Sorting, Searching, & Aligning

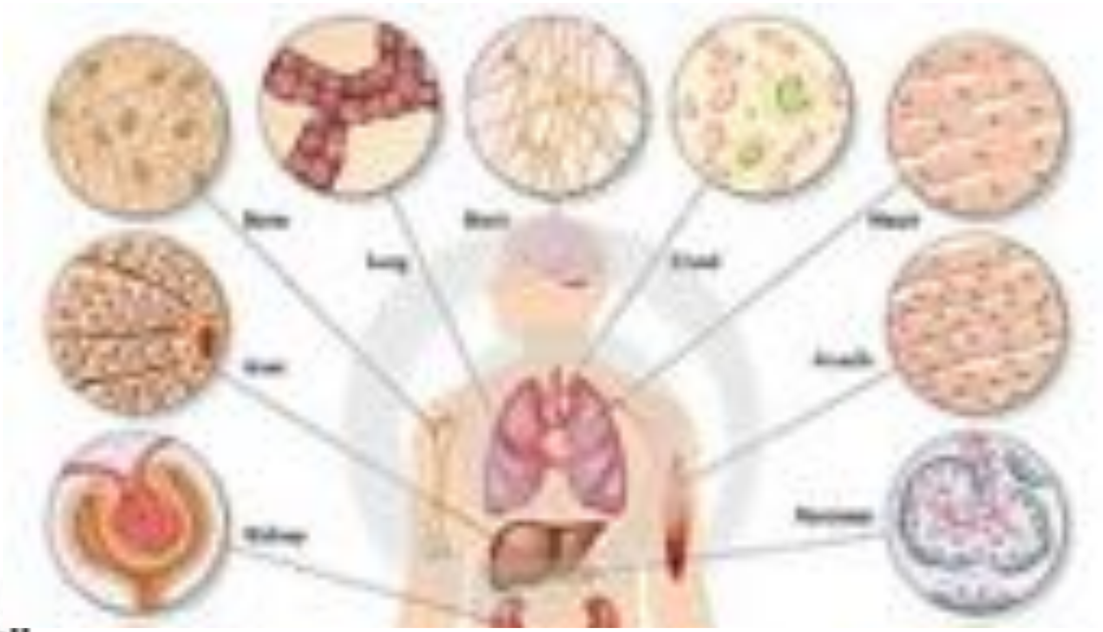
Michael Schatz

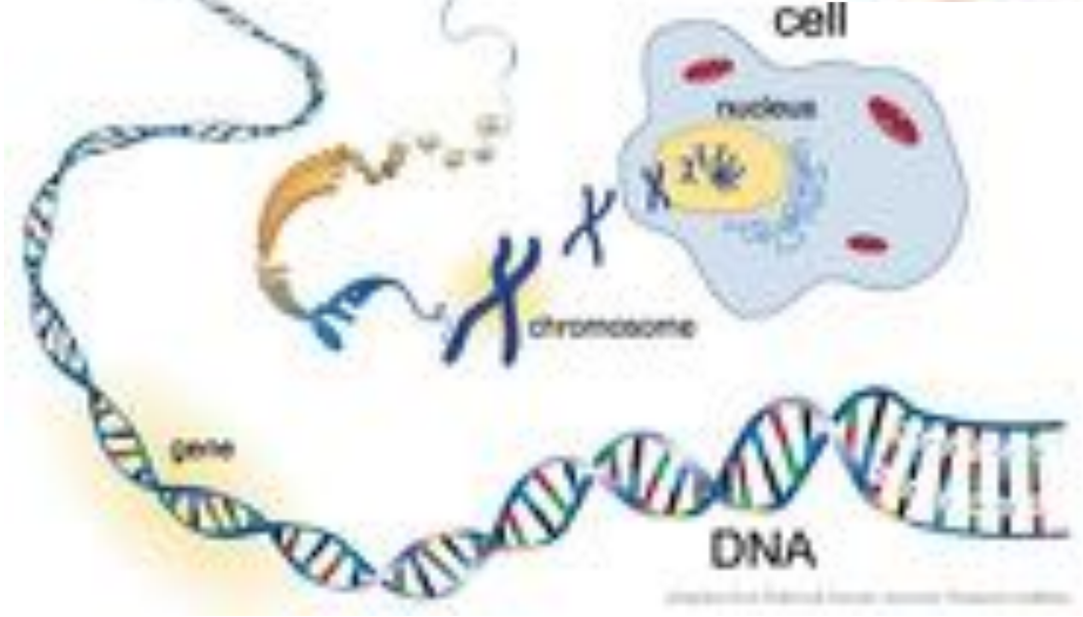
QB/Bioinformatics Lecture 1

Quantitative Biology 2014

Cells & DNA

Each cell of your body contains an exact copy of your 3 billion base pair genome.

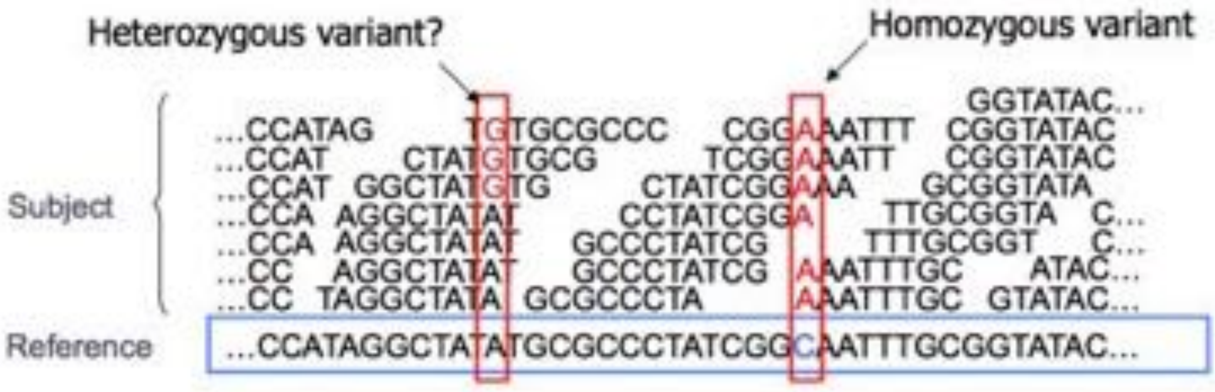




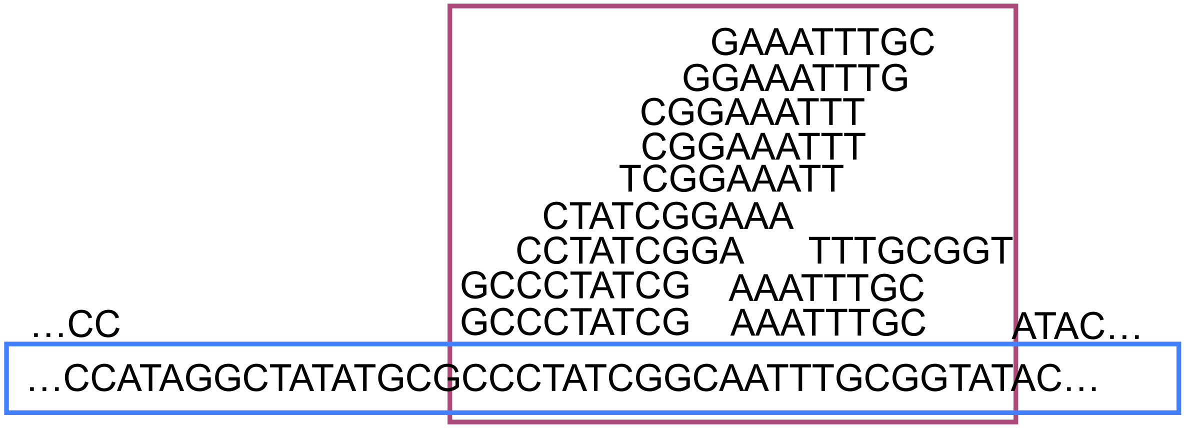
Your specific nucleotide sequence encodes the genetic program for your cells and ultimately your traits

Short Read Applications

- Genotyping: Identify Variations


Heterozygous variant?
Homozygous variant
Subject
GGTATAC...
...CCATAG TGTGCGCCC CGGAAATTT CGGTATAC
...CCAT CTATGTGCG TCGGAAATT CGGTATAC
...CCAT GGCTATGTG CTATCGGAAA GCGGTATA
...CCA AGGCTATAT CCTATCGGA TTGCGGTA C...
...CCA AGGCTATAT GCCCTATCG TTTGCGGT C...
...CC AGGCTATAT GCCCTATCG AAATTTGC ATAC...
...CC TAGGCTATA GCGCCCTA AAATTTGC GTATAC...
Reference
...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...

- *-seq: Classify & measure significant peaks



GAAATTTGC

GGAAATTTG

CGGAAATTT

CGGAAATTT

TCGGAAATT

CTATCGGAAA

CCTATCGGA TTTGCGGT

GCCCTATCG AAATTTGC

...CC GCCCTATCG AAATTTGC ATAC...

...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...

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

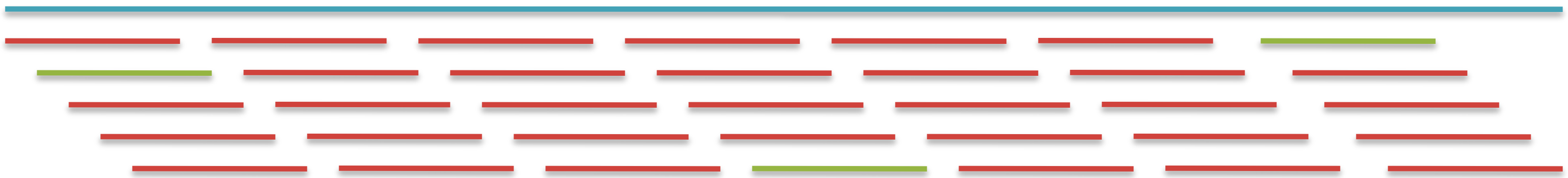
- Where is GATTACA in the human genome?

- 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?]

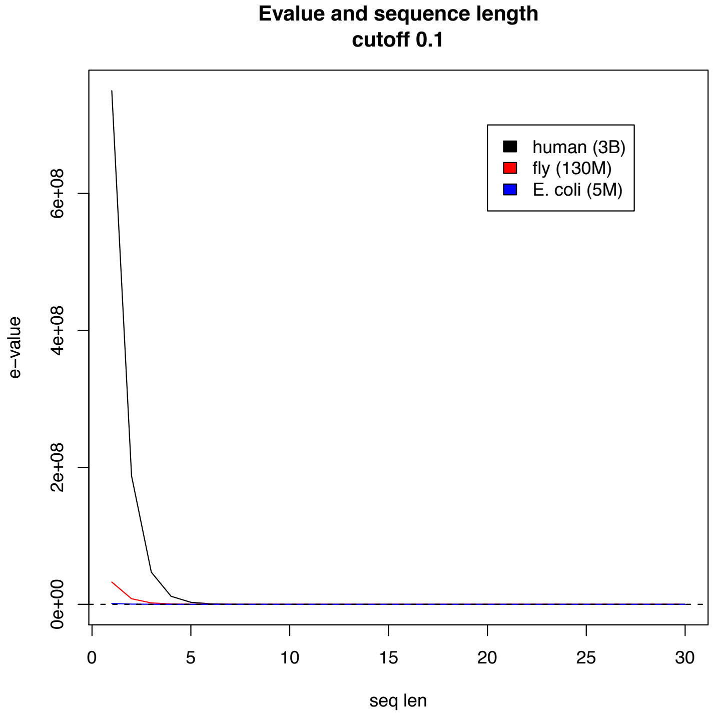
[How long would it take if we double both?]

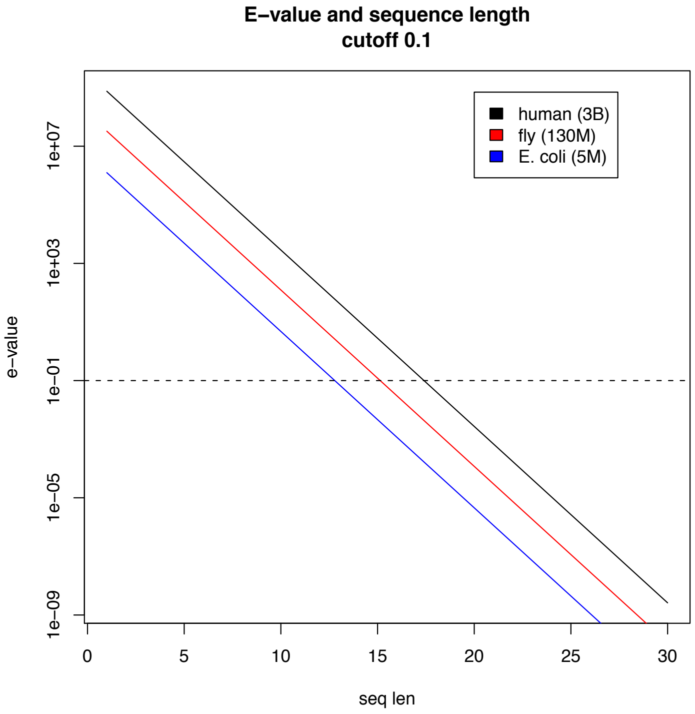
Expected Occurrences

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/(4^m) [183,105 expected occurrences]

[How long do the reads need to be for a significant match?]





Brute Force Reflections

Why check every position?

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

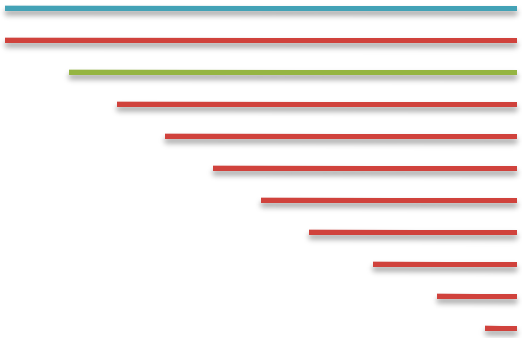
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
								G	A	T	T	A	C	A	

- 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

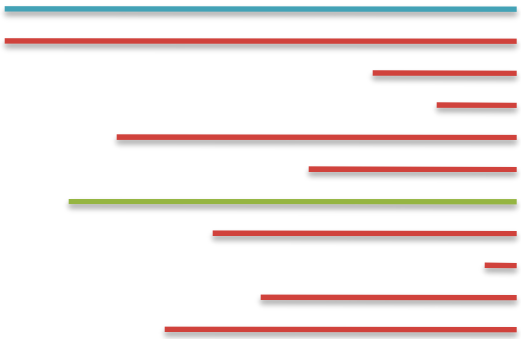
Suffix Arrays: Searching the Phone Book

- 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?]

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;

#	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
Lo →	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!

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

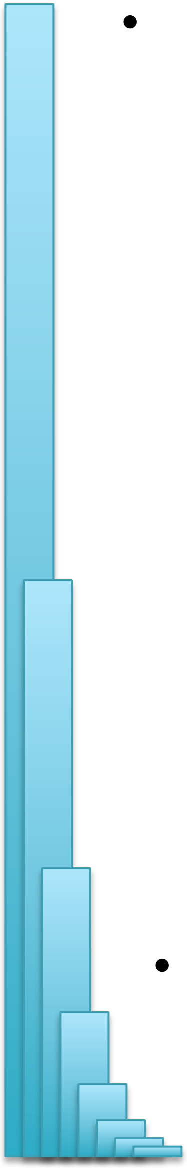
```
Initialize search range to entire list
    mid = (hi+lo)/2; middle = suffix[mid]
    if query matches middle: done
    else if query < middle: pick low range
    else if query > middle: pick hi range
Repeat until done or empty range                    [WHEN?]
```

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

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

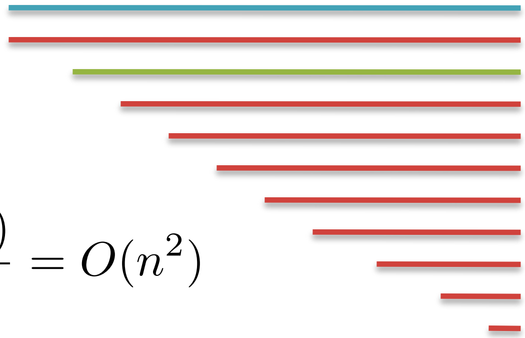
[How long does it take to search 6B or 24B nucleotides?]

Suffix Array Construction



- How can we store the suffix array?

  [How many characters are in all suffixes combined?]



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

- 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

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

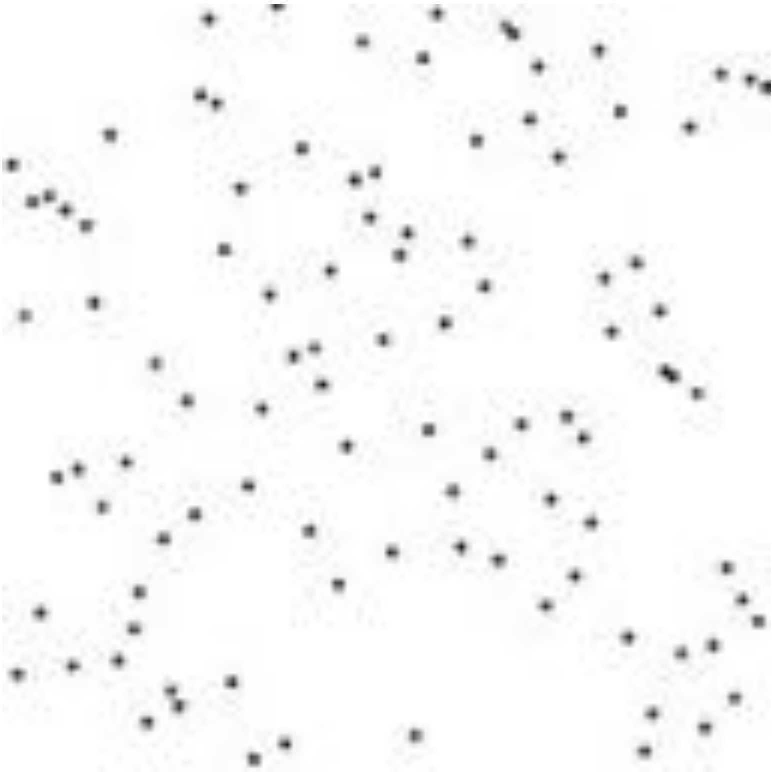
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78

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



http://en.wikipedia.org/wiki/Selection_sort

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

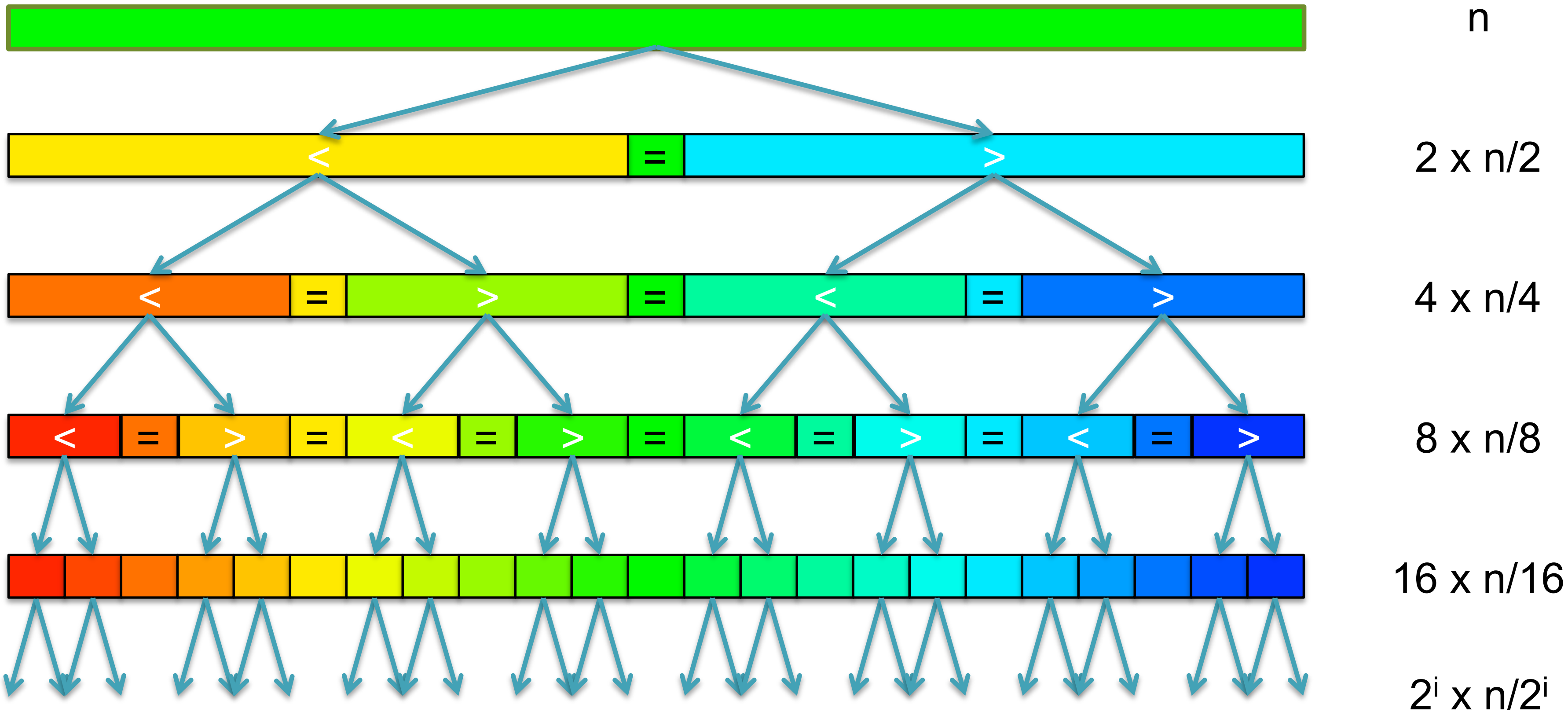
$$T = n + (n-1) + (n-2) + \cdots + 3 + 2 + 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

  - 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?]

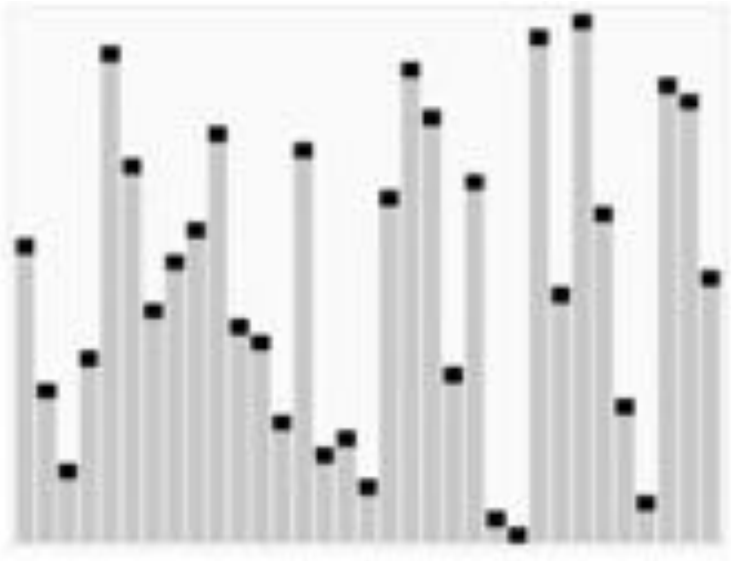
QuickSort Analysis

- 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))

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

$$T(n) = n + 2(\frac{n}{2}) + 4(\frac{n}{4}) + \cdots + n(\frac{n}{n}) = \sum_{i=0}^{lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{lg(n)} n = O(n \lg n)$$

[~94B]

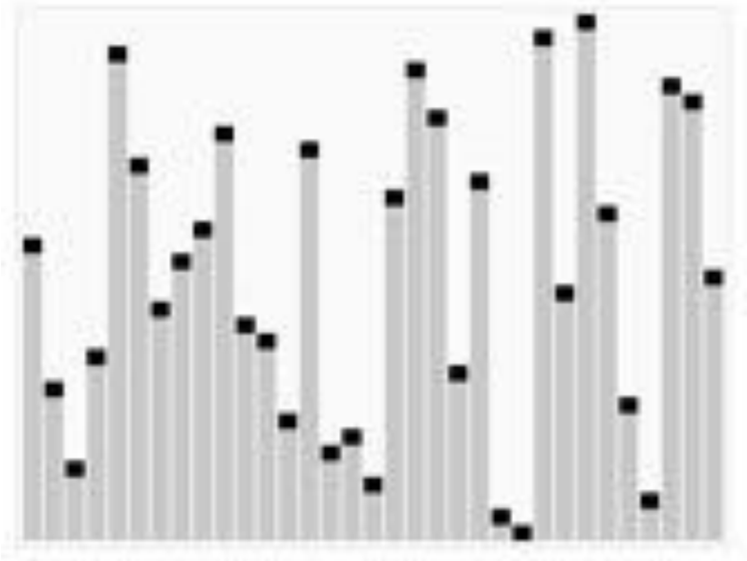
QuickSort Analysis

- QuickSort(Input: list of n numbers)

```
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))

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

$$T(n) = n + 2(\frac{n}{2}) + 4(\frac{n}{4}) + \cdots + n(\frac{n}{n}) = \sum_{i=0}^{lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{lg(n)} n = O(n \lg n)$$

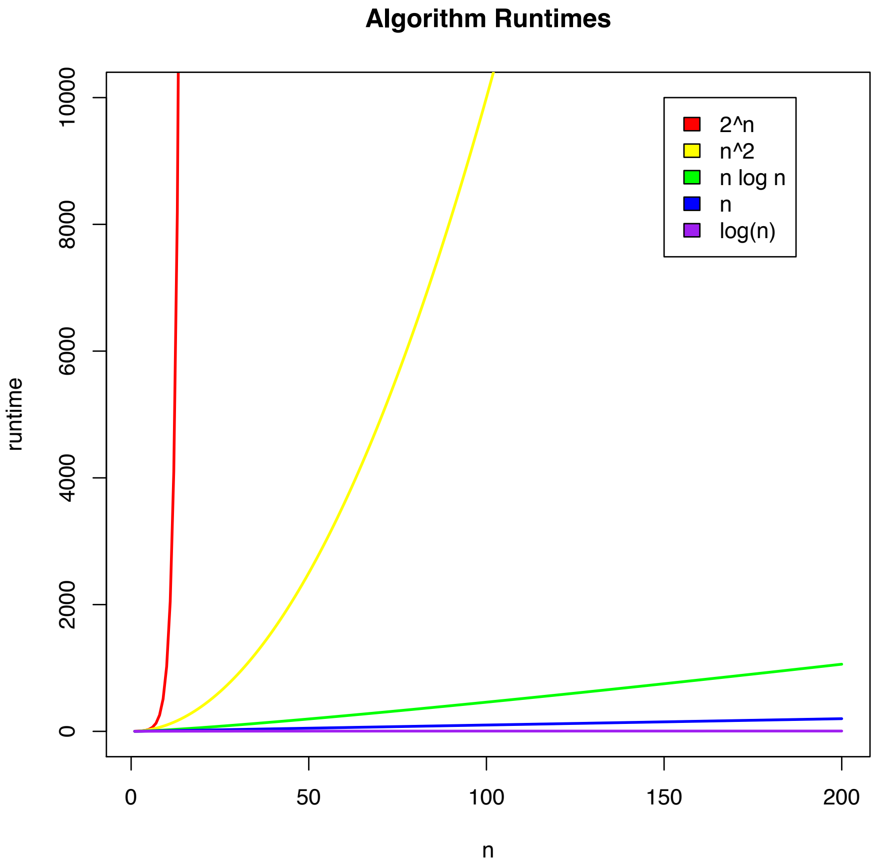
[~94B]

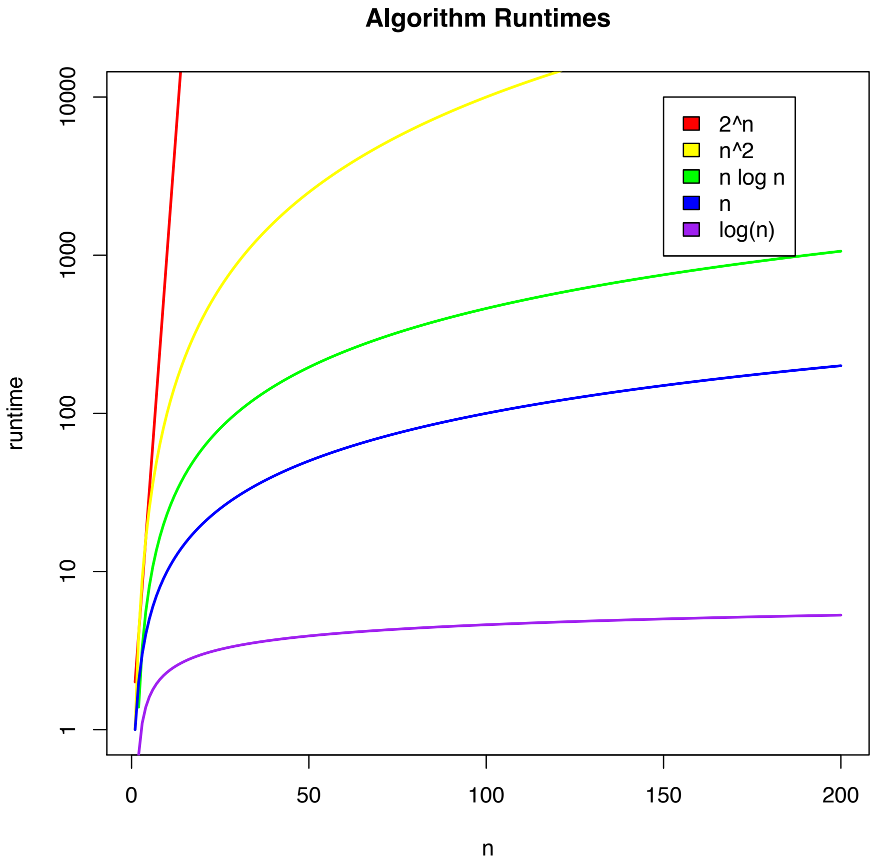
QuickSort in Python

```
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

Algorithmic Complexity





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

THE G-NOME PROJECT

Break

Algorithmic challenge

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

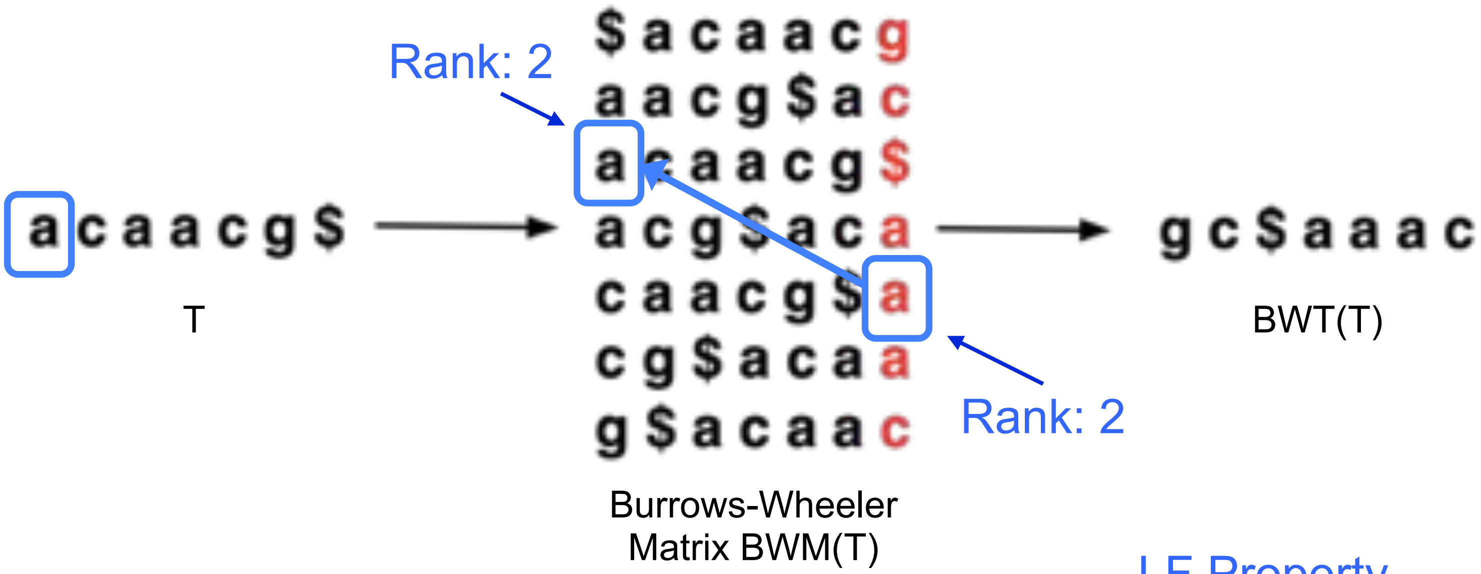
What would such an index look like?

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



acaacg$

$acaacg
aacg$ac
acaacg$
acg$aca
caacg$a
cg$acaa
g$acaac

gc$aaac

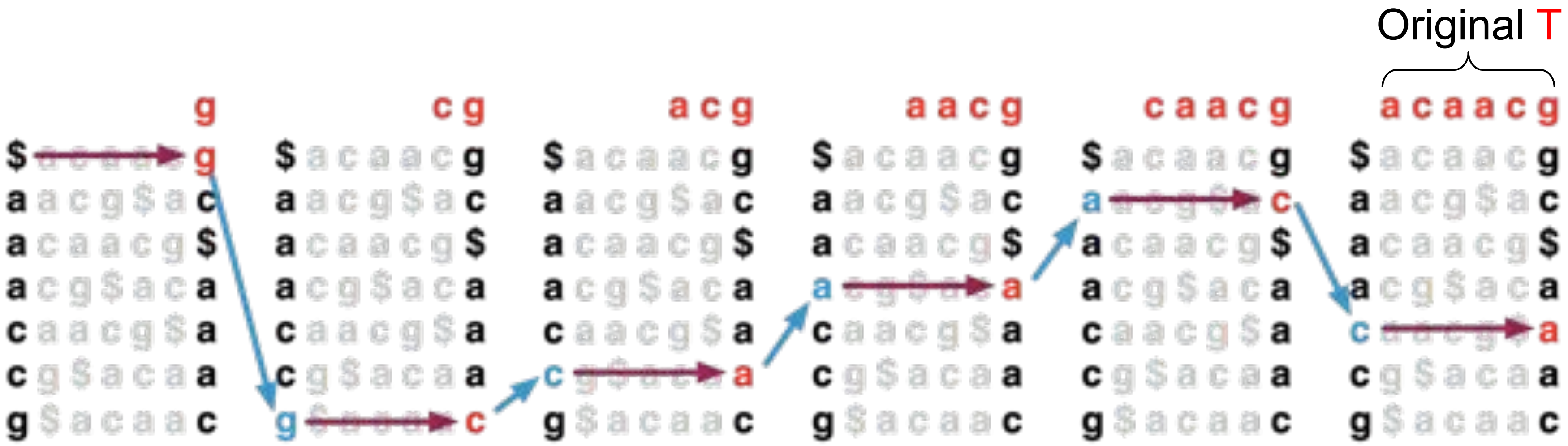
LF Property implicitly encodes Suffix Array

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

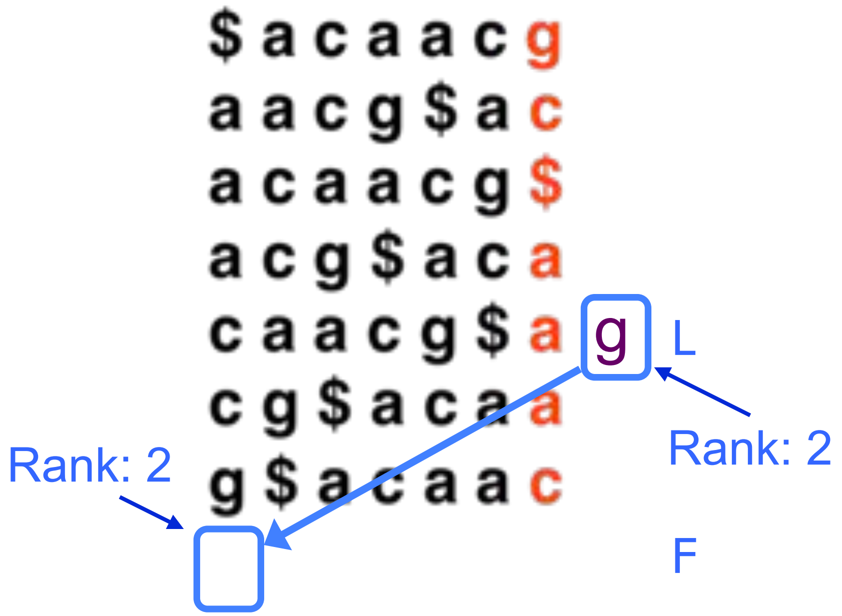


[Decode this BWT string: ACTGA$TTA]

BWT Exact Matching

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

LFc(5, g) = 8



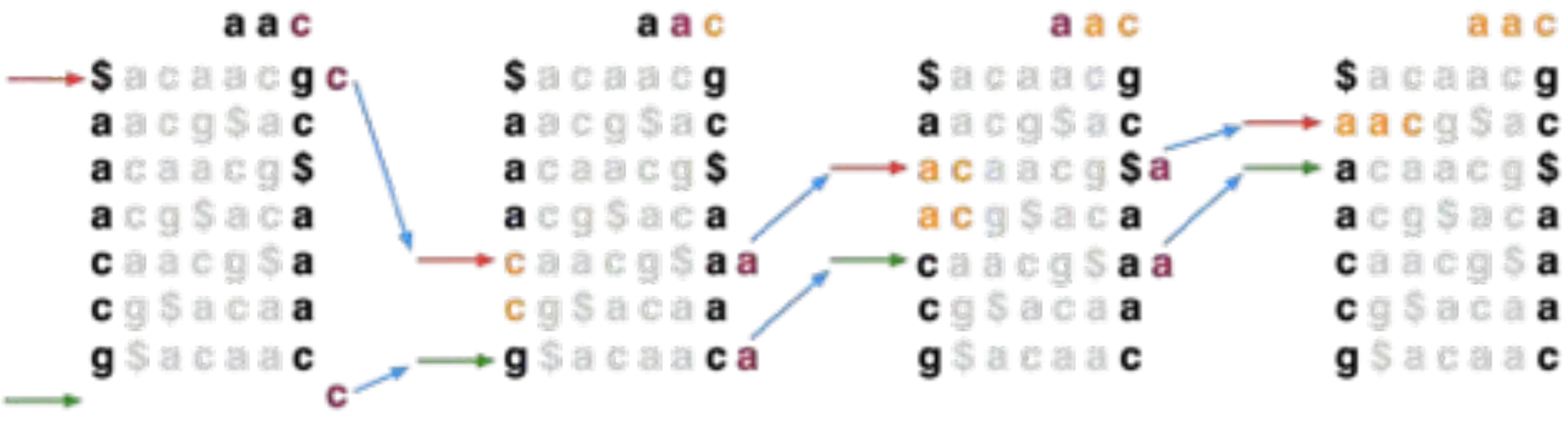
```
$ a c a a c g
a a c g $ a c
a c a a c g $
a c g $ a c a
c a a c g $ a
c g $ a c a a
g $ a c a a c
```

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

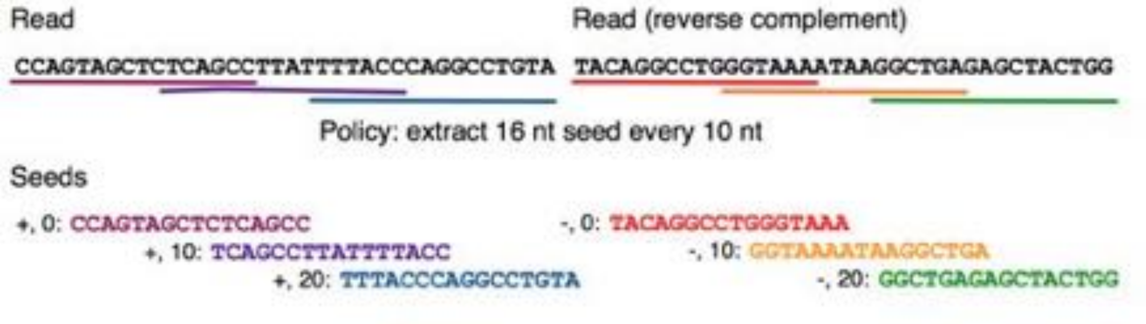


Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*

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

Algorithm Overview

1. Split read into segments



Read

CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA

Read (reverse complement)

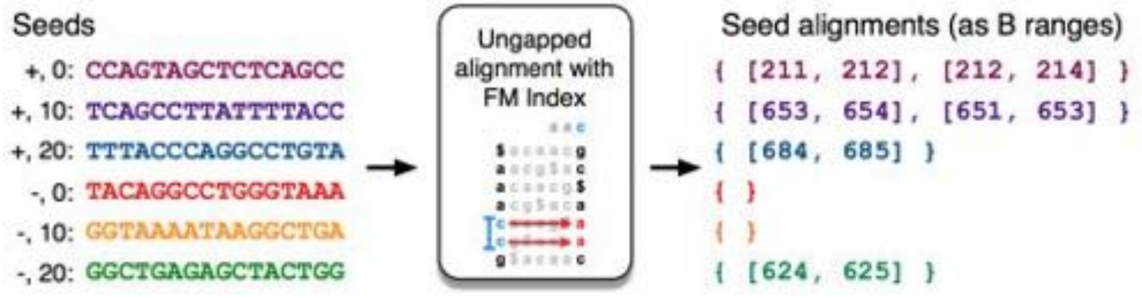
TACAGGCCTGGGTAAAATAAGGCTGAGAGCTACTGG

Policy: extract 16 nt seed every 10 nt

Seeds

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

2. Lookup each segment and prioritize



Seeds

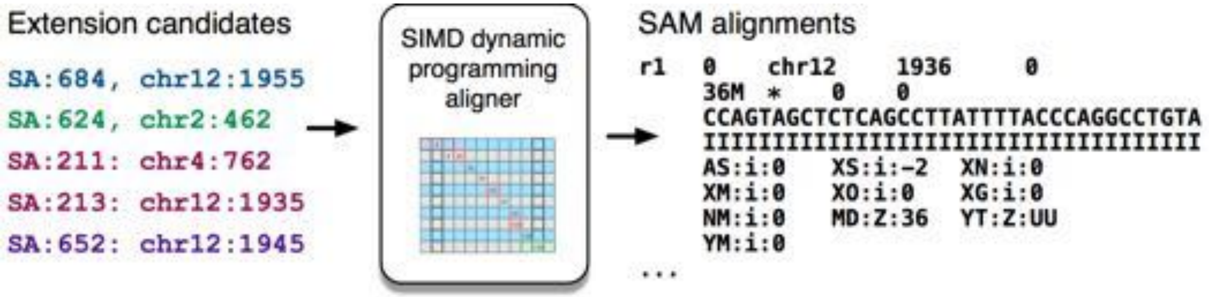
```
 +, 0: CCAGTAGCTCTCAGCC
+, 10: TCAGCCTTATTTTACC
+, 20: TTTACCCAGGCCTGTA
 -, 0: TACAGGCCTGGGTAAA
-, 10: GGTAAAATAAGGCTGA
-, 20: GGCTGAGAGCTACTGG
```

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
...
```

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

Next Time

- Friday:
  - HW Review
  - Group Discussion of ENCODE

- Monday:
  - Dynamic Programming & Alignment applications

- Tuesday:
  - Graphs & Assembly

- Thursday:
  - Diversity of modern and ancient humans

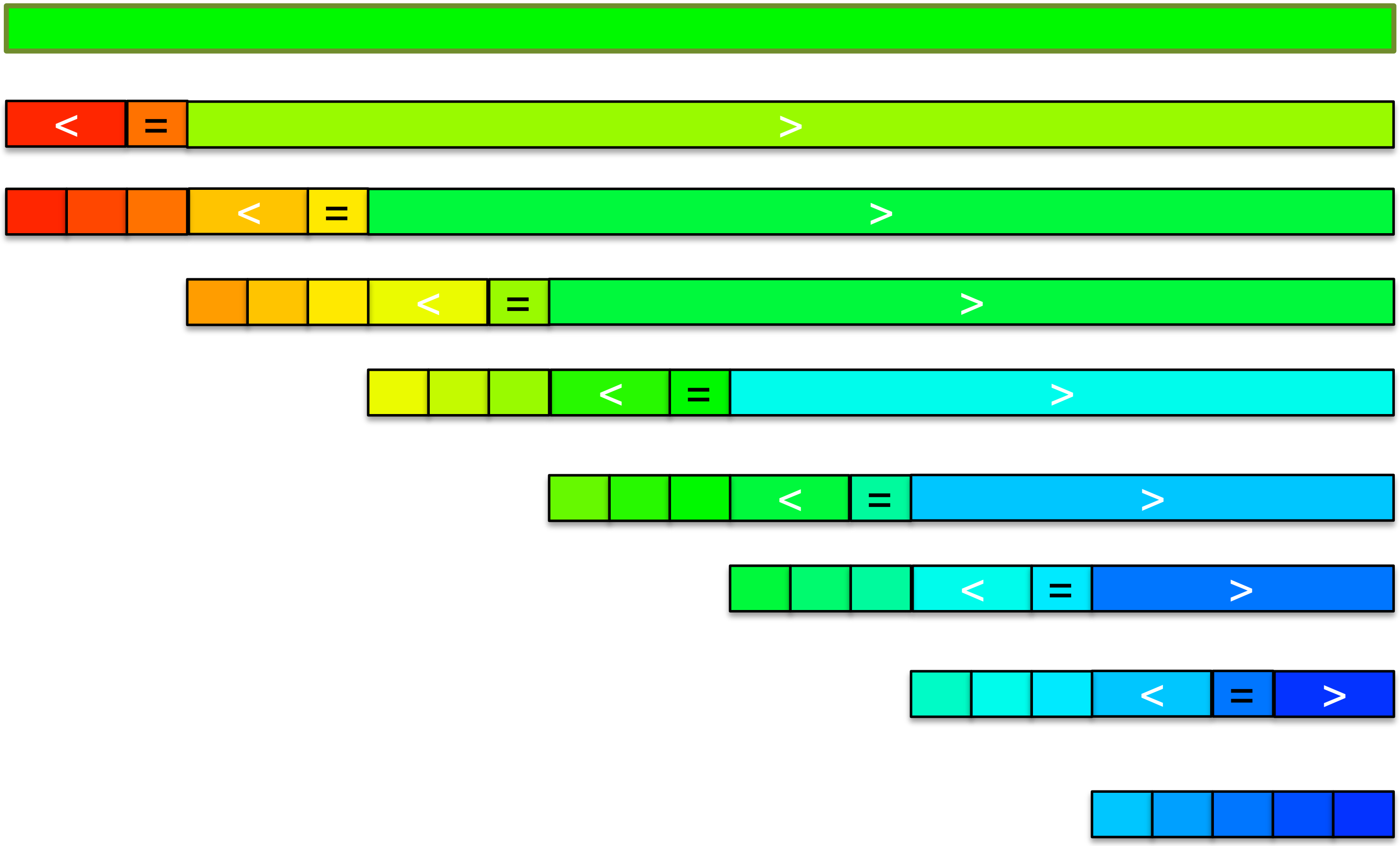
- Friday:
  - Gene Finding + ChromHMM + Review

Thank You!

http://schatzlab.cshl.edu
@mike_schatz

Picking the Median

- What if we miss the median and do a 90/10 split instead?



n

$n/10 + 9n/10$

$\ldots + 81n/100$

$\ldots + 729n/1000$

$\ldots + 6561n/10000$

$\ldots + 59049n/100000$

$\ldots + 531441n/1000000$

$\ldots + 4782969n/10000000$

$\ldots + 9^i n/10^i$

[How many times can we cut 10% off a list?]

Randomized Quicksort

- 90/10 split runtime analysis

$$T(n) = n + T(\frac{n}{10}) + T(\frac{9n}{10})$$

$$T(n) = n + \frac{n}{10} + T(\frac{n}{100}) + T(\frac{9n}{100}) + \frac{9n}{10} + T(\frac{9n}{100}) + T(\frac{81n}{100})$$

$$T(n) = n + n + T(\frac{n}{100}) + 2T(\frac{9n}{100}) + T(\frac{81n}{100})$$

$$T(n) = \sum_{i=0}^{\log_{10/9}(n)} n = O(n \lg n)$$

Find smallest x s.t.

$$(9/10)^x n \leq 1$$

$$(10/9)^x \geq n$$

$$x \geq \log_{10/9} n$$

- If we randomly pick a pivot, we will get at least a 90/10 split with very high probability
  - – Everything is okay as long as we always slice off a fraction of the list

[Challenge Question: What happens if we slice 1 element]