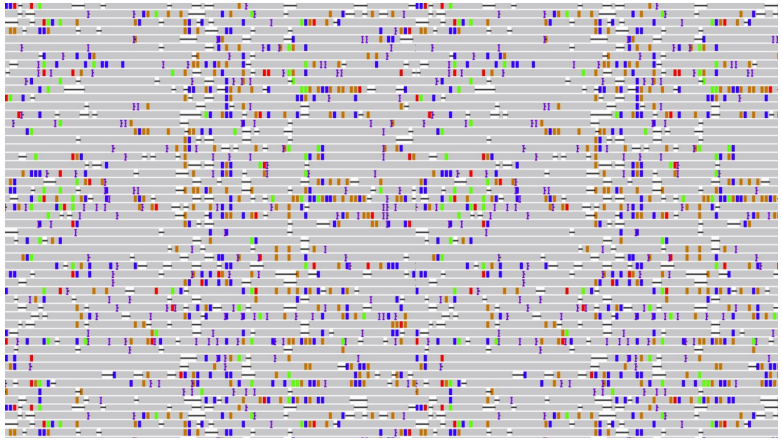
Kraken: ultrafast metagenomic sequence classification using exact alignments (Wood et. al., 2014)

Why is it difficult?

- Genomes are very large
- Many reads in each experiment
- Sequencing error
- Repetitive sequences



	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Total number of bases (in Tb)	0.04	2.07	11.25	64.53	211.68	885.14	1713.64	2623.68	4233.11	5005.71
Over all data size (in TB)	0.06	12.29	27.10	121.96	217.65	650.77	1180.33	1714.12	2627.35	3051.43
Number of open acessible bases (in Tb)	0.04	2.07	10.94	54.70	155.41	360.14	803.10	1424.81	2495.47	2943.81
Data size of open accessible bases (in TB)	0.06	12.29	26.91	105.96	180.86	312.41	597.41	975.99	1640.96	1903.10



https://nanoporetech.com/resource-centre/think-small-nanopores-sensing-and-synthesis

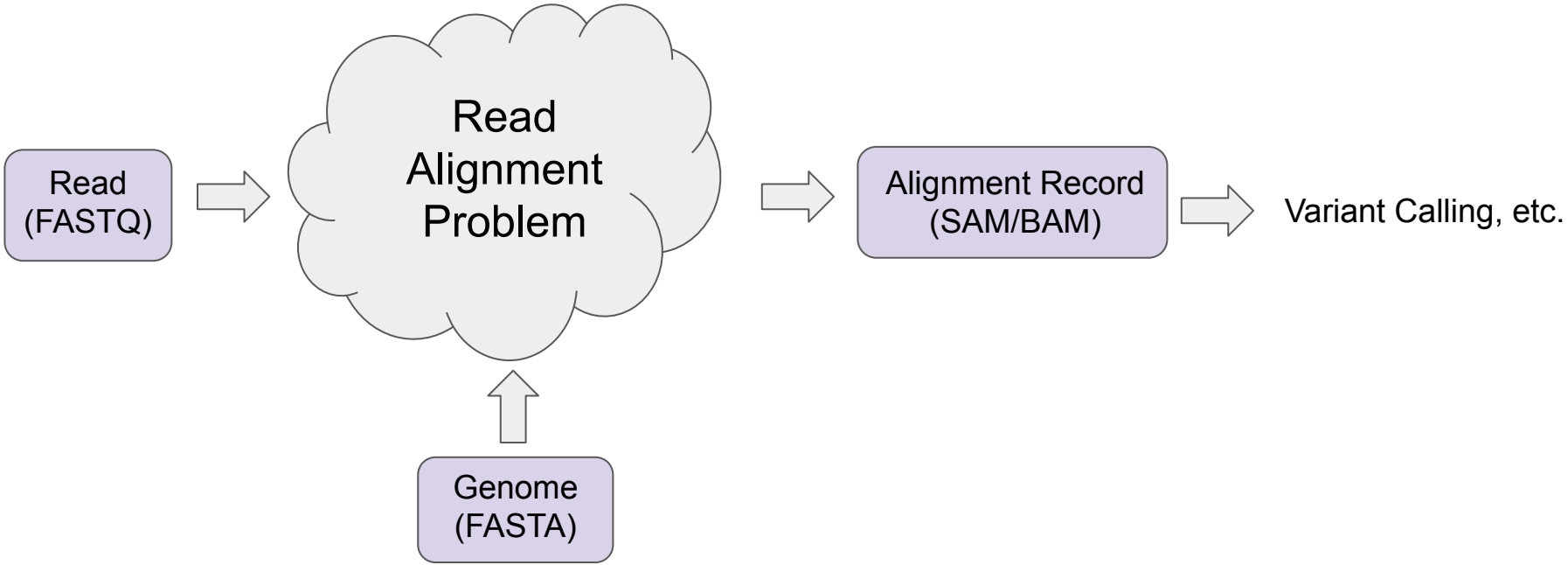
PLoS Genetics

Public Library of Science

Repetitive Elements May Comprise Over Two-Thirds of the Human Genome

A. P. Jason de Koning, Wanjun Gu, [...], and David D. Pollock

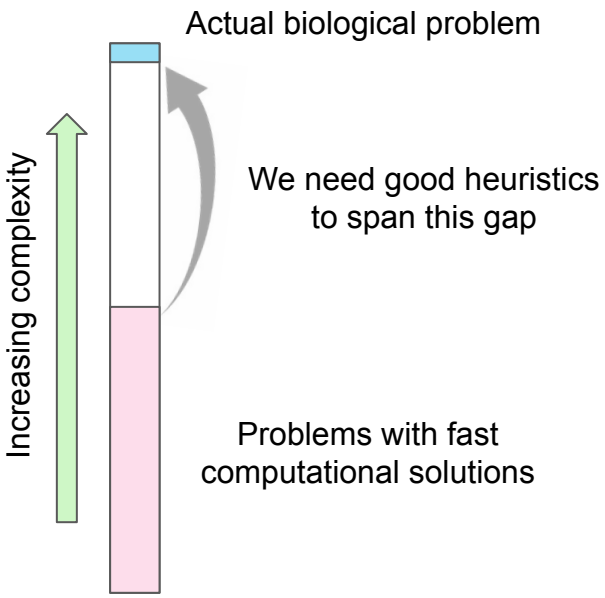
Alignment Algorithms



Breaking the Problem Down

Common technique is to solve an easier version of the problem and then bridge the gap later

Original Problem: Given a read, where in the genome is the best match for the read (given some distance measure)?



Easier Read Alignment Problems

Original Problem: Given a read, where in the genome is the best match for the read (given some distance measure)?

What would make this problem easier?

Easier Read Alignment Problems

Original Problem: Given a read, where in the genome is the best match for the read (given some distance measure)?

What would make this problem easier?

Source of Difficulty	Why is it harder?	Easier Version
Genome large	Lots of candidate matches	Restrict to some region of the genome
Sequencing error	Even “matches” aren’t exactly the same sequence	Require exact matches
Genome repetitive	Lots of locations can be close to the best match	Report any sufficiently good match

Seed and Extend Motivation

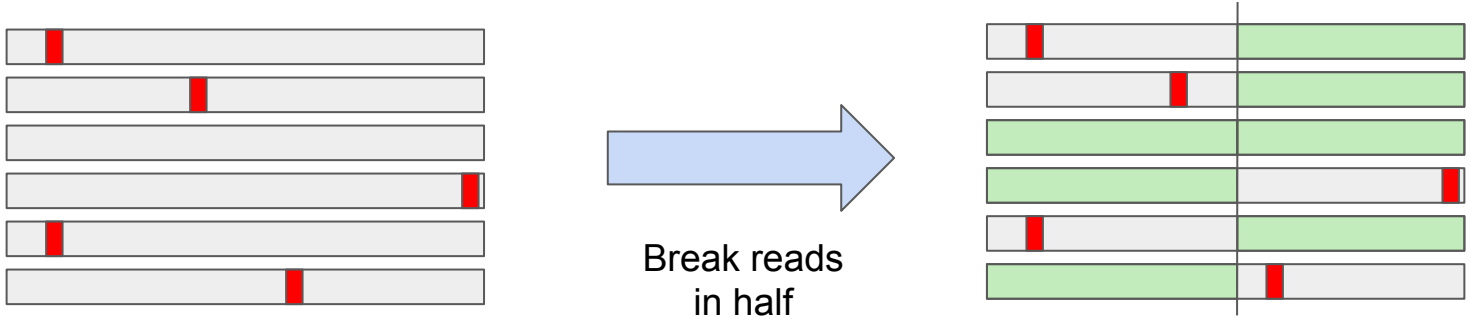
Source of Difficulty	Why is it harder?	Easier Version
Genome large	Lots of candidate matches	Restrict to some region of the genome
Sequencing error	Even “matches” aren’t exactly the same sequence	Require exact matches
Genome repetitive	Lots of locations can be close to the best match	Report any sufficiently good match

Easy Version: Given a sequence, where in the genome are the <u>exact</u> matches to it (if there are any)?

This lecture focuses on this version of the problem, and seed and extend is one heuristic for bridging the gap between this and the original problem.

Pigeonhole Principle

Suppose we could guarantee that for any read, the best match would have at most 1 mismatch.



Example reads
(mismatches in red)

Every read has at least one
half with no mismatches

This principle generalizes to more mismatches; if a read has at most k mismatches and we divide it into k+1 segments, there will be at least one mismatch-free segment.

Seed and Extend Technique

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

→ Ungapped alignment with FM Index

→ 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

→ SIMD dynamic programming aligner

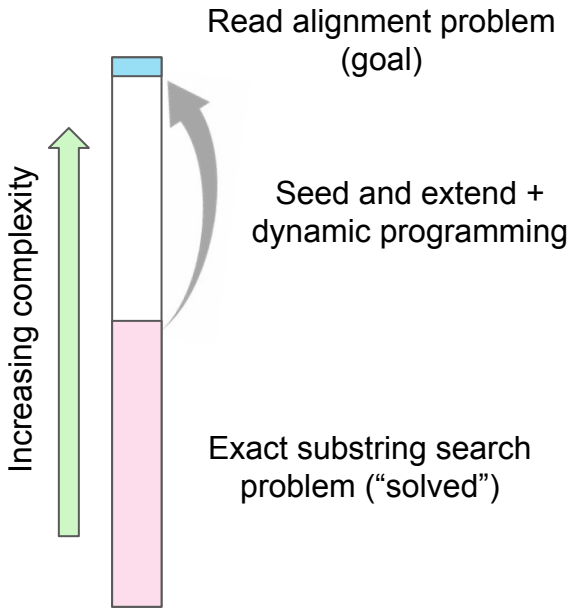
→ SAM alignments

```
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)

Areas of Improvement

- ➔ Improve exact substring search algorithms
- ➔ Extend substring search algorithms to work with some amount of mismatches
- ➔ Better heuristics to go from exact matches to full alignments



Brute Force Algorithm

Brute Force Algorithm

- 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)

Brute Force Reflections

Why check every position?

- GATTACA can't possibly start at positions 10-14 [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

Is this Good Enough?

How long would this algorithm take to align 30x short (100 bp) read data to the human genome on a single processor?

Is this Good Enough?

Genome size:
3,000,000,000 bp

Number of reads:
(3 billion bp) * (30x coverage) / (100 bp per read) = 900 million reads

Runtime per read:
n + m = (3 billion) + 100 = 3.0001 billion operation ~= 3 seconds/read

Total runtime:
3 seconds/read * 900 million read ~= 86 years

If we start this run right now on 100 processors in parallel, it'll finish right around the end of the calendar year!

Data Structures are Everywhere

A <u>data structure</u> is a way of organizing data so that the updates/queries you need for a specific application are more efficient.

Example: You have a list of numbers, and want to know how many times different numbers occur in the list. If you know in advance that you'll get many queries, how can you restructure the data so that each query is faster?

{11, 1, 3, 5, 17, 23, 12, 35, 54, 22, 19, 4, 31, 42, 12, 23, 1, 12, 14, 20, 1}

Data Structures Example

Example: You have a list of numbers, and want to know how many times different numbers occur in the list. If you know in advance that you'll get many queries, how can you restructure the data so that each query is faster?

{11, 1, 3, 5, 17, 23, 12, 35, 54, 22, 19, 4, 31, 42, 12, 23, 1, 12, 14, 20, 1}

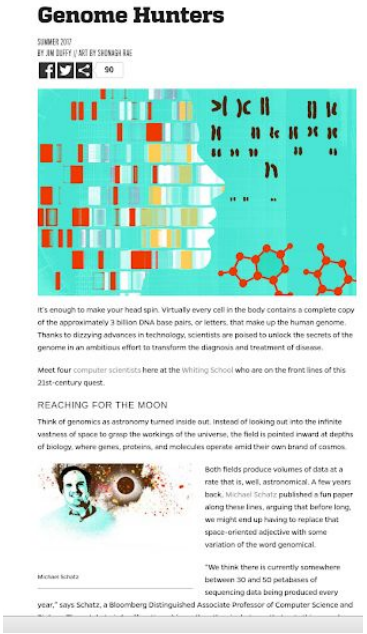
Frequency Array/Hash Table
Map each number to the number of times it occurs, and then just look up this number for each query.

Sorted Array
All of the copies of a number will be stored together, and algorithms like binary search can be used to quickly find where they are in O(log(n)) operations.

Suffix Arrays

Phone Books: A Data Structure for Text

- Suppose you are trying to call the Genome Hunters to help you analyze some genomes
  - We don't need to check every page to find “Genome Hunters”
  - Sorting alphabetically lets us immediately skip 96% (25/26) of our contacts list *without any loss in accuracy*



Genome Hunters

SUMMER 2017

BY JIM DUFFY | NOT BY SHONAGH RAE

90

It's enough to make your head spin. Virtually every cell in the body contains a complete copy of the approximately 3 billion DNA base pairs, or letters, that make up the human genome. Thanks to dizzying advances in technology, scientists are poised to unlock the secrets of the genome in an ambitious effort to transform the diagnosis and treatment of disease.

Meet four computer scientists here at the Whiting School who are on the front lines of this 21st-century quest.

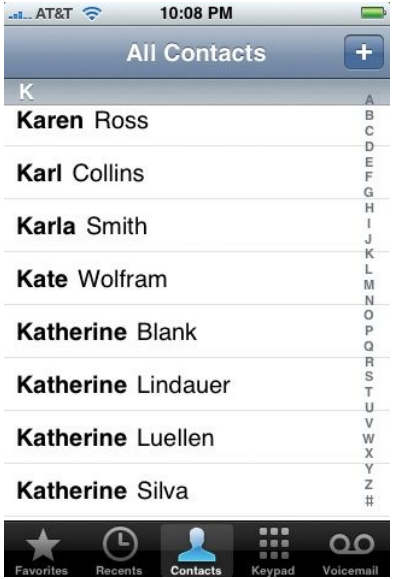
REACHING FOR THE MOON

Think of genomics as astronomy turned inside out. Instead of looking out into the infinite vastness of space to grasp the workings of the universe, the field is pointed inward at depths of biology, where genes, proteins, and molecules operate amid their own brand of cosmos.

Both fields produce volumes of data at a rate that is, well, astronomical. A few years back, Michael Schatz published a fun paper along these lines, arguing that before long, we might end up having to replace that space-oriented adjective with some variation of the word genomical.

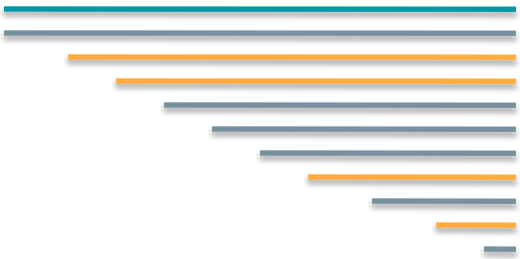
Michael Schatz

"We think there is currently somewhere between 30 and 50 petabases of sequencing data being produced every year," says Schatz, a Bloomberg Distinguished Associate Professor of Computer Science and

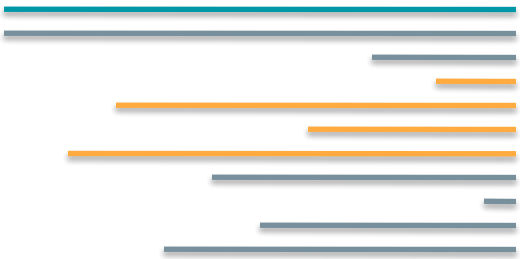


Suffix Arrays: Sorting a String

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



Split into n suffixes



Sort suffixes alphabetically

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
Lo	8	CC…	14
	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
Lo	8	CC…	14
	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;

	#	Sequence	Pos
	1	ACAGATTACC…	6
	2	ACC…	13
	3	AGATTACC…	8
	4	ATTACAGATTACC…	3
	5	ATTACC…	10
	6	C…	15
	7	CAGATTACC…	7
Lo	8	CC…	14
	9	GATTACAGATTACC…	2
	10	GATTACC…	9
Hi	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
Lo	8	CC…	14
	9	GATTACAGATTACC…	2
	10	GATTACC…	9
Hi	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!

	#	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 Hi	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: $n/(2^x) \leq 1$; $x = \lg_2(n)$ [32]

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

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: $n/(2^x) \leq 1$; $x = \lg_2(n)$ [32]

- Total Runtime: O(m lg n)

Can be reduced to O(m + lg n), or ~2 minutes on 30x human dataset, using an auxiliary data structure called the LCP array

Suffix Arrays in Practice

A few additional notes:

- Don’t actually store the suffixes - instead store their starting positions
- There are O(n) algorithms for construction
- Once the suffix array is made it can be used for all queries when mapping to the same reference (even for reads from different sequencing runs)

Downsides

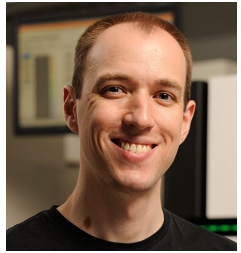
- Requires loading the entire suffix array into memory (high RAM)
- Binary search requires memory accesses which are far apart (bad for cache)

Burrows Wheeler Transform

Algorithmic challenge

How can we combine the speed of a suffix array O(m + lg(n)) (or even O(m)) with the size of a brute force analysis (n bytes)?

What would such an index look like?

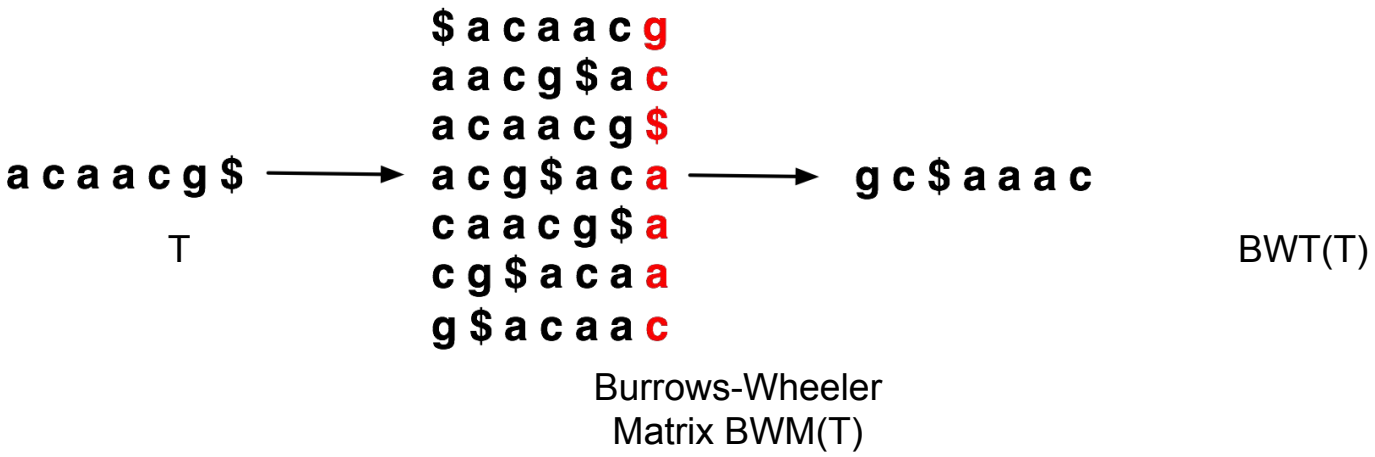


Bowtie: Ultrafast and memory efficient alignment of short DNA sequences to the human genome

Slides Courtesy of Ben Langmead

Burrows-Wheeler Transform

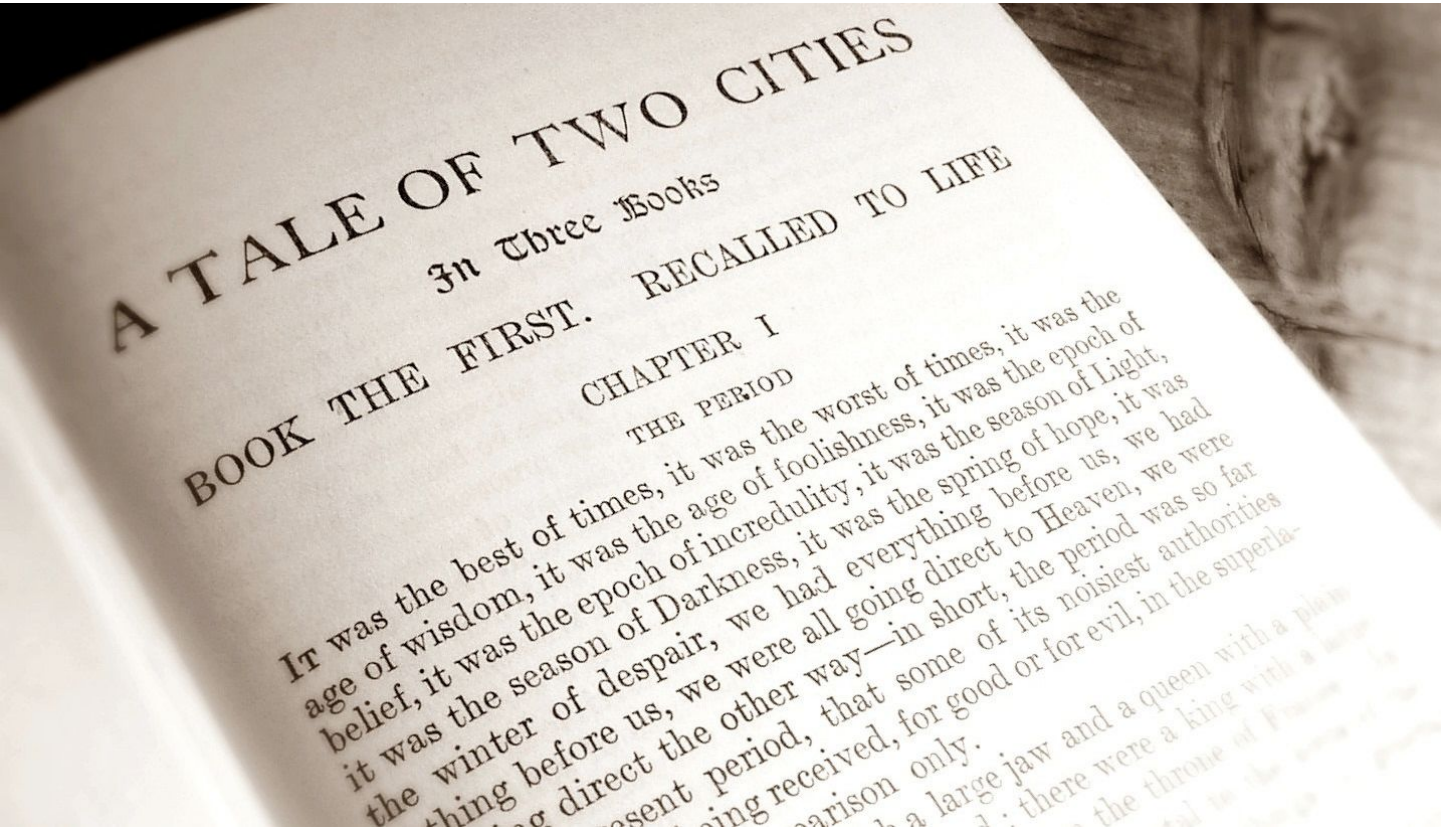
- Permutation of the characters in a text



- BWT(T) is the index for T

A block sorting lossless data compression algorithm.
Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation.* Technical Report 124

Run Length Encoding



Run Length Encoding

```
ref[614]:
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

Run Length Encoding:

- Replace a "run" of a character X with a single X followed by the length of the run
- GAAAAAAAATTACA => GA8T2ACA (reverse is also easy to implement)
- If your text contains numbers, then you will need to use a (slightly) more sophisticated encoding

Run Length Encoding

ref[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

rle(ref)[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_fo2lishnes2,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darknes2,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_al2_going_direct_to_Heaven,_we_were_al2_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_go2d_or_for_evil,_in_the_superlative_degre2_of_comparison_only.$
```

Run Length Encoding

ref[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

bwt[614]:

```
.dlmssftysesdtrsns_y__$_yfofeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,
,,nfrsdnnhereghettedndeteegeenstee,ssssst,esssnssffteedtttttttttr,,
,,eeefehh__p__fpDwwwwwwwwwwweehl_ew_____eoo_neeeoaaeoo____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo
oooooo____eennnnnnaai___ecc__tttttttttttttttttts_tsgltsLlvtt___hhoor
e_wrraddwlors_________r__lteirillre_ouaanooiioeoooooiiihkiiiiiio__iei
tsppioi_____ggnodsc_sss_gfhf_fffhwh_nsmo__uee_siooooaeeeeoo_ii
cgppeeaoaeooeesseuutetaaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciiIi
iiiiiisn_____oyoui__a_iiids__aiiaee_____tlar
```

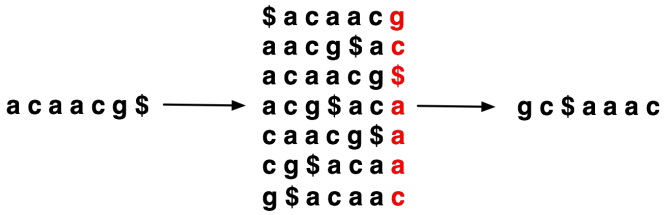
Run Length Encoding

ref[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

bwt[614]:

```
.dlmssftysesdtrsns_y__$_yfofeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,
,,nfrsdnnhereghettedndeteegeenstee,ssssst,esssnssffteedtttttttttr,,
,,eeefehh__p__fpDwwwwwwwwwwweehl_ew_____eoo_neeeoaaeoo____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo
oooooo____eennnnnnaai___ecc__tttttttttttttttttts_tsgltsLlvtt___hhoor
e_wrraddwlors_________r__lteirillre_ouaanooiioeoooooiiihkiiiiiio__iei
tsppioi_____________ggnodsc_sss_gfhf_fffhwh_nsmo__uee_siooaeeeeoo_ii
cgppeeaoaeooeesseuutetaaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciiIi
iiiiiisn_________________oyoui__a_iiids__aiiaee_____________________tlar
```



Why does the BWT tend to make runs in english text?

Run Length Encoding

bwt[614]:

```
.dlmssftysesdtrsns_y__$_yfofeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,
,,nfrsdnnhereghettedndeteegeenstee,ssssst,esssnssffteedtttttttttr,,
,,eeefehh__p__fpDwwwwwwwwwweehl_ew_____eoo_neeeoaaeoo____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo
oooooo____eennnnnnaai___ecc__tttttttttttttttttts_tsgltsLlvtt___hhoor
e_wrraddwlors_________r__lteirillre_ouaanooiioeoooooiiihkiiiiiio__iei
tsppioi_____ggnodsc_sss_gfhf_fffhwh_nsmo__uee_siooooaeeeeoo_ii
cgppeeaoaeooeesseuutetaaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciiIi
iiiiiisn_____oyoui__a_iiids__aiiaee_____tlar
```

rle(bwt)[464]:

```
.dlms2ftysesdtrsns_y_2$_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_2ei_in_2a
2ie_e3rei_hrs3nac2i2Ii7sn_15oyoui_2a_i3ds_2ai2ae2_21tlar
```

Run Length Encoding

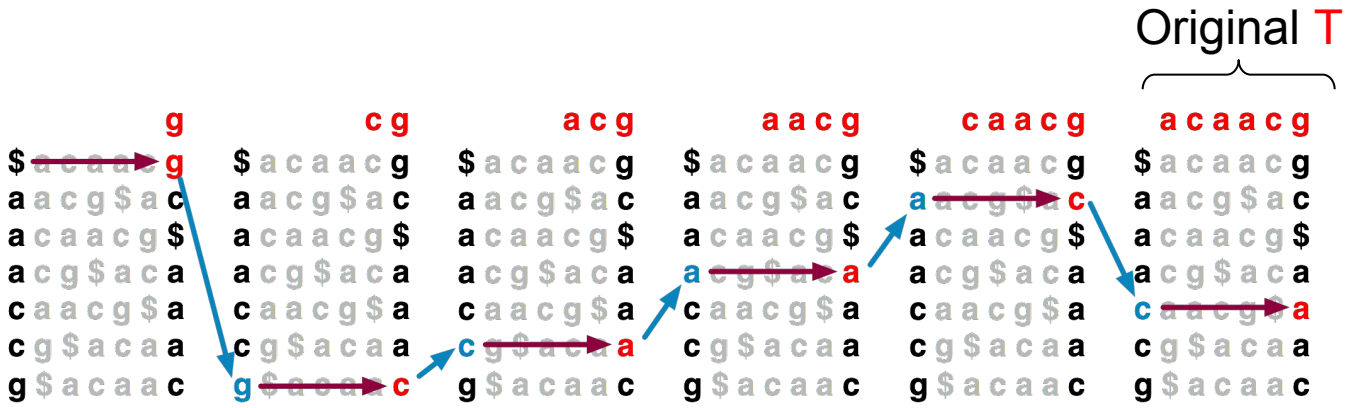
```
bwt[614]:
.dlmssftysesdtrsns_y__$_yfofeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,
,,nfrsdnnhereghettedndeteegeenstee,ssssst,esssnssffteedtttttttttr,,
,,eeefehh__p__fpDwwwwwwwwwweehl_ew_____eoo_neeeoaaeoo____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo
oooooo____eennnnnnaai___ecc__tttttttttttttttttts_tsgltsLlvtt___hhoor
e_wrraddwlors_________r__lteirillre_ouaanooiioeoooooiiihkiiiiiio__iei
tsppioi_____ggnodsc_sss_gfhf_fffhwh_nsmo__uee_siooooaeeeeoo_ii
cgppeeaoaeooeesseuutetaaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciiIi
iiiiiisn_____oyoui__a_iiids__aiiaee_____tlar
```

```
rle(bwt)[464]:
.dlms2ftysesdtrsns_y_2$_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_2ei_in_2a
2ie_e3rei_hrs3nac2i2Ii7sn_15oyoui_2a_i3ds_2ai2ae2_21tlar
```

Saved 614-464 = 150 bytes (24%) with zero loss of information!

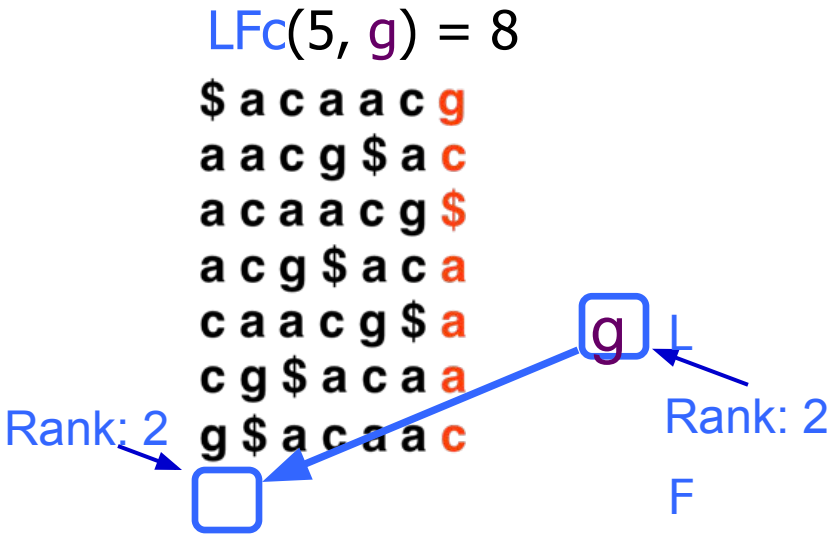
Burrows-Wheeler Transform

- Recreating T from BWT(T)
  - Start in the first row and apply **LF** repeatedly, accumulating predecessors along the way



BWT Exact Matching

- **LFc**(r, c) does the same thing as **LF**(r) but it ignores r's actual final character and "pretends" it's c:

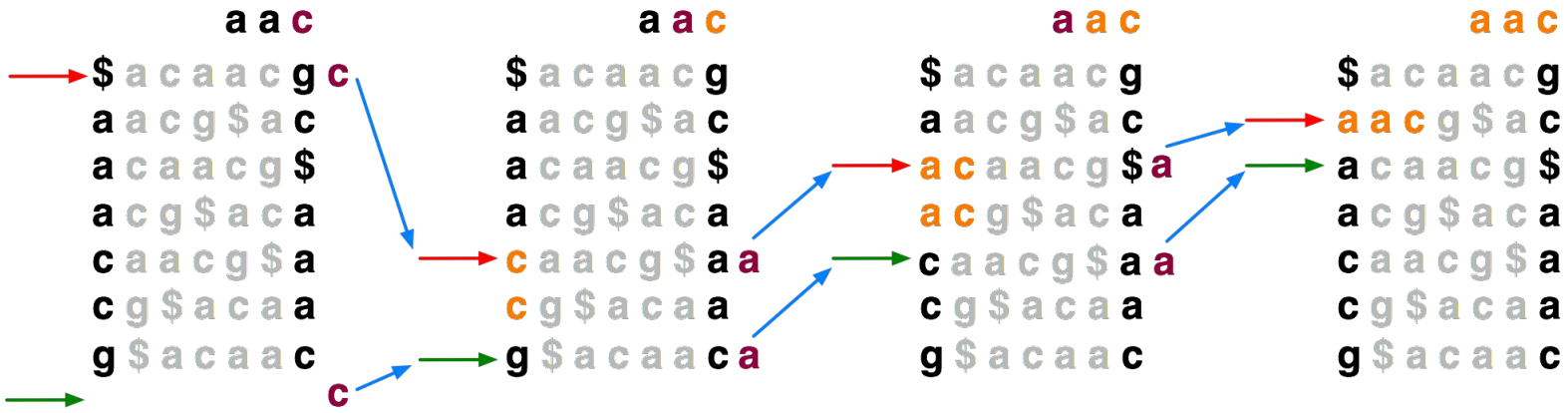


BWT Exact Matching

- Start with a range, (**top**, **bot**) encompassing all rows and repeatedly apply modified LF-mapping:

  top = **LFc**(**top**, **qc**); **bot** = **LFc**(**bot**, **qc**)

  qc = the next character to the left in the query

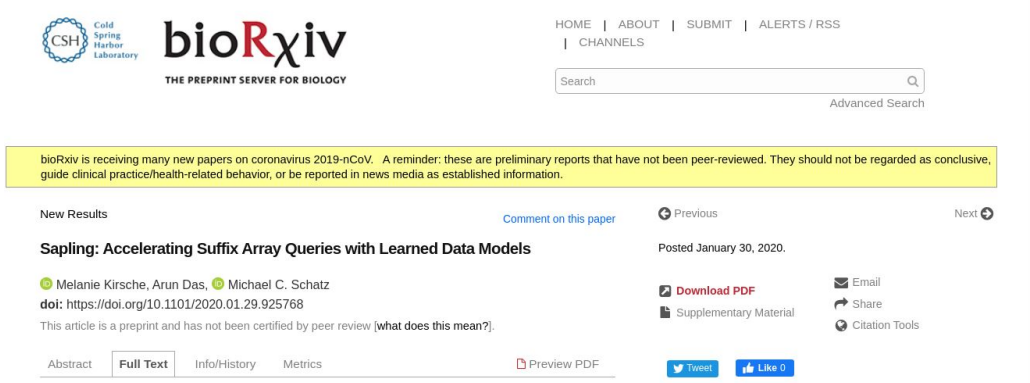


Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*

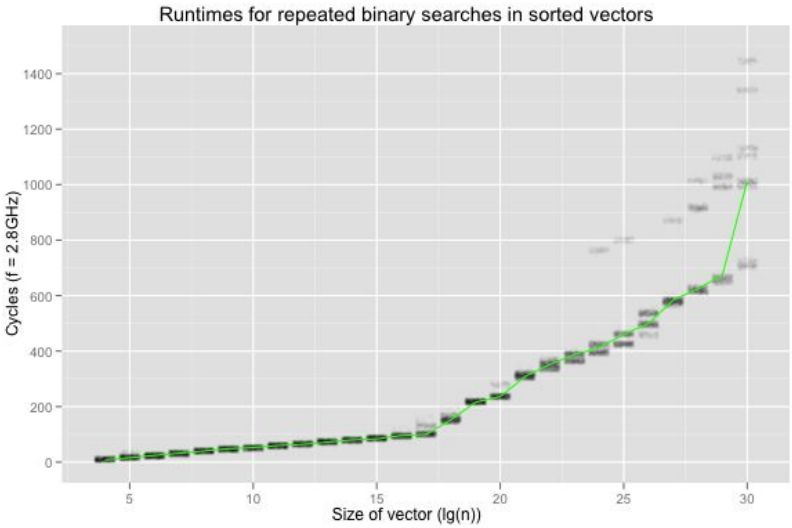
BWT Notes

- Query is O(m) because we apply m LF-mappings
- We still need a few other data structures
  - Cumulative frequency arrays
  - Suffix array
- But since we're getting the suffix array range with LF-mappings, we can get away with storing only a sample of it

Sapling: Accelerated Suffix Array Queries

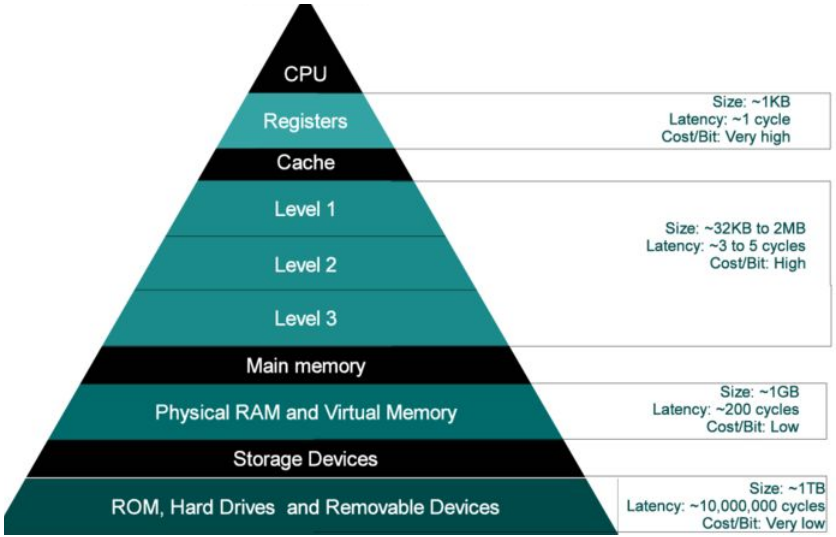


Binary Search and Caching

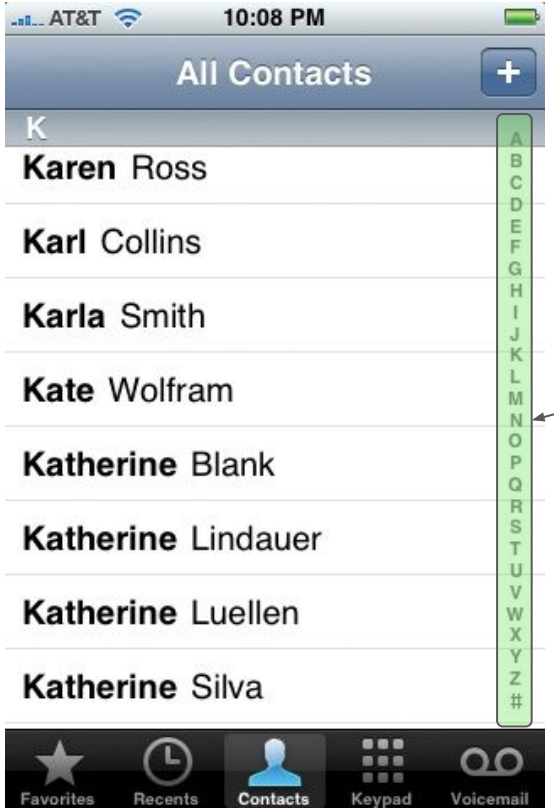


"*Binary Search is a pathological case for caches*" (Khuong, Paul)

Memory Hierarchy



Back to the Phone Book

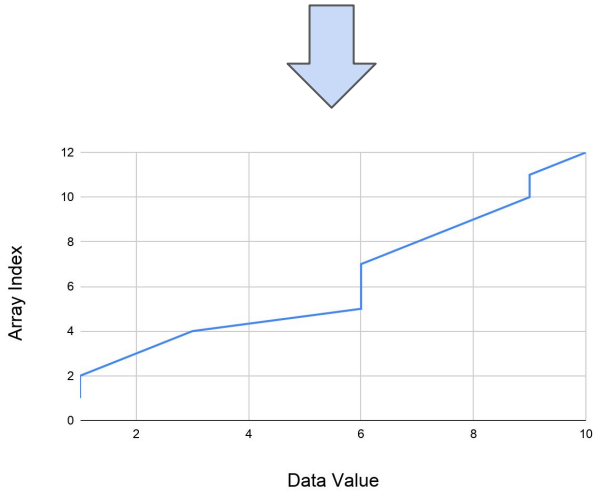


If your query starts with Z, using these markers is probably faster than binary search

How is the contact list implemented?

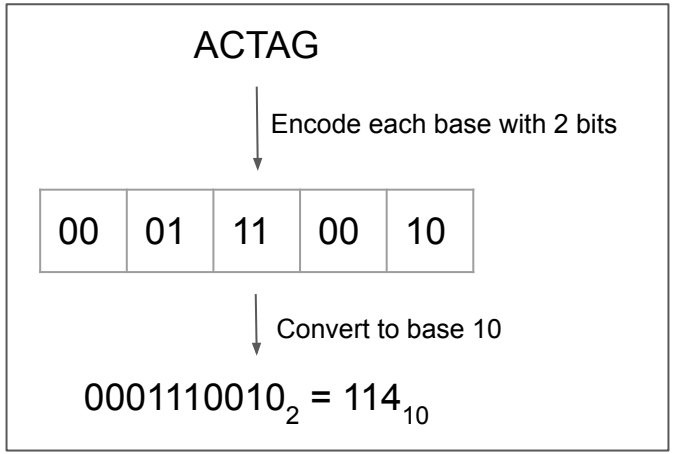
Data Structures as Functions

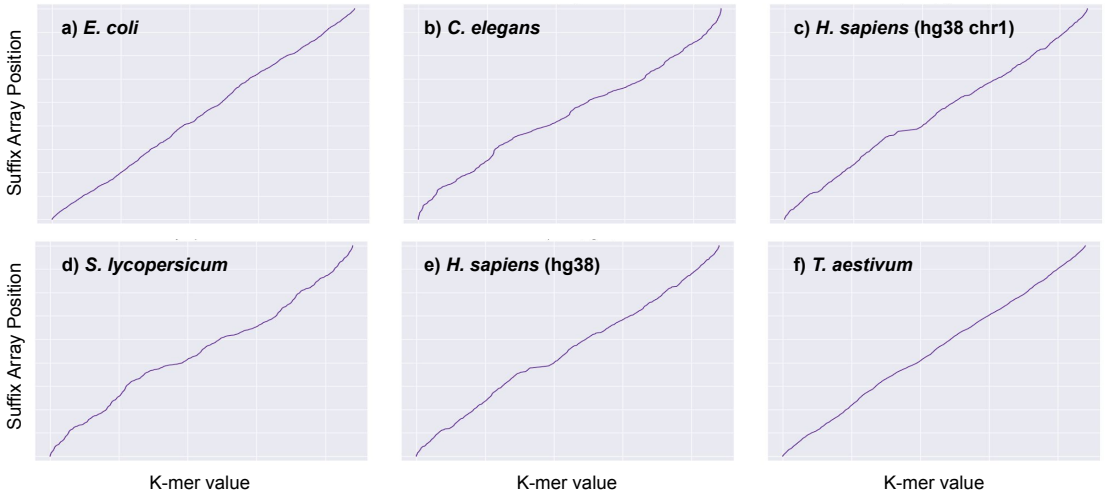
Index	1	2	3	4	5	6	7	8	9	10	11	12
Value	1	1	2	3	6	6	6	7	8	9	9	10



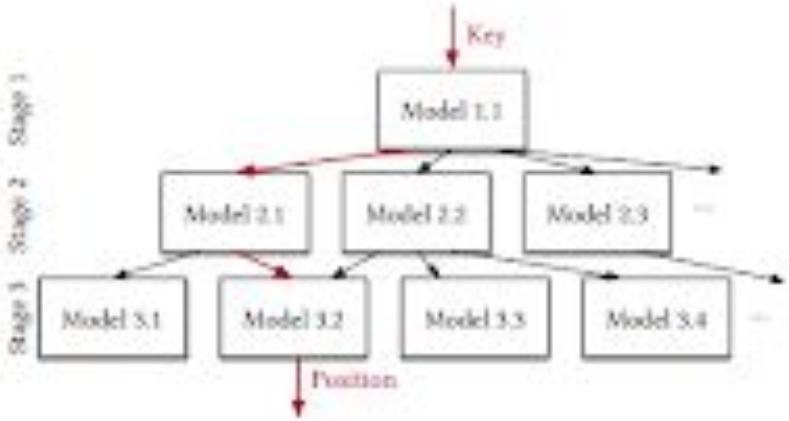
Goal: Approximate the function and use that to speed up queries

Suffix Array Position Function





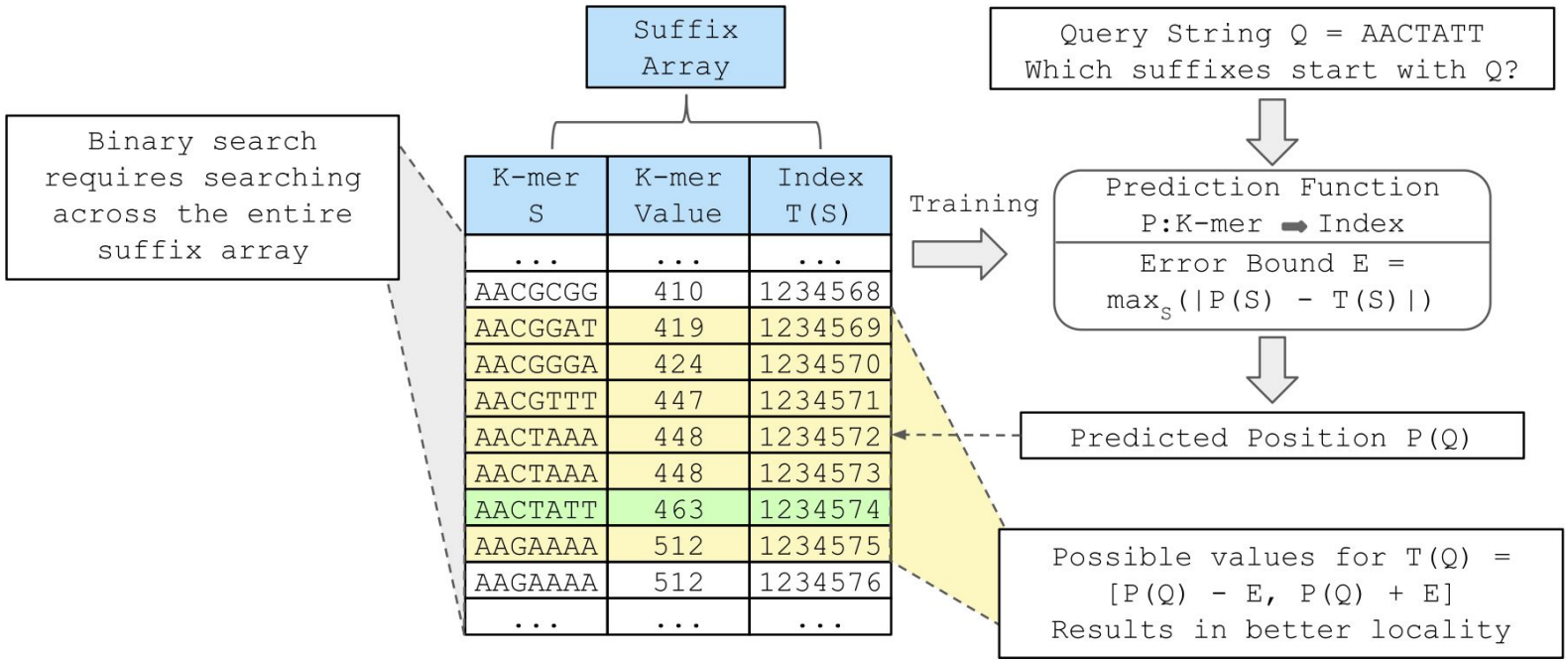
Learned Index Structures



The Case for Learned Index Structures (Kraska et. al., 2018)

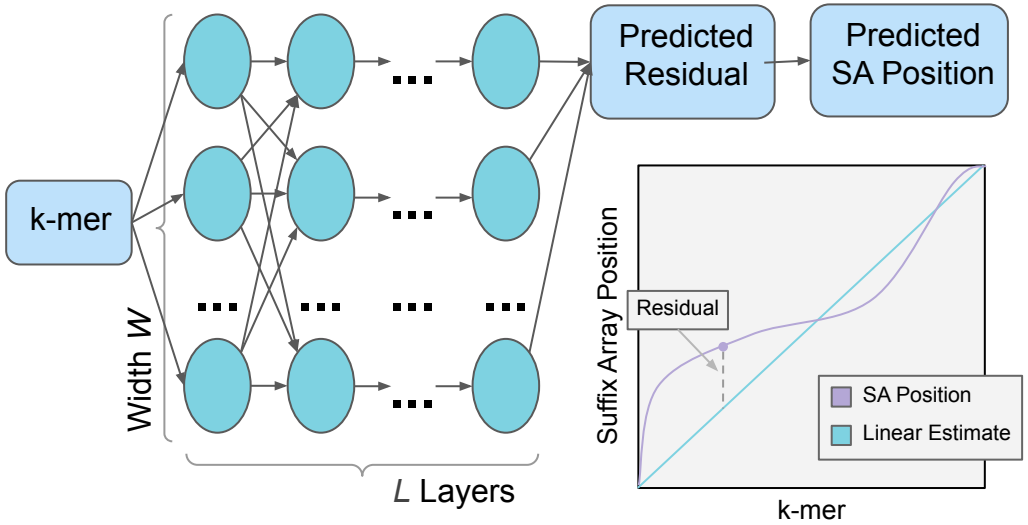
1. **Train** a model on all data points
2. Predict the position of all data points and **determine maximum prediction error** E
3. When performing a **lookup**, compute the predicted position, and search within the range [p - E, p + E]

Applying to Suffix Arrays

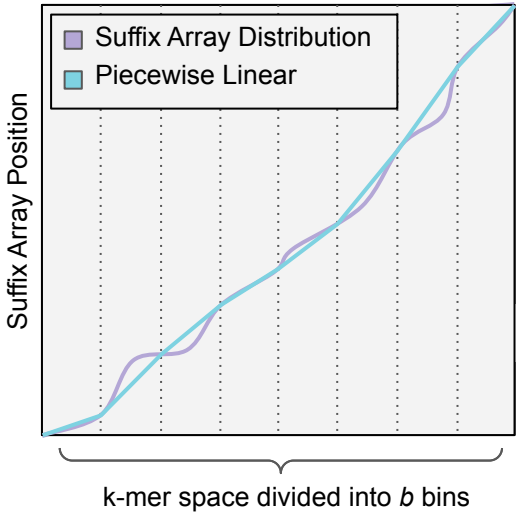


Modeling the function

a) ANN Architecture



b) Piecewise Linear Architecture

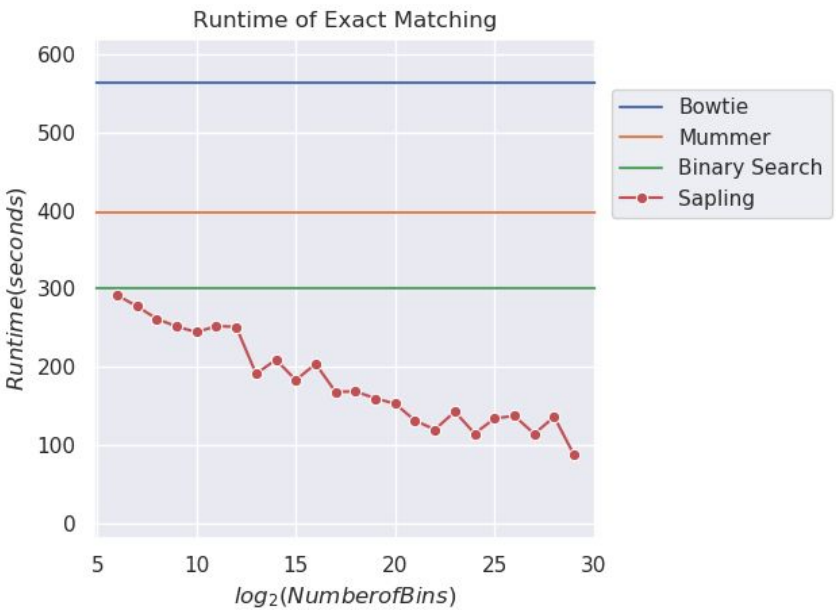


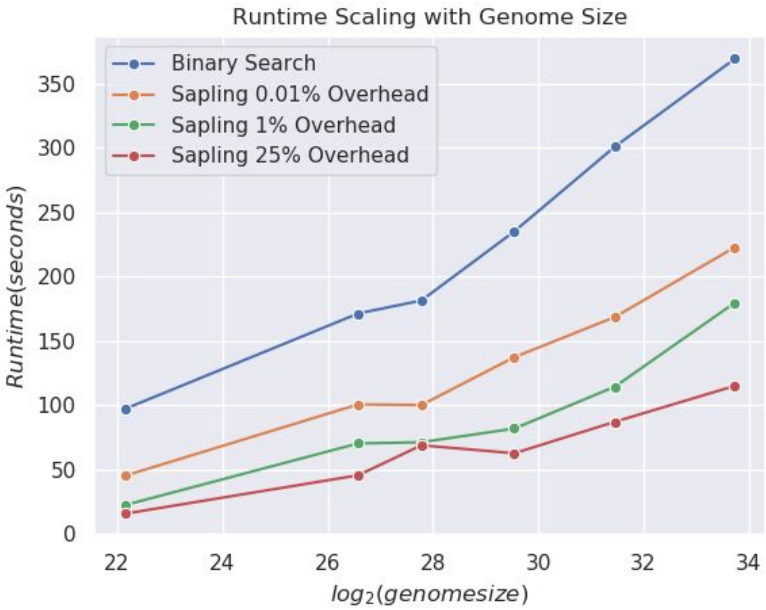
Modeling Results

Table 1. Summary of performance and model complexities for several PWL and ANN architectures.

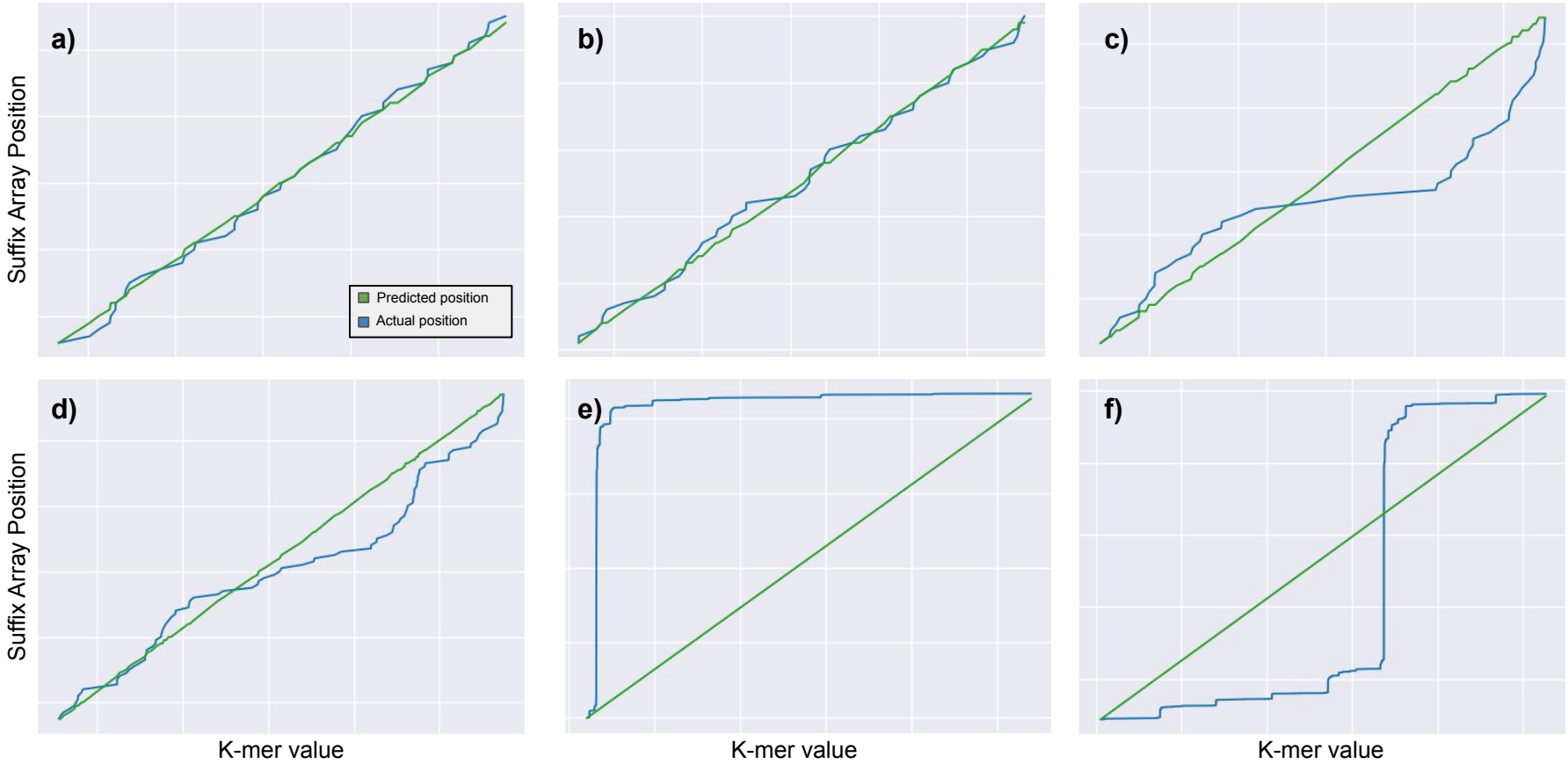
Model Type	Piecewise Linear	Piecewise Linear	Piecewise Linear	Neural Network	Neural Network	Neural Network
Number of Buckets	16k	256k	2m	1k	16k	16K
Width x Depth	N/A	N/A	N/A	32 x 1	32 x 1	128 x 2
Median Error	899	68	14	900	131	56
95th Percentile Error	7,858	1,579	653	4238	853	463
Maximum Error	263,165	180,453	135,664	45,839	24,081	13,264
Memory Overhead	256 KB	4 MB	32 MB	8 MB	131 MB	1245 MB

Runtime Results





What makes certain bins behave poorly?



Future Work

- ➔ Extend into full aligner (currently just a proof-of-concept with very simple seed-and-extend heuristics)
- ➔ Fitting other types of functions between simple piecewise linear and complex neural network
- ➔ Apply to other data structures (e.g., FM-index)
- ➔ Apply to other problems in genomics

Conclusions

- Two key components to read alignment
  - Exact substring search
  - Seed-and-extend heuristics

- Data structures enable fast algorithms for exact substring search
  - Suffix Arrays
  - FM-Index
  - Sapling
  - Many others!

- Even theoretically optimal algorithms may not be optimal in practice