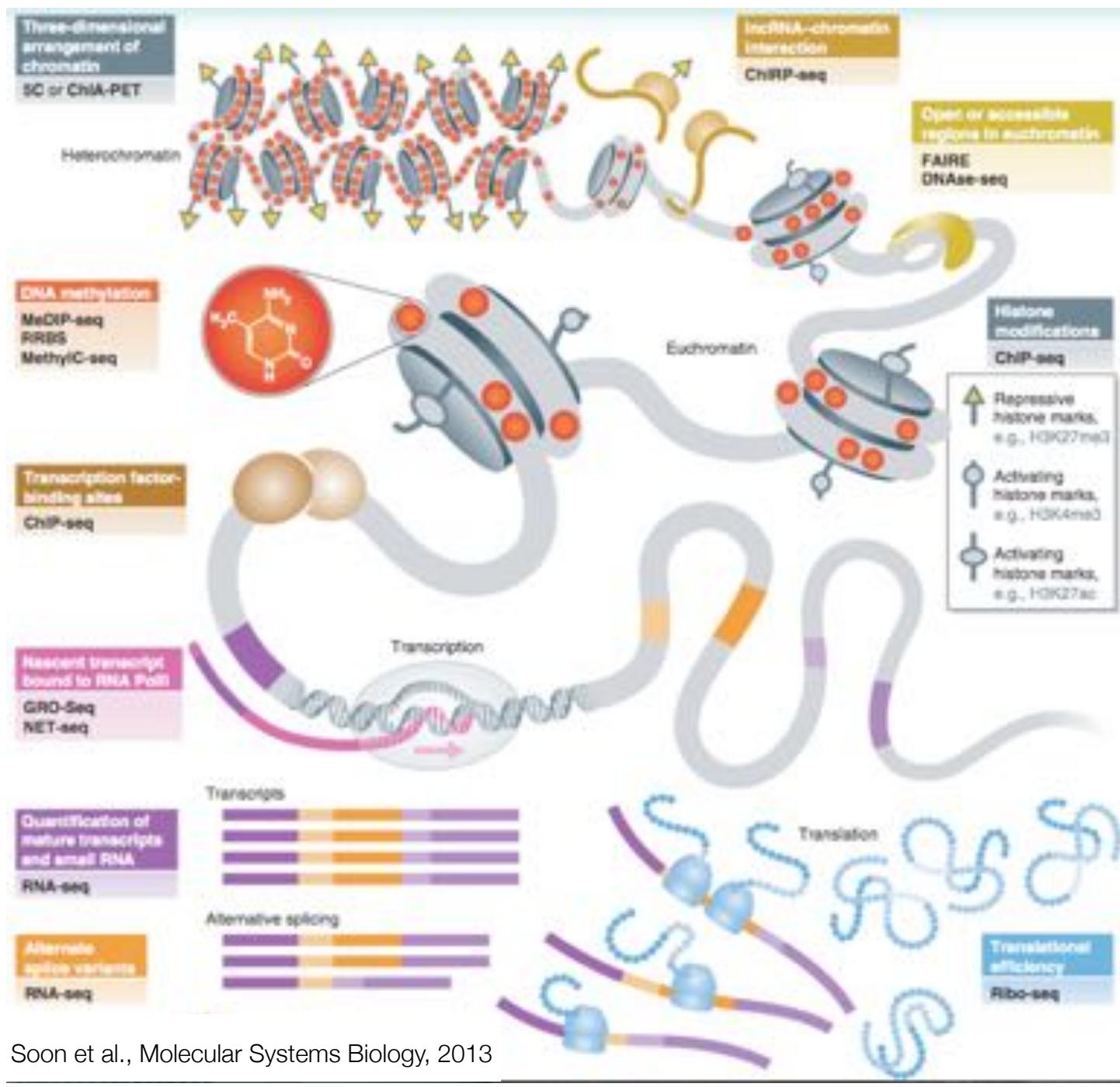


Sorting, Searching, & Aligning

Michael Schatz

Bioinformatics Lecture I
Quantitative Biology 2013





Sequencing Assays

1. Gregory E. Crawford et al., “Genome-wide Mapping of DNase Hypersensitive Sites Using Massively Parallel Signature Sequencing (MPSS),” *Genome Research* 16, no. 1 (January 1, 2006): 123–131, doi:10.1101/gr.4074106.
2. David S. Johnson et al., “Genome-Wide Mapping of in Vivo Protein-DNA Interactions,” *Science* 316, no. 5830 (June 8, 2007): 1497–1502, doi:10.1126/science.1141319.
3. Tarjei S. Mikkelsen et al., “Genome-wide Maps of Chromatin State in Pluripotent and Lineage-committed Cells,” *Nature* 448, no. 7153 (August 2, 2007): 553–560, doi:10.1038/nature06008.
4. Nathan A. Baird et al., “Rapid SNP Discovery and Genetic Mapping Using Sequenced RAD Markers,” *PLoS ONE* 3, no. 10 (October 13, 2008): e3376, doi:10.1371/journal.pone.0003376.
5. Leighton J. Core, Joshua J. Waterfall, and John T. Lis, “Nascent RNA Sequencing Reveals Widespread Pausing and Divergent Initiation at Human Promoters,” *Science* 322, no. 5909 (December 19, 2008): 1845–1848, doi:10.1126/science.1162228.
6. Thomas A. Down et al., “A Bayesian Deconvolution Strategy for Immunoprecipitation-based DNA Methylome Analysis,” *Nature Biotechnology* 26, no. 7 (July 2008): 779–785, doi:10.1038/nbt1414.
7. Ali Mortazavi et al., “Mapping and Quantifying Mammalian Transcriptomes by RNA-Seq,” *Nature Methods* 5, no. 7 (July 2008): 621–628, doi:10.1038/nmeth.1226.
8. Alayne L. Brunner et al., “Distinct DNA Methylation Patterns Characterize Differentiated Human Embryonic Stem Cells and Developing Human Fetal Liver,” *Genome Research* 19, no. 6 (June 1, 2009): 1044–1056, doi:10.1101/gr.088773.108.
9. Melissa J. Fullwood et al., “An Oestrogen-receptor- α -bound Human Chromatin Interactome,” *Nature* 462, no. 7269 (November 5, 2009): 58–64, doi:10.1038/nature08497.
10. Jay R. Hesselberth et al., “Global Mapping of protein-DNA Interactions in Vivo by Digital Genomic Footprinting,” *Nature Methods* 6, no. 4 (April 2009): 283–289, doi:10.1038/nmeth.1313.
11. Nicholas T. Ingolia et al., “Genome-Wide Analysis in Vivo of Translation with Nucleotide Resolution Using Ribosome Profiling,” *Science* 324, no. 5924 (April 10, 2009): 218–223, doi:10.1126/science.1168978.
12. Gemma C. Langridge et al., “Simultaneous Assay of Every *Salmonella Typhi* Gene Using One Million Transposon Mutants,” *Genome Research* (October 13, 2009), doi:10.1101/gr.097097.109.
13. Erez Lieberman-Aiden et al., “Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome,” *Science* 326, no. 5950 (October 9, 2009): 289–293, doi:10.1126/science.1181369.
14. Ryan Lister et al., “Human DNA Methylomes at Base Resolution Show Widespread Epigenomic Differences,” *Nature* 462, no. 7271 (November 19, 2009): 315–322, doi:10.1038/nature08514.
15. Andrew M. Smith et al., “Quantitative Phenotyping via Deep Barcode Sequencing,” *Genome Research* (July 21, 2009), doi:10.1101/

Short Read Applications

- Genotyping: Identify Variations

A sequence of DNA short reads. The first seven lines are aligned vertically, showing variations at specific positions highlighted by red boxes. The eighth line is a reference sequence. A blue box highlights the last two lines, which are identical.

...CCATAG	TATGCGCCC	CGGAATT	GGTATAC...
...CCAT	CTATATGCG	TCGGAAATT	CGGTATAC
...CCAT	GGCTATATG	CTATCGGAAA	GCGGTATA
...CCA	AGGCTATAT	CCTATCGGA	TTGCGGTA C...
...CCA	AGGCTATAT	GCCCTATCG	TTTGCGGT C...
...CC	AGGCTATAT	GCCCTATCG	AAATTTCGC ATAC...
...CC	TAGGCTATA	GCGCCCTA	AAATTTCGC GTATAC...
...CCATAGGCTATATGCGCCCTATCGGC	CAATTGCGGTATAC...		

- *-seq: Classify & measure significant peaks

A sequence of DNA short reads. The first seven lines are aligned vertically, showing variations at specific positions. The eighth line is a reference sequence. A pink box highlights the first seven lines, indicating they are significant peaks.

...CC	GAAATTTC	GGAAATTG	CGGAAATT	CGGAAATT	TCGGAAATT	CTATCGGAAA	CCTATCGGA	TTTGCGGT	
...CCATAGGCTATATGCGCCCTATCGGC	CAATTGCGGTATAC...								

Exact Matching Review & Overview

Where is GATTACA in the human genome?

Brute Force
(3 GB)

BANANA
BAN
ANA
NAN
ANA

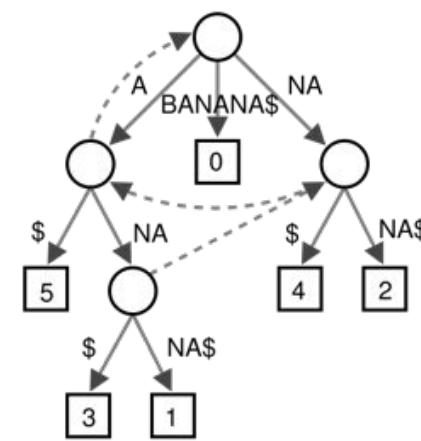
Naive
Slow & Easy

Suffix Array
(>15 GB)

6	\$
5	A\$
3	ANA\$
1	ANANA\$
0	BANANA\$
4	NA\$
2	NANA\$

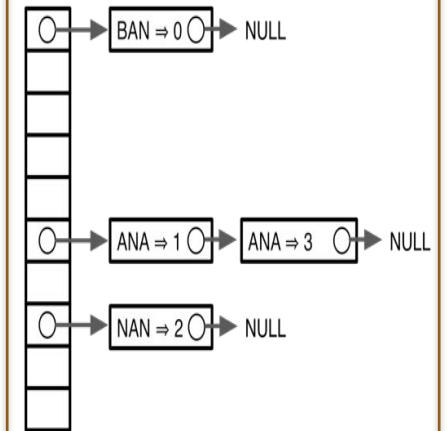
Vmatch, PacBio Aligner
Binary Search

Suffix Tree
(>51 GB)



MUMmer, MUMmerGPU
Tree Searching

Hash Table
(>15 GB)



BLAST, MAQ, ZOOM,
RMAP, CloudBurst
Seed-and-extend

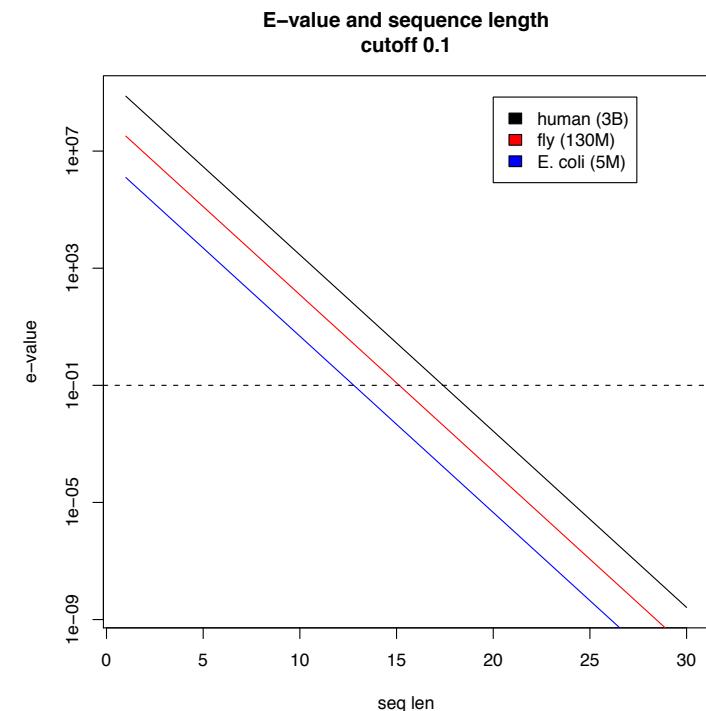
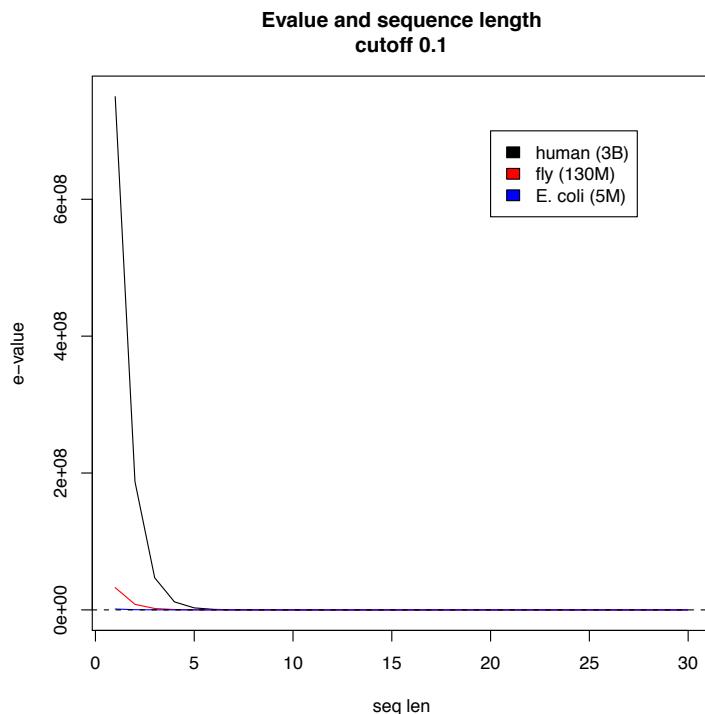
*** These are general techniques applicable to any search problem ***

Expected Occurrences

The expected number of occurrences (e-value) of a given sequence in a genome depends on the length of the genome and inversely on the length of the sequence

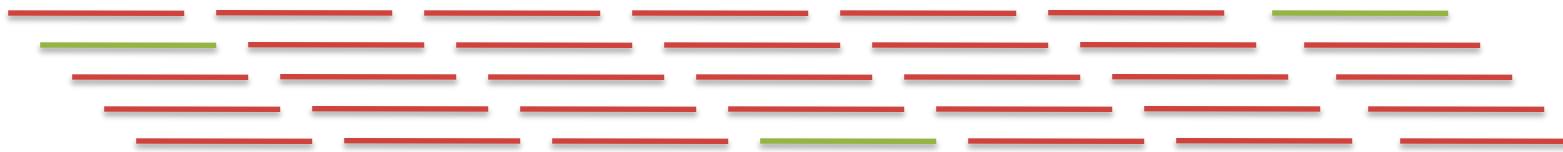
- 1 in 4 bases are G, 1 in 16 positions are GA, 1 in 64 positions are GAT
- 1 in 16,384 should be GATTACA
- $E=(n-m+1)/(4^m)$

[183,105 expected occurrences]



[Challenge Question: What is the expected distribution & variance?]

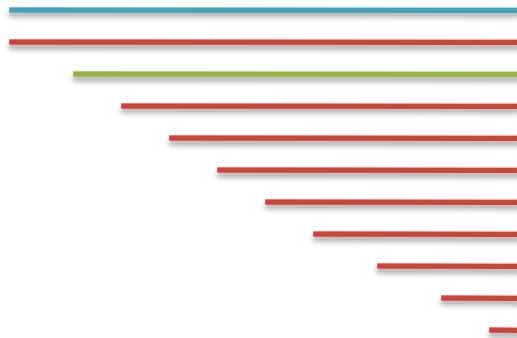
I. Brute Force



- 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: $(n-m+1) * m$ [2IB]
- 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?]

2. Suffix Arrays

- 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

Searching the Index

- **Strategy 2: Binary search**
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - Lo = 1; 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
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
 - $\text{Lo} = 1; \text{Hi} = 15; \text{Mid} = (1+15)/2 = 8$
 - Middle = Suffix[8] = CC

#	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

Lo
→

Hi
→

Searching the Index

- **Strategy 2: Binary search**
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $\text{Lo} = 1; \text{Hi} = 15; \text{Mid} = (1+15)/2 = 8$
 - Middle = Suffix[8] = CC
=> Higher: Lo = Mid + 1

#	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;

#	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

Lo
→

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: $Lo = Mid + 1$
 - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
 - 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
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo

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
 - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
 - 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
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo

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; 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

Lo

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; 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;

#	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

Lo
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; 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
9	GATTACA GATTACC...	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!

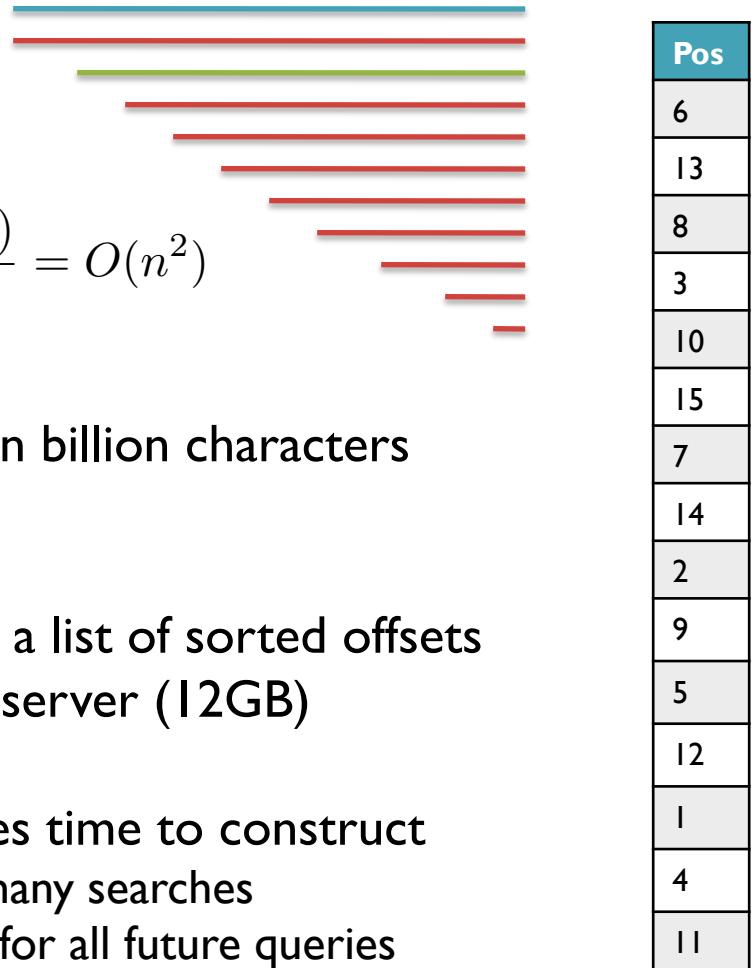
- Looking up a query loops 32 times instead of 3B

[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?]

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



- 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

TGATTACAGATTACC

Sorting

Quickly sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[How do you do it?]

6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19

6, 13, 14, 29, 31, 39, 64, 78, 50, 63, 61, 19

6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61

6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61

6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61

6, 13, 14, 19, 29, 31, 39, 50, 64, 78, 63, 61

6, 13, 14, 19, 29, 31, 39, 50, 61, 64, 78, 63

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

Sorting

Quickly sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[How do you do it?]

6, 14, 29, 31, 39, 64, 78, 50, 13, 63, 61, 19

6, 13, 14, 29, 31, 39, 64, 78, 50, 63, 61, 19

6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61

6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61

6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61

6, 13, 14, 19, 29, 31, 39, 50, 64, 78, 63, 61

6, 13, 14, 19, 29, 31, 39, 50, 61, 64, 78, 63

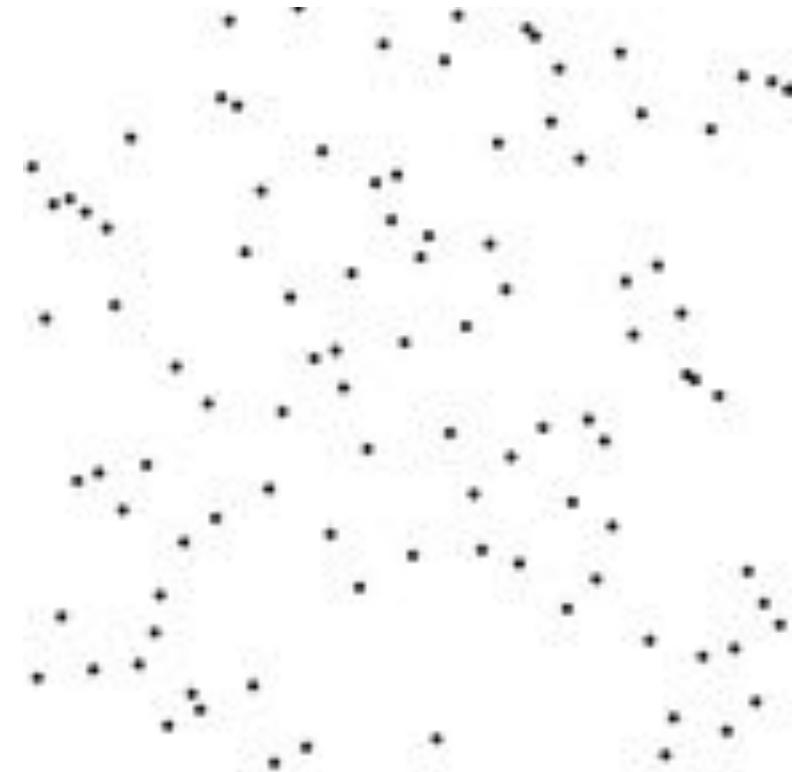
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78



Selection Sort Analysis

- Selection Sort (Input: list of n numbers)

```
for pos = 1 to n
```

```
    // find the smallest element in [pos, n]
```

```
    smallest = pos
```

```
    for check = pos+1 to n
```

```
        if (list[check] < list[smallest]): smallest = check
```

```
    // move the smallest element to the front
```

```
    tmp = list[smallest]
```

```
    list[pos] = list[smallest]
```

```
    list[smallest] = tmp
```

- Analysis

$$T = n + (n - 1) + (n - 2) + \dots + 3 + 2 + 1 = \sum_{i=1}^n i = \frac{n(n + 1)}{2} = O(n^2)$$

- Outer loop: pos = 1 to n

- Inner loop: check = pos to n

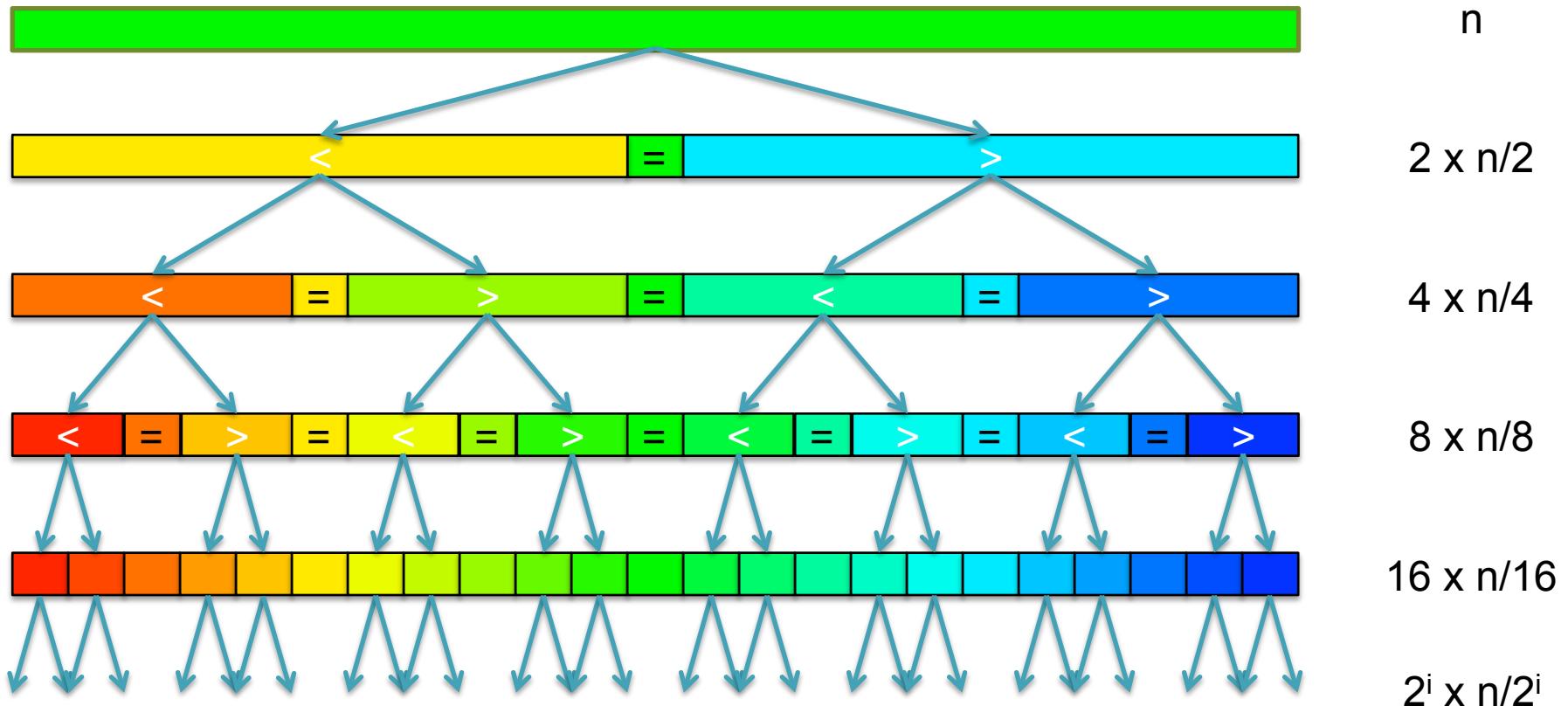
- Running time: Outer * Inner = $O(n^2)$

[4.5 Billion Billion]

[Challenge Questions: Why is this slow? / Can we sort any faster?]

Divide and Conquer

- Selection sort is slow because it rescans the entire list for each element
 - How can we split up the unsorted list into independent ranges?
 - Hint 1: Binary search splits up the problem into 2 independent ranges (hi/lo)
 - Hint 2: Assume we know the median value of a list



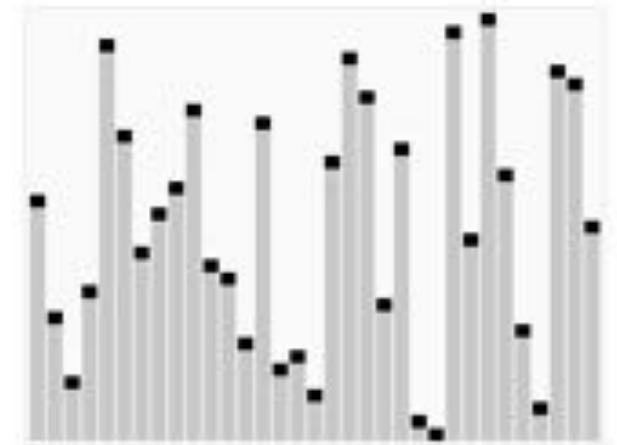
[How many times can we split a list in half?]

QuickSort Analysis

- QuickSort(Input: list of n numbers)
// see if we can quit
if (length(list)) <= 1): return list

// split list into lo & hi
pivot = median(list)
lo = {}; hi = {};
for (i = 1 to length(list))
 if (list[i] < pivot): append(lo, list[i])
 else: append(hi, list[i])

// recurse on sublists
return (append(QuickSort(lo), QuickSort(hi)))



<http://en.wikipedia.org/wiki/Quicksort>

- Analysis (Assume we can find the median in $O(n)$)

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ O(n) + 2T(n/2) & \text{else} \end{cases}$$

$$T(n) = n + 2\left(\frac{n}{2}\right) + 4\left(\frac{n}{4}\right) + \cdots + n\left(\frac{n}{n}\right) = \sum_{i=0}^{\lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{\lg(n)} n = O(n \lg n) \quad [\sim 94B]$$

QuickSort Analysis

- QuickSort(Input: list of n numbers)
// see if we can quit
if (length(list)) <= 1): return list

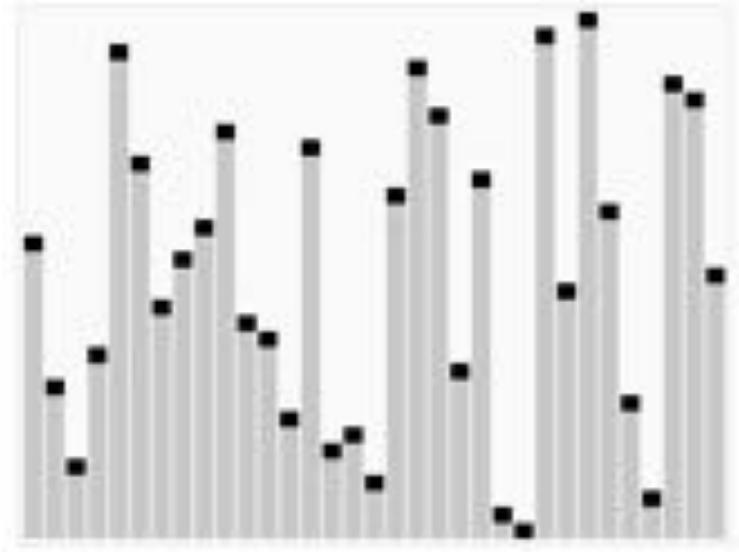
// split list into lo & hi
pivot = median(list)
lo = {}; hi = {};
for (i = 1 to length(list))
 if (list[i] < pivot): append(lo, list[i])
 else: append(hi, list[i])

// recurse on sublists
return (append(QuickSort(lo), QuickSort(hi)))

- Analysis (Assume we can find the median in $O(n)$)

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ O(n) + 2T(n/2) & \text{else} \end{cases}$$

$$T(n) = n + 2\left(\frac{n}{2}\right) + 4\left(\frac{n}{4}\right) + \cdots + n\left(\frac{n}{n}\right) = \sum_{i=0}^{\lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{\lg(n)} n = O(n \lg n) \quad [\sim 94B]$$



<http://en.wikipedia.org/wiki/Quicksort>

QuickSort in Python

`list.sort()`

- The goal of software engineering is to build libraries of correct reusable functions that implement higher level ideas
 - Build complex software out of simple components
 - Software tends to be 90% plumbing, 10% research
 - You still need to know how they work
 - Python requires an explicit representation of the strings

3. Sorting in Linear Time

- Can we sort faster than $O(n \lg n)$?
 - No – Not if we have to compare elements to each other
 - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100

3. Sorting in Linear Time

- Can we sort faster than $O(n \lg n)$?
 - No – Not if we have to compare elements to each other
 - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100

3. Sorting in Linear Time

- Can we sort faster than $O(n \lg n)$?
 - No – Not if we have to compare elements to each other
 - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

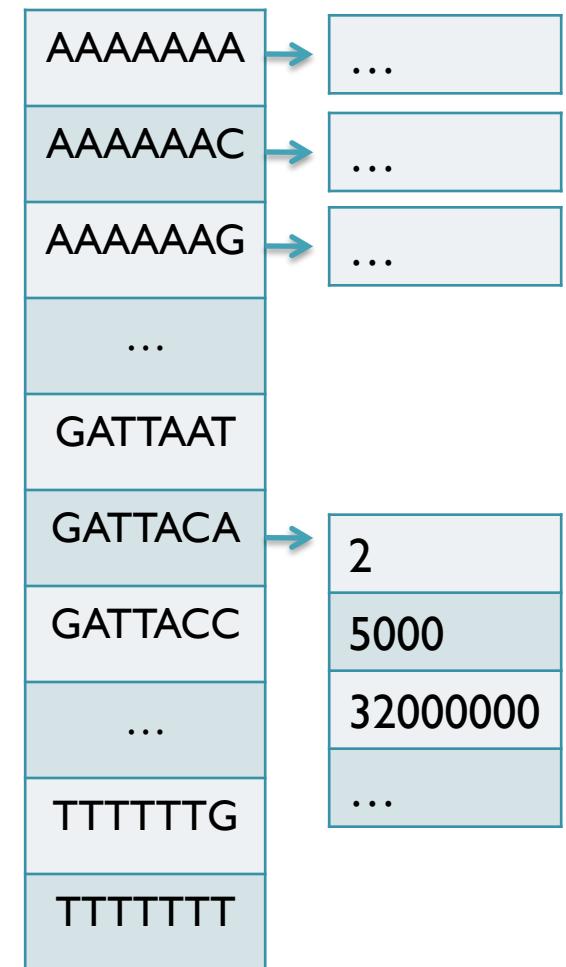
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100

6,13,14,19,29,31,39,50,61,63,64,78

```
for(i = 1 to 100) { cnt[i] = 0; }
for(i = 1 to n) { cnt[list[i]]++; }
for(i = 1 to 100) { while (cnt[i] > 0){print i; cnt[i]--}} [3B instead of 94B]
```

4. Hashing

- Where is GATTACA in the human genome?
 - Build an inverted index (table) of every kmer in the genome
- How do we access the table?
 - We can only use numbers to index
 - `table[GATTACA] <-` error, does not compute
 - Encode sequences as numbers
 - Simple: A = 0, C = 1, G = 2, T = 3
 - GATTACA = 2 0 3 3 0 1 0
 - Smart: A = 00_2 , C = 01_2 , G = 10_2 , T = 11_2
 - GATTACA = $10\ 00\ 11\ 11\ 00\ 01\ 00_2 = 9156_{10}$
- Running time
 - Construction: $O(n)$
 - Lookup: $O(1) + O(z)$
 - Sorts the genome mers in linear time



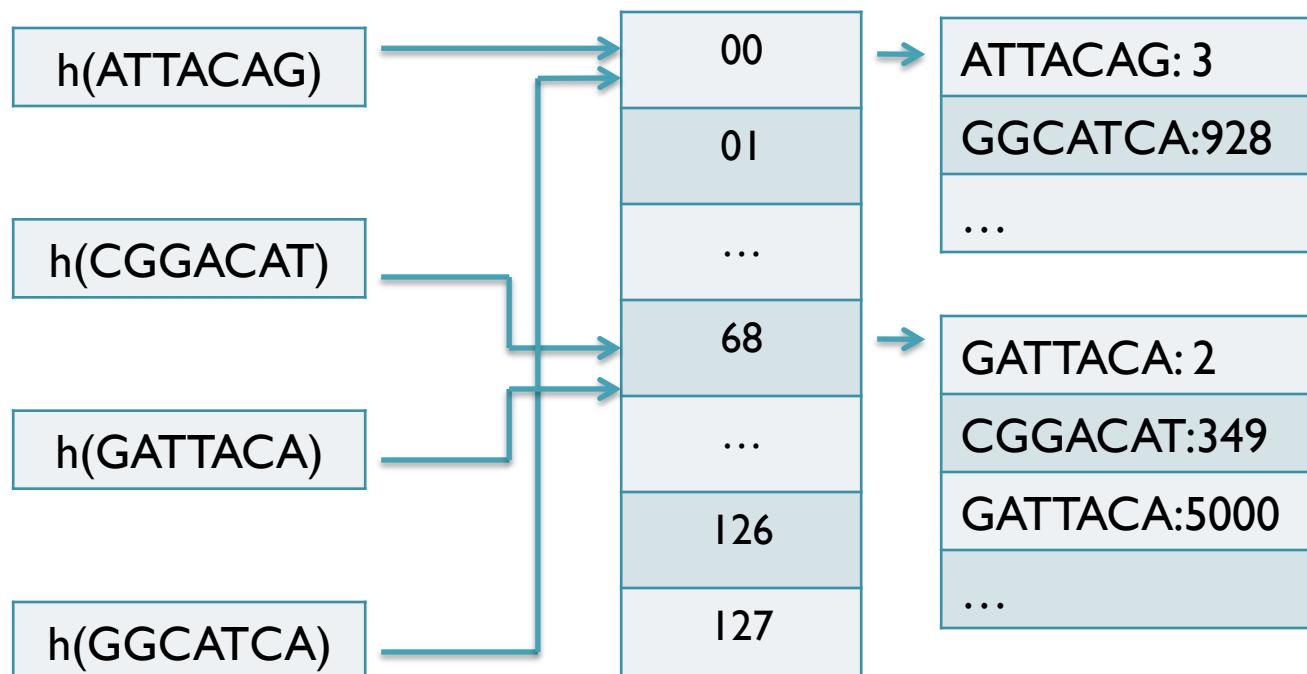
Hash Tables and Hash Functions

- Number of possible sequences of length $k = 4^k$
 - $4^7 = 16,384$ (easy to store)
 - $4^{20} = 1,099,511,627,776$ (impossible to directly store in RAM)
 - There are only 3B 20-mers in the genome
 - ⇒ Even if we could build this table, 99.7% will be empty
 - ⇒ But we don't know which cells are empty until we try
- Use a hash function to shrink the possible range
 - Maps a number n in $[0, R]$ to h in $[0, H]$
 - » Use 128 buckets instead of 16,384, or 1B instead of 1T
 - Division: $\text{hash}(n) = H * n / R;$
 - » $\text{hash(GATTACA)} = 128 * 9156 / 16384 = 71$
 - Modulo: $\text{hash}(n) = n \% H$
 - » $\text{hash(GATTACA)} = 9156 \% 128 = 68$

[What properties do we want in a hash functions?]

Hash Table Lookup

- By construction, multiple keys have the same hash value
 - Store elements with the same key in a bucket chained together
 - A good hash evenly distributes the values: R/H have the same hash value
 - Looking up a value scans the entire bucket
 - Slows down the search as a function of the hash table load
 - Warning: This complexity is usually hidden in the hash table code



[How many elements do we expect per bucket?]



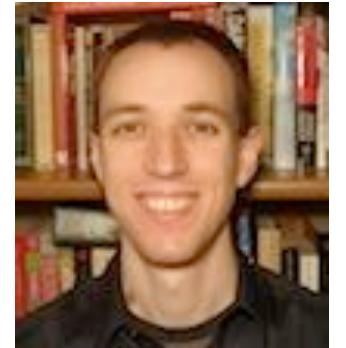
THE G-NOME PROJECT

Break

Algorithmic challenge

How can we combine the speed of a suffix array or hash table exact match ($O(|q|)$ or $O(\lg(n))$) with the size of a brute force analysis (n bytes)?

What would such an index look like?

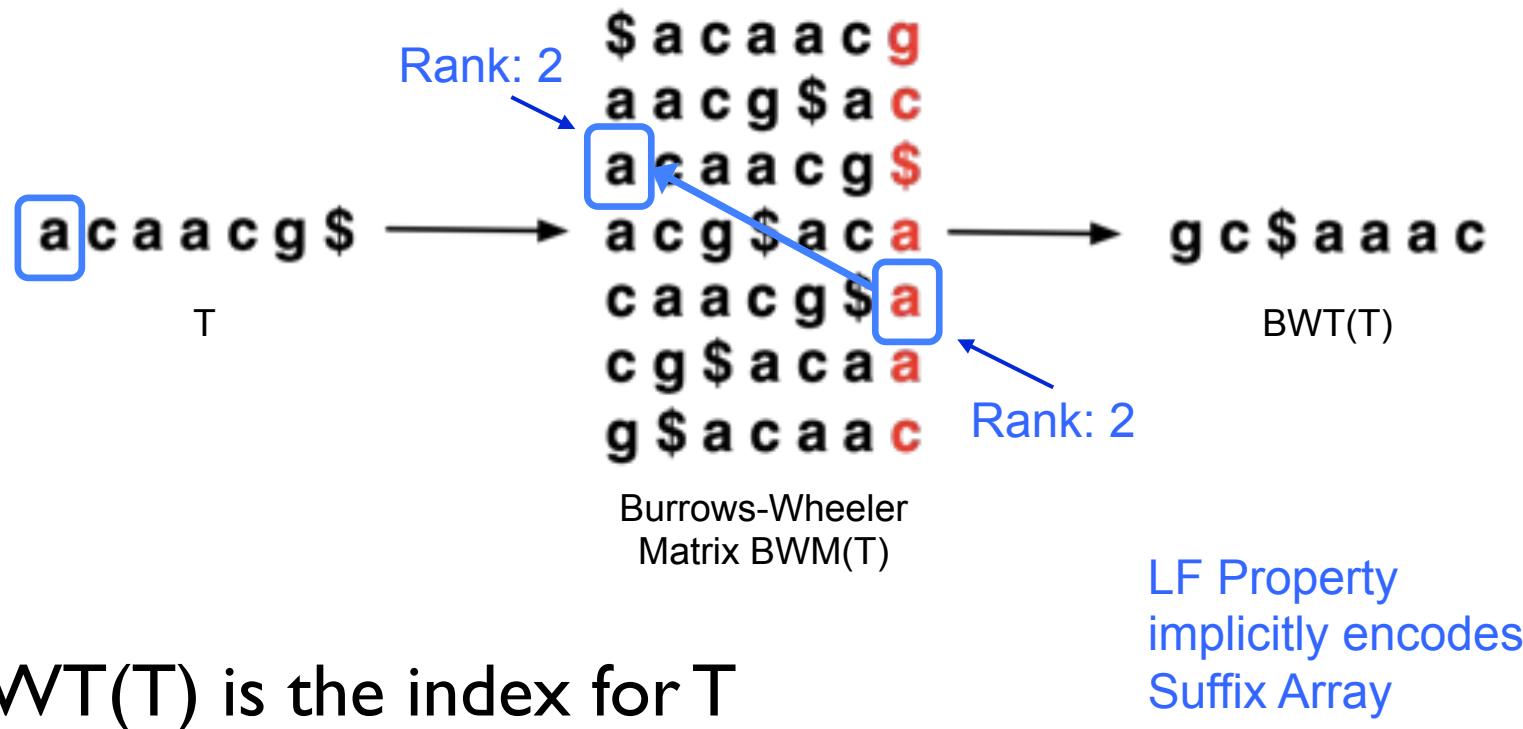


Bowtie: Ultrafast and memory efficient alignment of short DNA sequences to the human genome

Slides Courtesy of Ben Langmead
langmead@umiacs.umd.edu

Burrows-Wheeler Transform

- Reversible permutation of the characters in a text



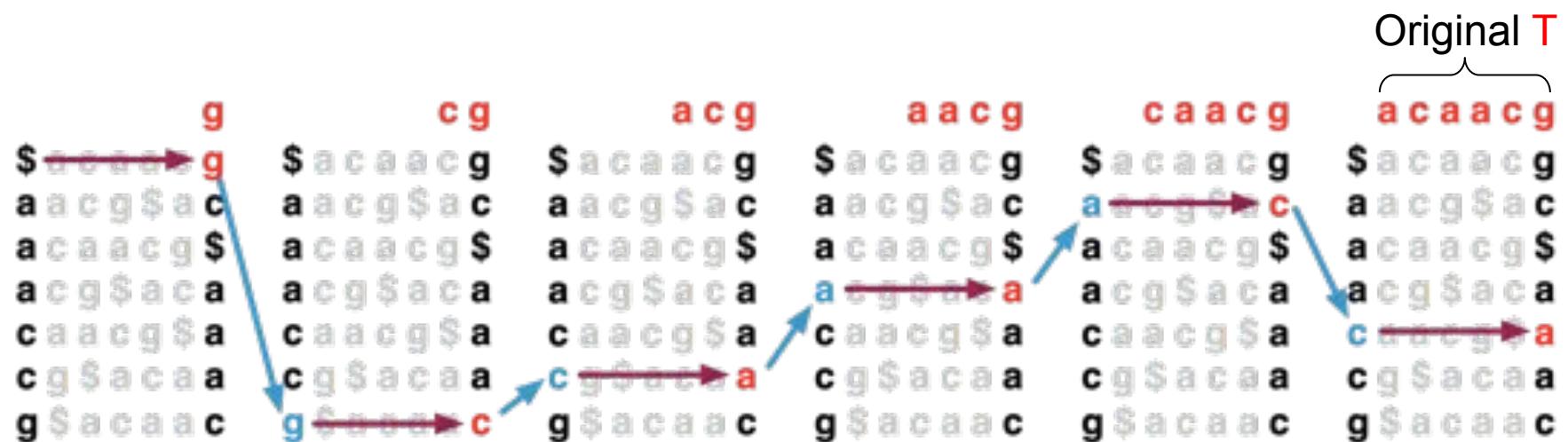
- $\text{BWT}(T)$ is the index for T

A block sorting lossless data compression algorithm.

Burrows M, Wheeler DJ (1994) Digital Equipment Corporation. Technical Report 124

Burrows-Wheeler Transform

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

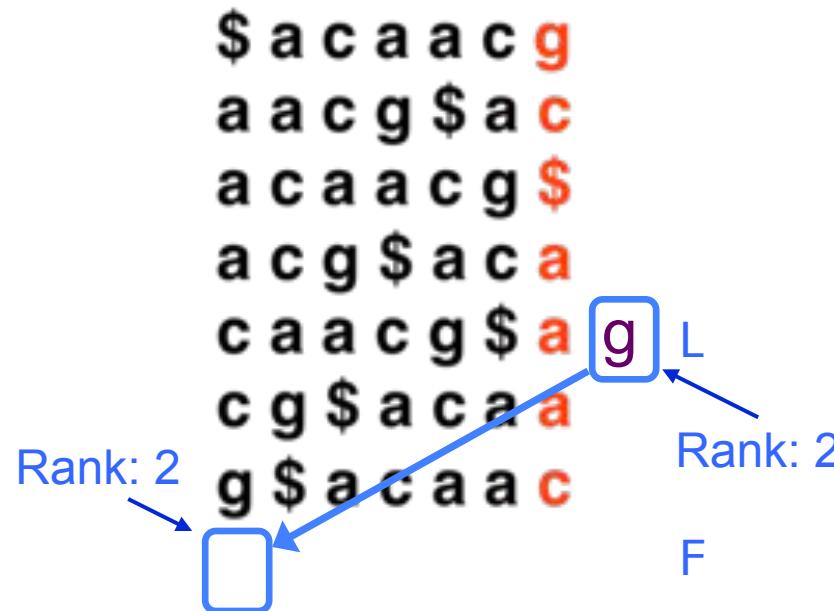


[Decode this BWT string: ACTGA\$TTA]

BWT Exact Matching

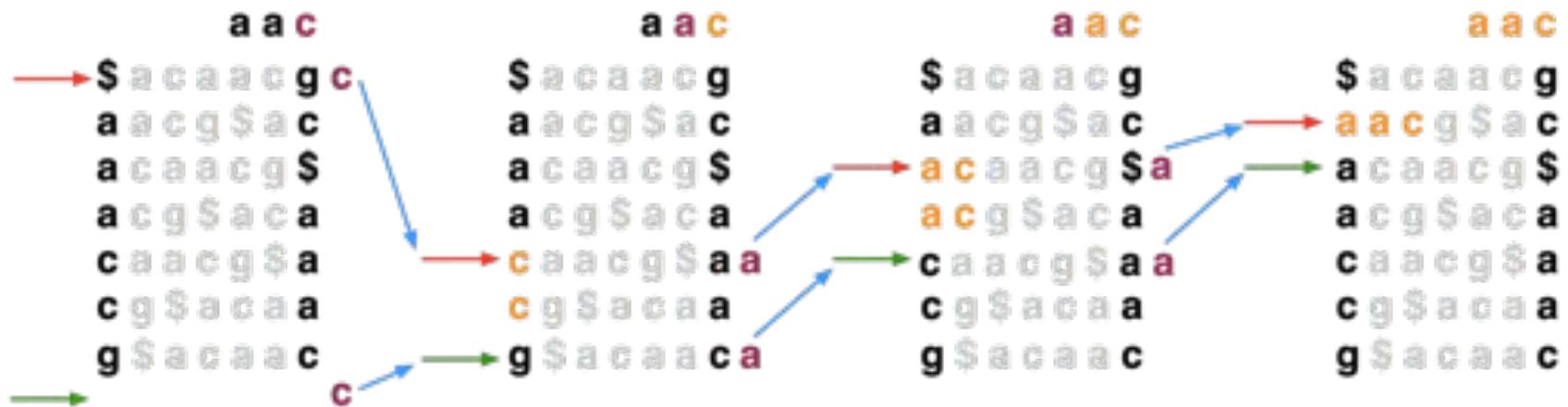
- $\text{LFc}(r, c)$ does the same thing as $\text{LF}(r)$ but it ignores r 's actual final character and “pretends” it's c :

$$\text{LFc}(5, \text{g}) = 8$$



BWT Exact Matching

- Start with a range, (**top**, **bot**) encompassing all rows and repeatedly apply **LFc**:
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.

[Search for TTA this BWT string: ACTGA\$TTA]

Algorithm Overview

1. Split read into segments

Read

Policy: extract 16 nt seed every 10 nt

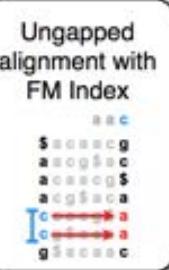
Read (reverse complement)


Seeds

+, 0: CCAGTAGCTCTCAGCC	-, 0: TACAGGCCTGGGTAAA
+, 10: TCAGCCTTATTTACC	-, 10: GGTAAAAATAAGGCTGA
+, 20: TTTACCCAGGCCTGTA	-, 20: GGCTGAGAGCTACTGG

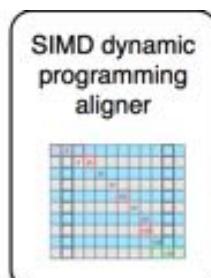
2. Lookup each segment and prioritize

Seeds

+, 0: CCAGTAGCTCTCAGCC	→	Ungapped alignment with FM Index	→	Seed alignments (as B ranges)
+, 10: TCAGCCTTATTTACC				{ [211, 212], [212, 214] }
+, 20: TTTACCCAGGCCTGTA				{ [653, 654], [651, 653] }
-, 0: TACAGGCCTGGGTAAA				{ [684, 685] }
-, 10: GGTAAAAATAAGGCTGA				{ }
-, 20: GGCTGAGAGCTACTGG				{ }
				{ [624, 625] }

3. Evaluate end-to-end match

Extension candidates

SA:684, chr12:1955	→	SIMD dynamic programming aligner	→	SAM alignments
SA:624, chr2:462				r1 0 chr12 1936 0
SA:211: chr4:762				36M * 0 0
SA:213: chr12:1935				CCAGTAGCTCTCAGCCTTATTTACCCAGGCCTGTA
SA:652: chr12:1945				II

Exact Matching Review

- E-value depends on length of genome and inversely on query length
 - $E = (n-m+1)/4^m$

Brute Force
(3 GB)

BANANA
BAN
ANA
NAN
ANA

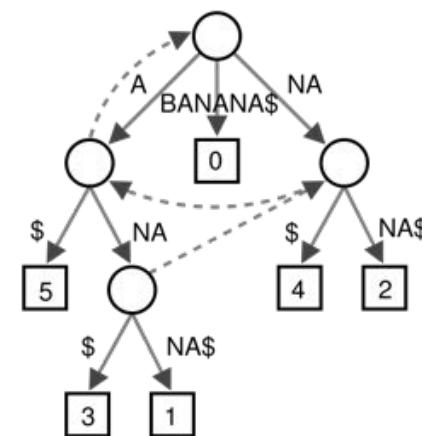
Naive
Slow & Easy

Suffix Array
(>15 GB)

6	\$
5	A\$
3	ANA\$
1	ANANA\$
0	BANANA\$
4	NA\$
2	NANA\$

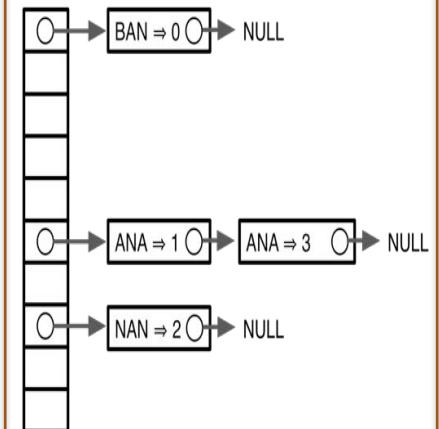
Vmatch, PacBio Aligner
Binary Search

Suffix Tree
(>51 GB)

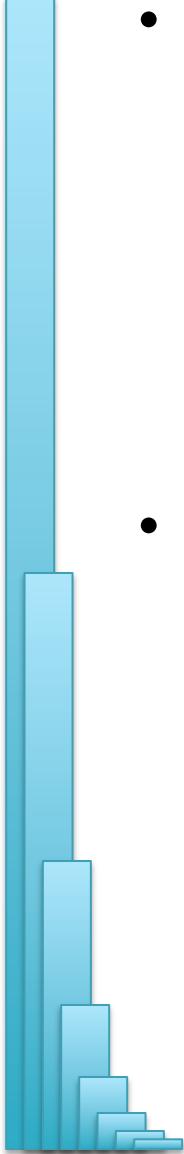


MUMmer, MUMmerGPU
Tree Walking & DFS

Hash Table
(>15 GB)



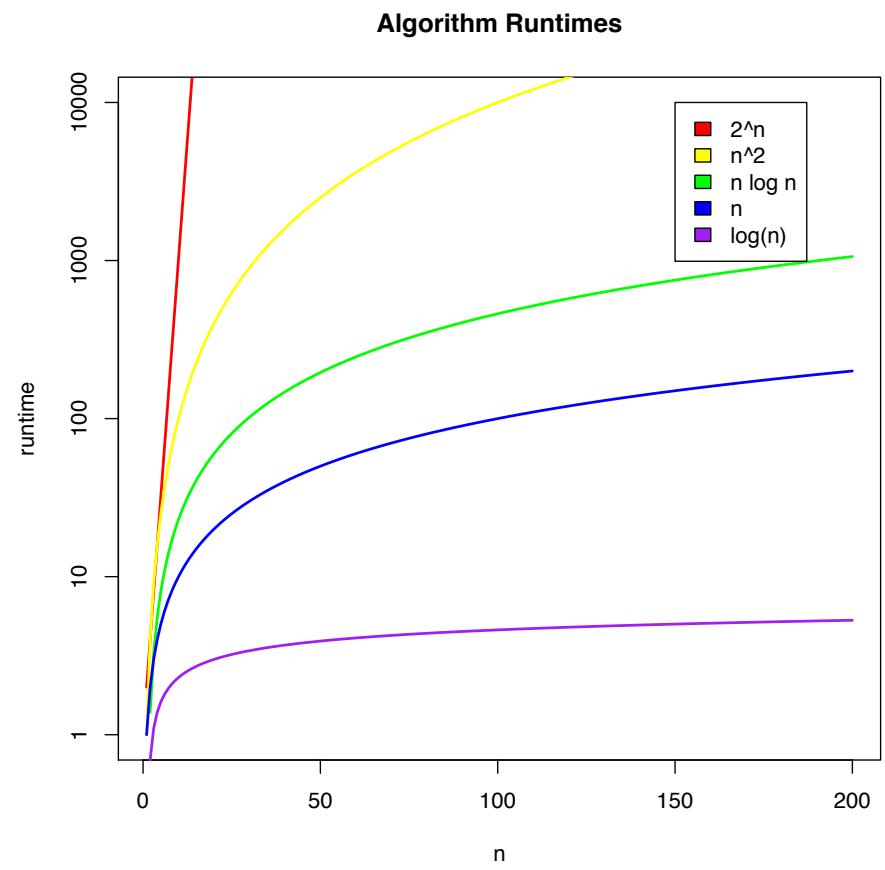
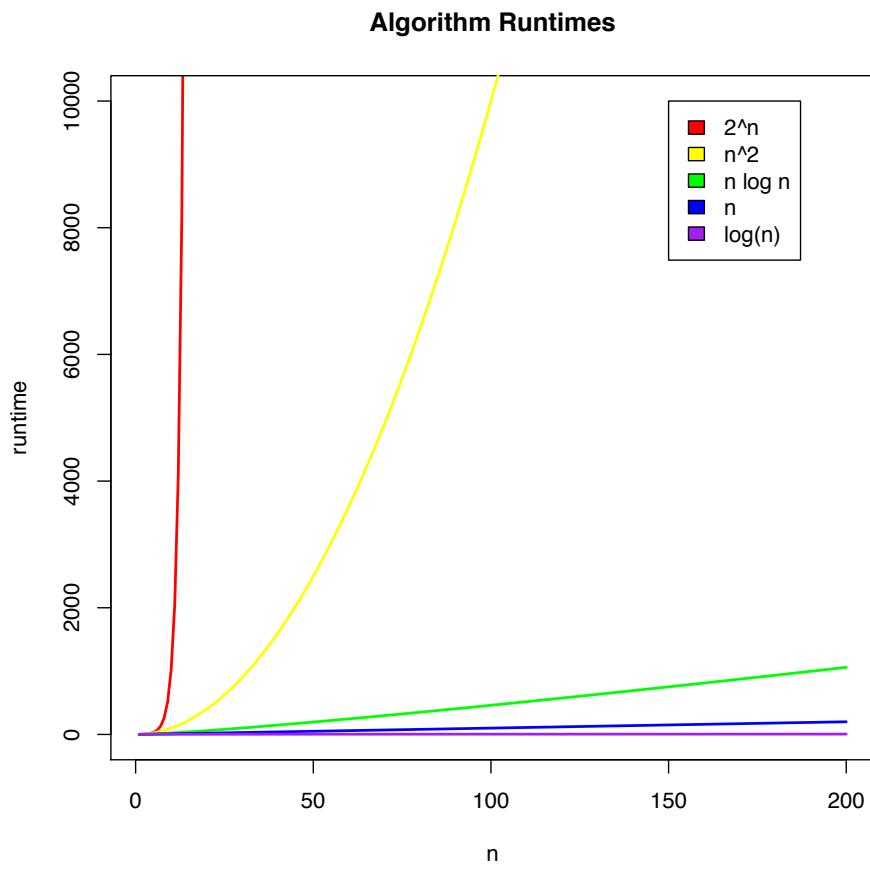
BLAST, MAQ, ZOOM,
RMAP, CloudBurst
Seed-and-extend



Algorithms Summary

- Algorithms choreograph the dance of data inside the machine
 - Algorithms add provable precision to your method
 - A smarter algorithm can solve the same problem with much less work
 - Sequences are really fundamental to biology, learn the techniques to analyze them
- Techniques
 - Binary search: Fast lookup in any sorted list
 - Divide-and-conquer: Split a hard problem into an easier problem
 - Recursion: Solve a problem using a function of itself
 - Hashing: Storing sets across a huge range of values
 - Indexing: Focus on the search on the important parts
 - Different indexing schemes have different space/time features

Algorithmic Complexity



What is the runtime as a function of the input size?

Next Time

- In-exact alignment
 - Smith & Waterman (1981) *Identification of Common Molecular Subsequences*. J. of Molecular Biology. 147:195-197.
- Sequence Homology
 - Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990). *Basic local alignment search tool*. J of Molecular Biology. 215 (3): 403–410.
- Whole Genome Alignment
 - A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O.White, and S.L. Salzberg (1999) *Alignment of Whole Genomes*. Nucleic Acids Research (27):11 2369-2376.

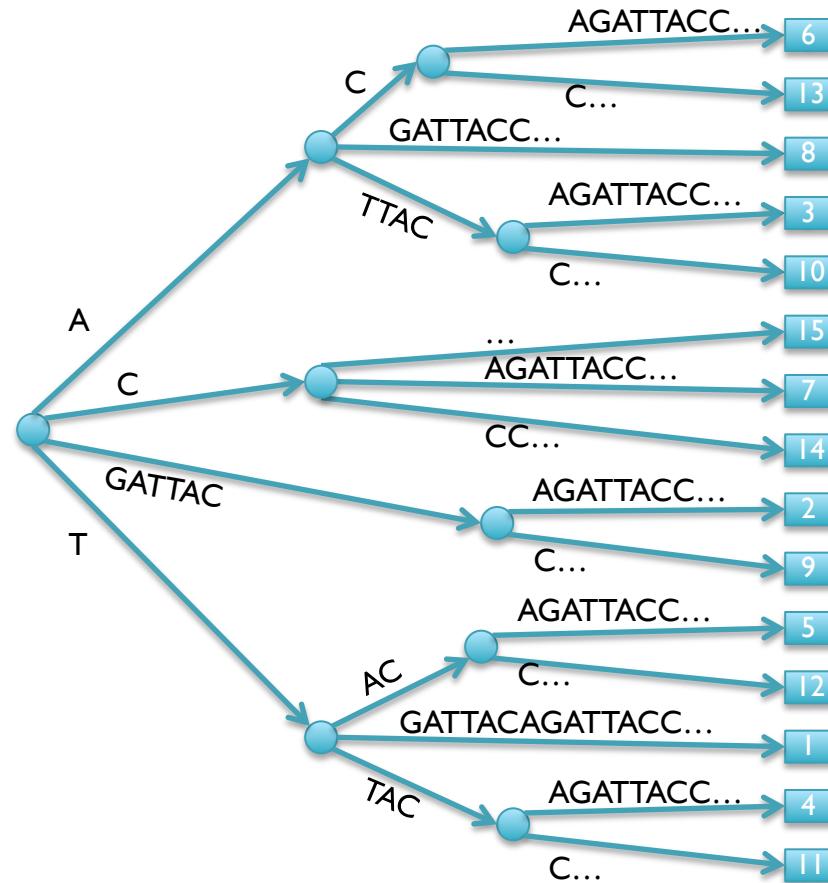
Thank You!

<http://schatzlab.cshl.edu>

@mike_schatz

3. Suffix Trees (Optional)

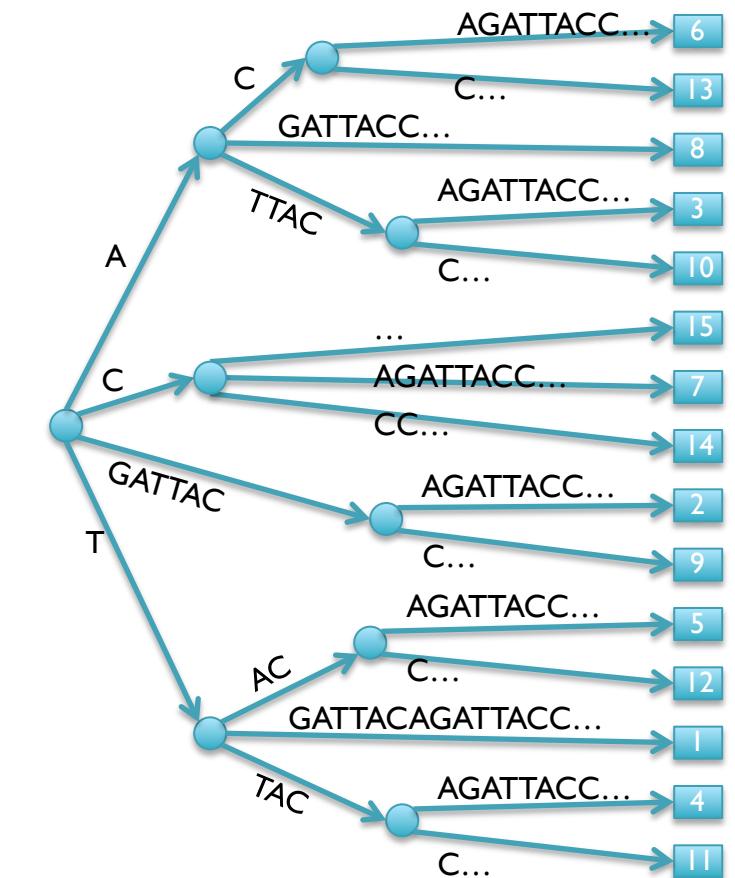
#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11



- Suffix Tree = Tree of suffixes (indexes **all** substrings of a sequence)
 - 1 Leaf (\$) for each suffix, path-label to leaf spells the suffix
 - Nodes have at least 2 and at most 5 children (A,C,G,T,\$)

Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
 - GATTACA

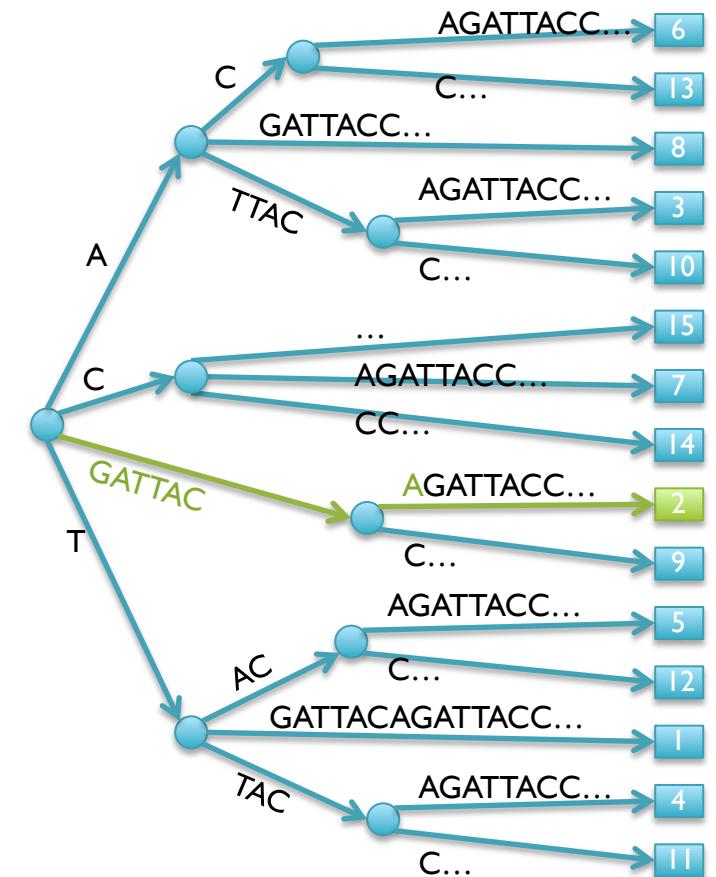


Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
 - GATTACA
 - Matches at position 2

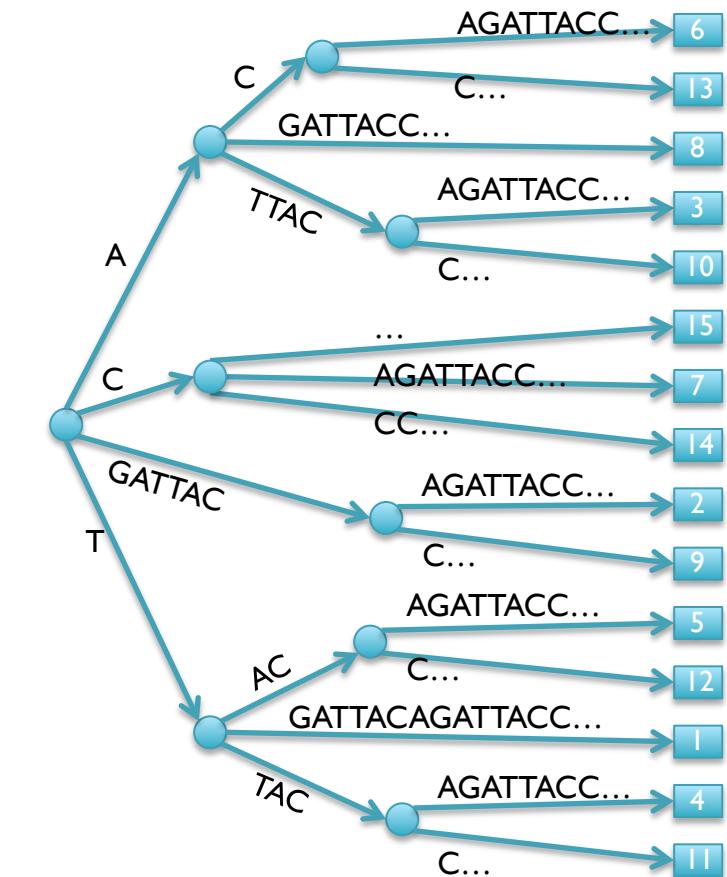
WalkTree

```
cur = ST.Root;  
qrypos = 0;  
while (cur)  
    edge = cur.getEdge(qry[qrypos]);  
    dist = matchstrings(edge, qry, qrypos)  
    if (qrypos+dist == length(qry))  
        print "end-to-end match"  
    else if (dist == length(edge))  
        cur=cur.getNode(edge[0]);  
        qrypos+=dist  
    else  
        print "no match"
```



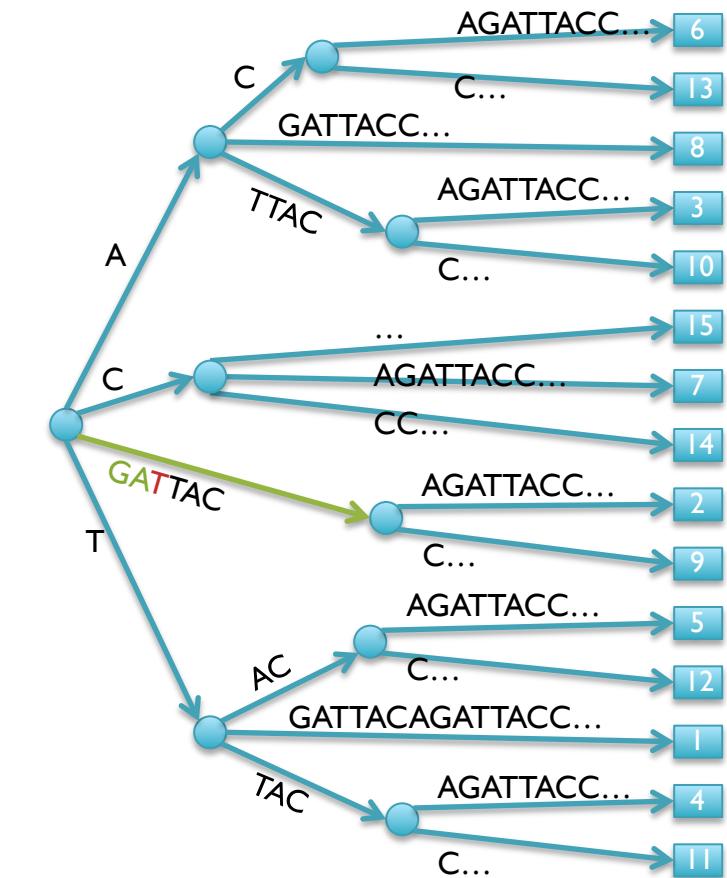
Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
 - GACTACA



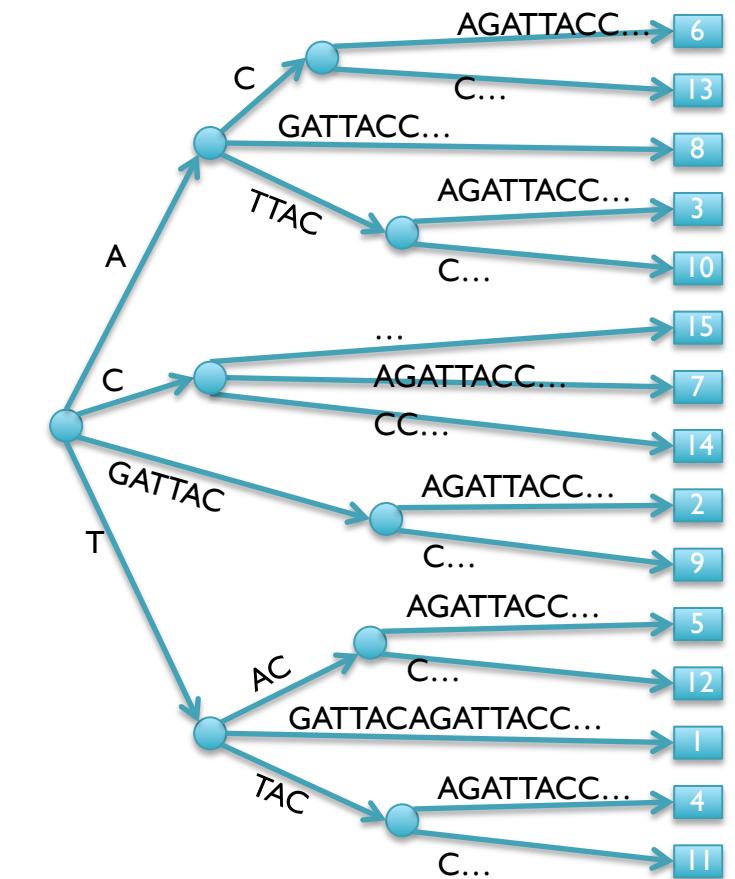
Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
 - GACTACA
 - Fell off tree – no match



Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
 - ATTAC



Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree

- ATTAC
 - Matches at 3 and 10

- Query Lookup in 2 phases:
 - I. Walk along edges to find matches
 2. Walk subtree to find positions

```
DepthFirstPrint(Node cur)
```

```
if cur.isLeaf
```

```
    print cur.pos
```

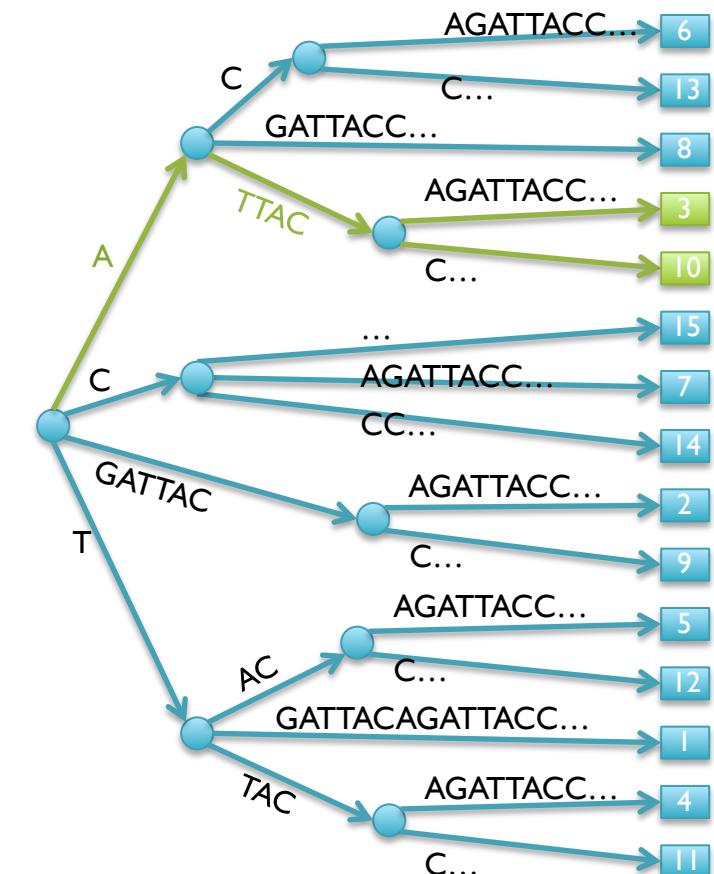
```
else
```

```
    foreach child in cur.children
```

```
        DepthFirstPrint(child)
```

[What is the running time of DFP

=> How many nodes does the tree have?]



Suffix Tree Properties & Applications

Properties

- Number of Nodes/Edges: $O(n)$
- Tree Size: $O(n)$
- Max Depth: $O(n)$
- Construction Time:
 - Tricky to implement, prove efficiency
 - Brute force algorithm requires $O(n^2)$

Applications

- Sorting all suffixes: $O(n)$
- Check for query: $O(m)$
- Find all z occurrences of a query $O(m + z)$
- Find maximal exact matches $O(m)$
- Longest common substring $O(m)$
- ...
- Used for many string algorithms in linear time
 - Many can be implemented on suffix arrays using a little extra work

[HOW?]

