

# Read mapping

Michael Schatz

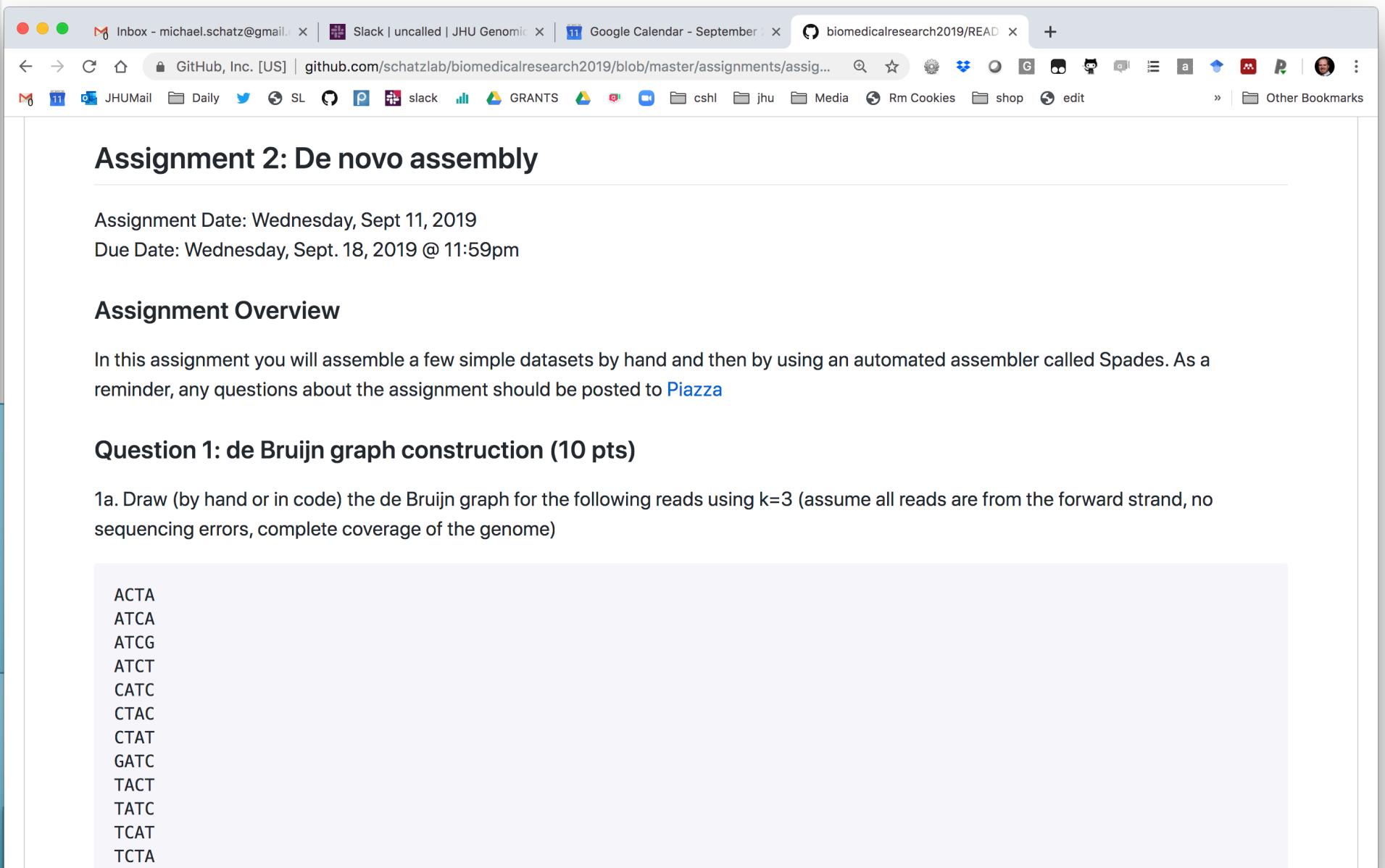
Sept 18, 2019

Lecture 6: Computational Biomedical Research



# Assignment 2: De novo Assembly

## Due Wed Sept 18 @ 11:59pm



A screenshot of a web browser window displaying the assignment details. The browser has several tabs open at the top, including 'Inbox - michael.schatz@gmail.com', 'Slack | uncalled | JHU Genomic', 'Google Calendar - September', and 'biomedicalresearch2019/READ'. The main content area shows the assignment title 'Assignment 2: De novo assembly' and its due date 'Wednesday, Sept 11, 2019'. Below this, there is an 'Assignment Overview' section and a 'Question 1: de Bruijn graph construction (10 pts)' section. The question 1 section contains a list of DNA sequence reads: ACTA, ATCA, ATCG, ATCT, CATC, CTAC, CTAT, GATC, TACT, TATC, TCAT, and TCTA.

## Assignment 2: De novo assembly

Assignment Date: Wednesday, Sept 11, 2019  
Due Date: Wednesday, Sept. 18, 2019 @ 11:59pm

### Assignment Overview

In this assignment you will assemble a few simple datasets by hand and then by using an automated assembler called Spades. As a reminder, any questions about the assignment should be posted to [Piazza](#)

### Question 1: de Bruijn graph construction (10 pts)

1a. Draw (by hand or in code) the de Bruijn graph for the following reads using  $k=3$  (assume all reads are from the forward strand, no sequencing errors, complete coverage of the genome)

```
ACTA
ATCA
ATCG
ATCT
CATC
CTAC
CTAT
GATC
TACT
TATC
TCAT
TCTA
```

# Assignment 3: Sequence Alignment

## Due Monday Sept 30 @ 11:59pm

Assignment Date: Wednesday, Sept 18, 2019  
Due Date: Wednesday, Sept. 25, 2019 @ 11:59pm

### Assignment Overview

In this assignment you will consider the requirements for sequence alignment. As a reminder, any questions about the assignment should be posted to [Piazza](#)

#### Question 1: Minimum Alignment Lengths (10 pts)

Determine how many bases long a given pattern P should be to ensure that occurrences of P are unlikely to be chance events ( $e < 0.00001$ ) in genomes of the following sizes:

- 1a. 5.2Mb (Bacillus anthracis – the microbe that causes anthrax)
- 1b. 100Mb (Caenorhabditis elegans - model worm)
- 1c. 3.1Gb (Homo sapiens - human)
- 1d. 18GB (Triticum aestivum – bread wheat)
- 1e. 670Gb (Polychaos dubium – amoeba, has largest known genome)

#### Question 2: Edit distance (10 pts)

Compute the edit distance of (a portion of) the human hemoglobin alpha and beta subunits, showing the dynamic programming matrix and the aligned sequences. Assume a fixed unit cost to substitute one amino acid for another.

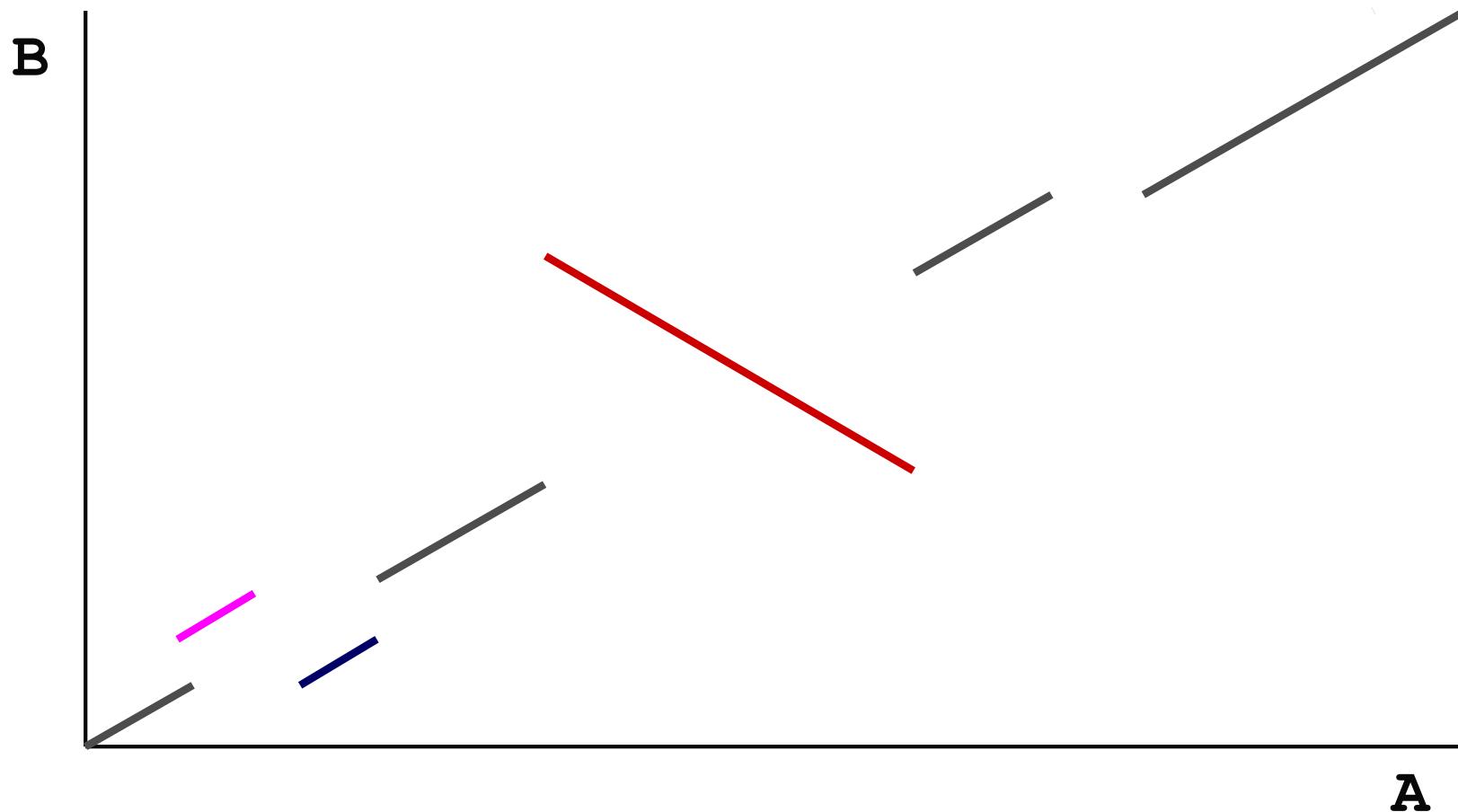
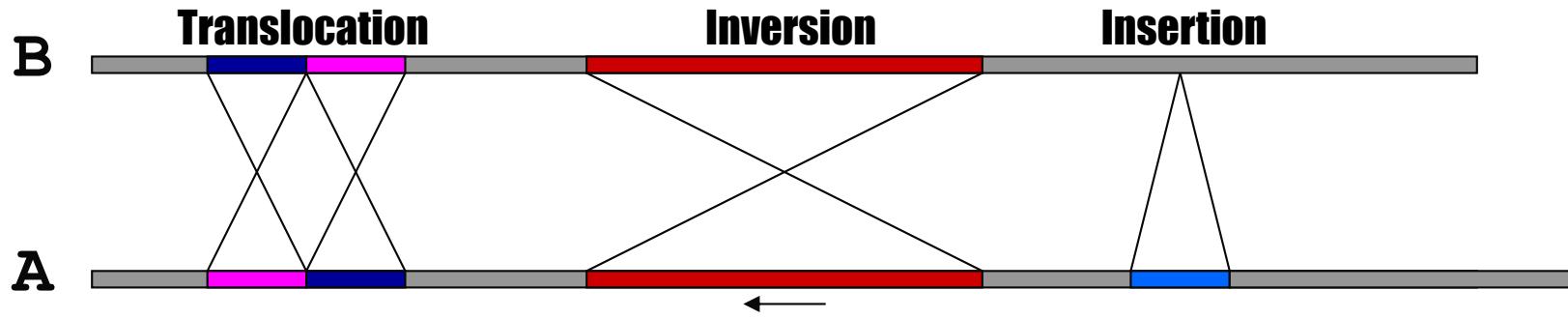
```
Alpha: EALERMFLSFPTTKTYFPHFDSLHGSAQVK
Beta:  EALGRLLVYPWTQRFFESFGDLSTPDAMGNPKVK
```

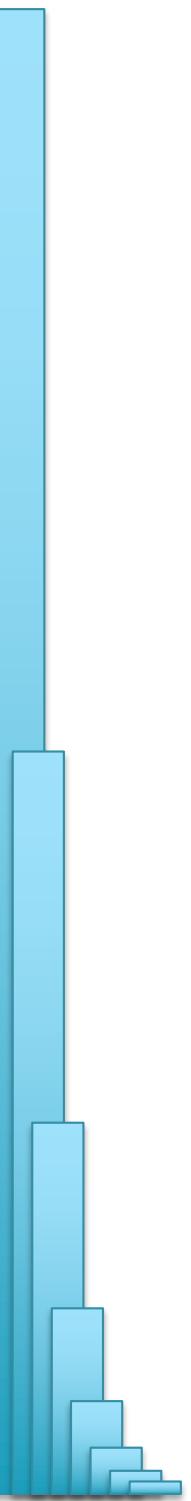
#### Packaging

The solutions to the above questions should be submitted as a single PDF document that includes your name, email address, and all relevant figures (as needed). Make

<https://github.com/schatzlab/biomedicalresearch2019>

# Part I: Recap

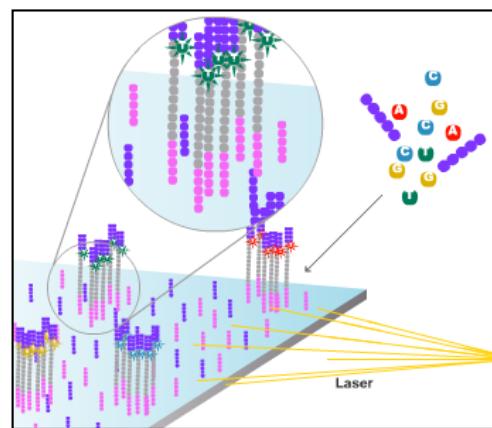
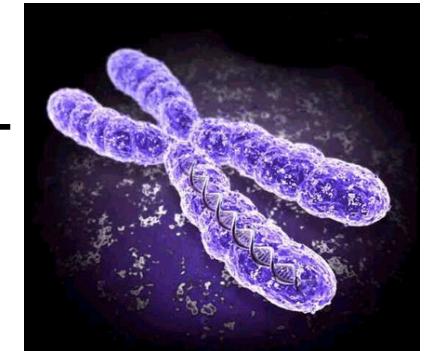
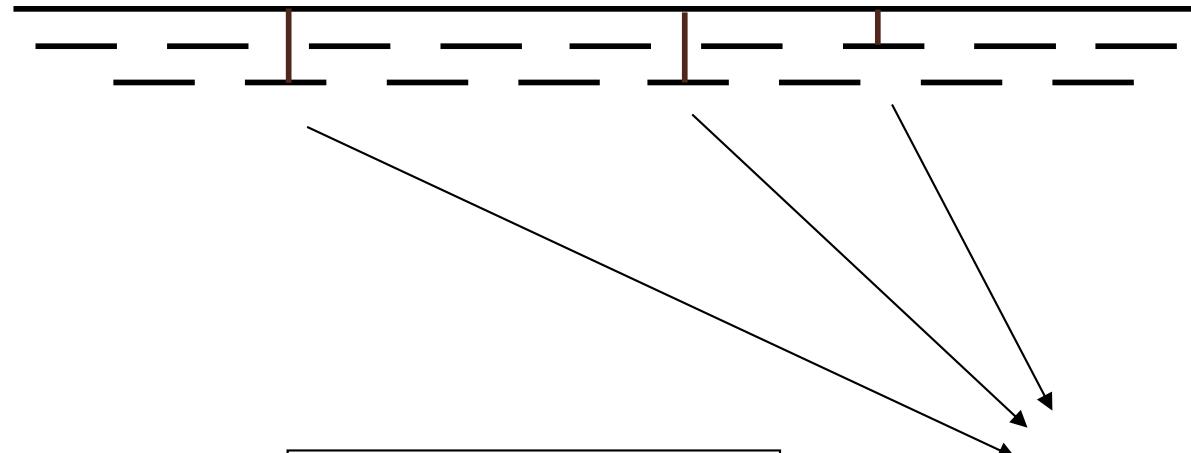




## Part 2: Read Mapping

# Personal Genomics

How does your genome compare to the reference?



Heart Disease

Cancer

Presidential Smile

# Searching for GATTACA

- Where is GATTACA in the human genome?
- Strategy I: Brute Force

I	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 I: Brute Force

I	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 I: Brute Force

I	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	...
		<b>G</b>	<b>A</b>	<b>T</b>	<b>T</b>	<b>A</b>	<b>C</b>	<b>A</b>	...						

No match at offset 3...

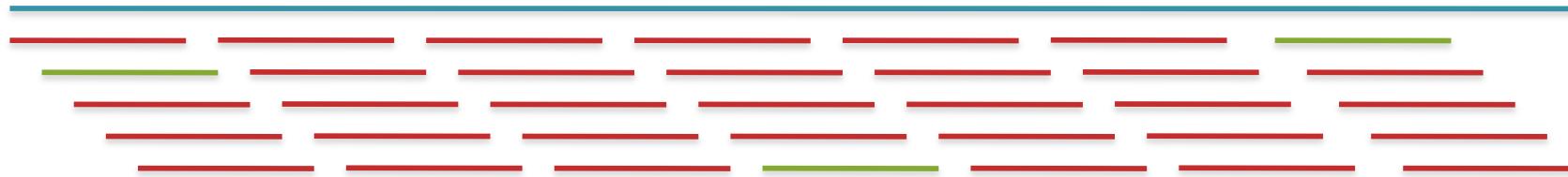
# Searching for GATTACA

- Where is GATTACA in the human genome?
- Strategy I: Brute Force

I	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



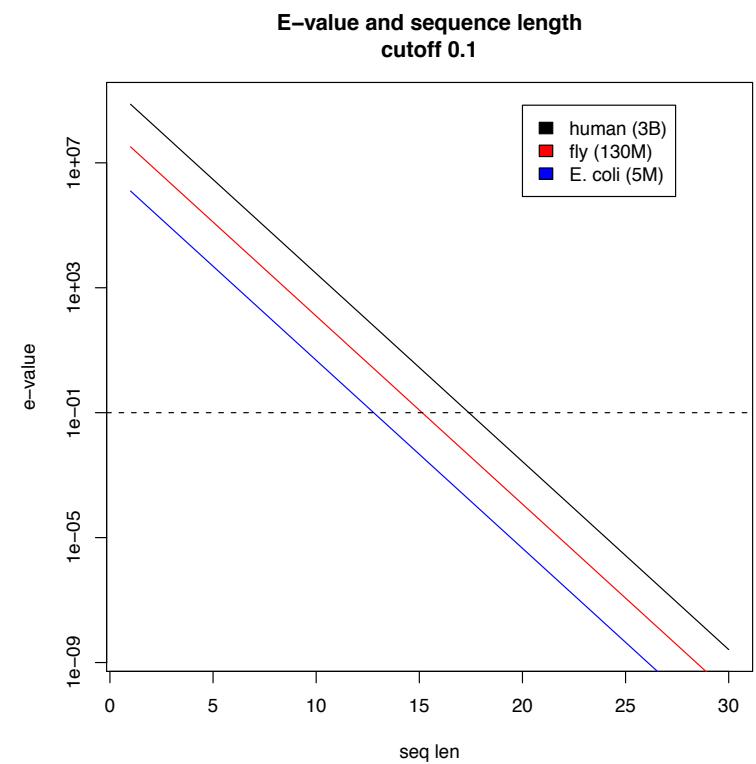
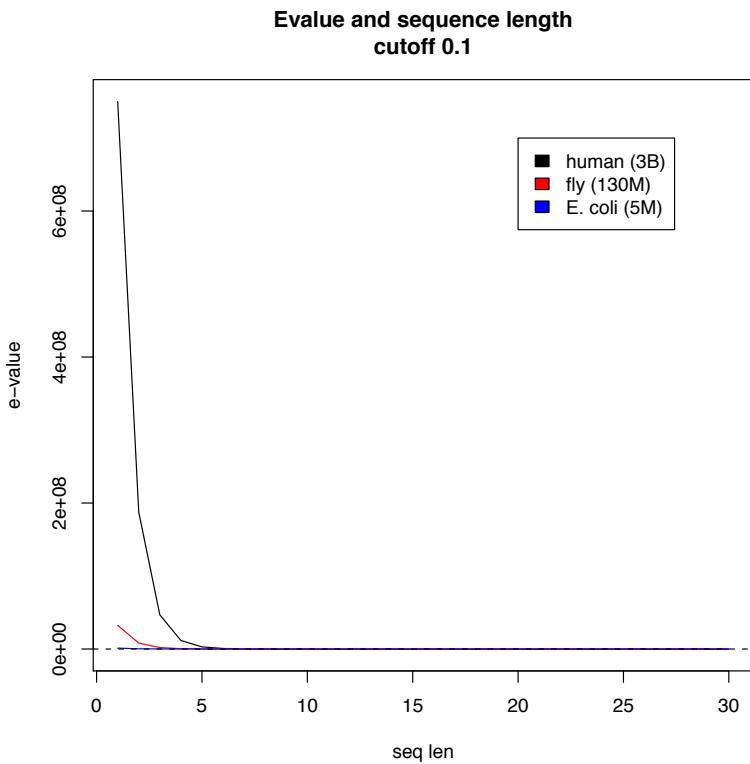
- Brute Force:
  - At every possible offset in the genome:
    - Do all of the characters of the query match?
- Analysis
  - Simple, easy to understand
  - Genome length =  $n$  [3B]
  - Query length =  $m$  [7]
  - Comparisons:  $(n-m+1) * m$  [2IB]
- Overall runtime:  $O(nm)$ 
  - [How long would it take if we double the genome size, read length?]
  - [How long would it take if we double both?]

# 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?]

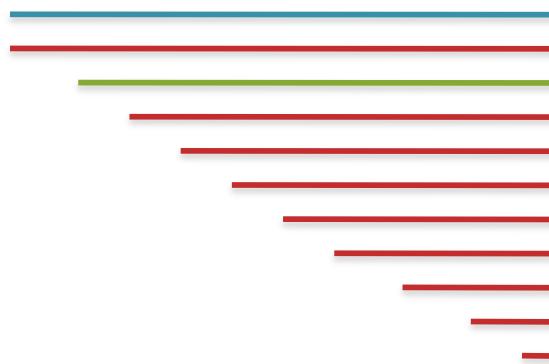
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
										G A T T A C C A					

- Improve runtime to  $O(n + m)$  [3B + 7]
  - If we double both, it just takes twice as long
  - Knuth-Morris-Pratt, 1977
  - Boyer-Moyer, 1977, 1991
- For one-off scans, this is the best we can do (optimal performance)
  - We have to read every character of the genome, and every character of the query
  - For short queries, runtime is dominated by the length of the genome

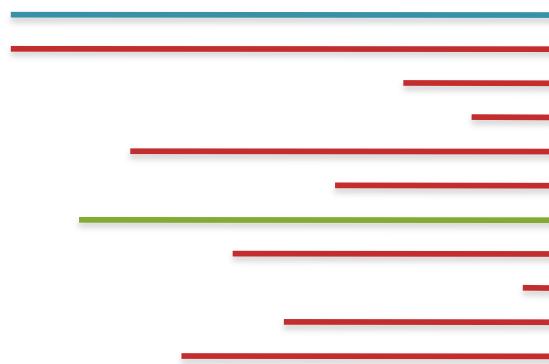
**How can we make this go faster?**

# 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;

#	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

#	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  
=> 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

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;

#	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

# 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	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo      Hi

# Binary Search Analysis

- Binary Search

Initialize search range to entire list

$\text{mid} = (\text{hi}+\text{lo})/2$ ;  $\text{middle} = \text{suffix}[\text{mid}]$

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:  $O(m \lg n)$

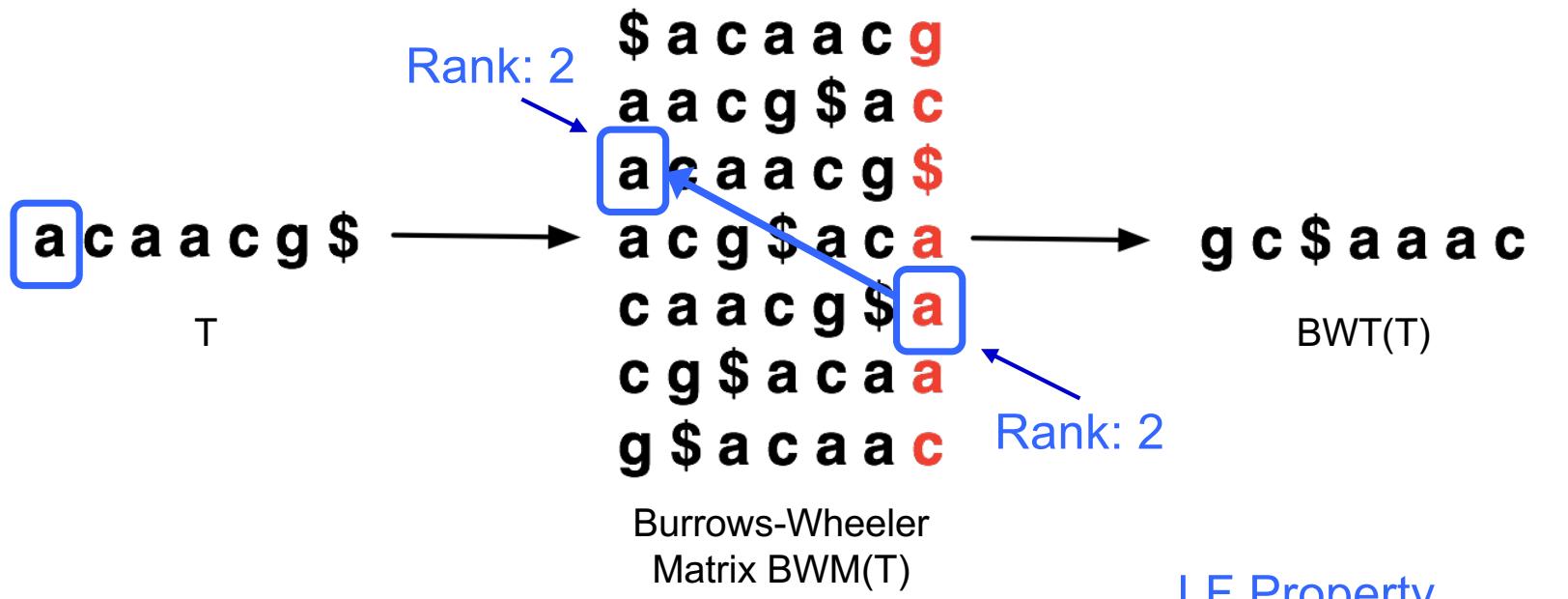
- More complicated, but **much** faster!

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

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

# Burrows-Wheeler Transform

- Reversible permutation of the characters in a text



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

LF Property  
implicitly encodes  
Suffix Array

A block sorting lossless data compression algorithm.

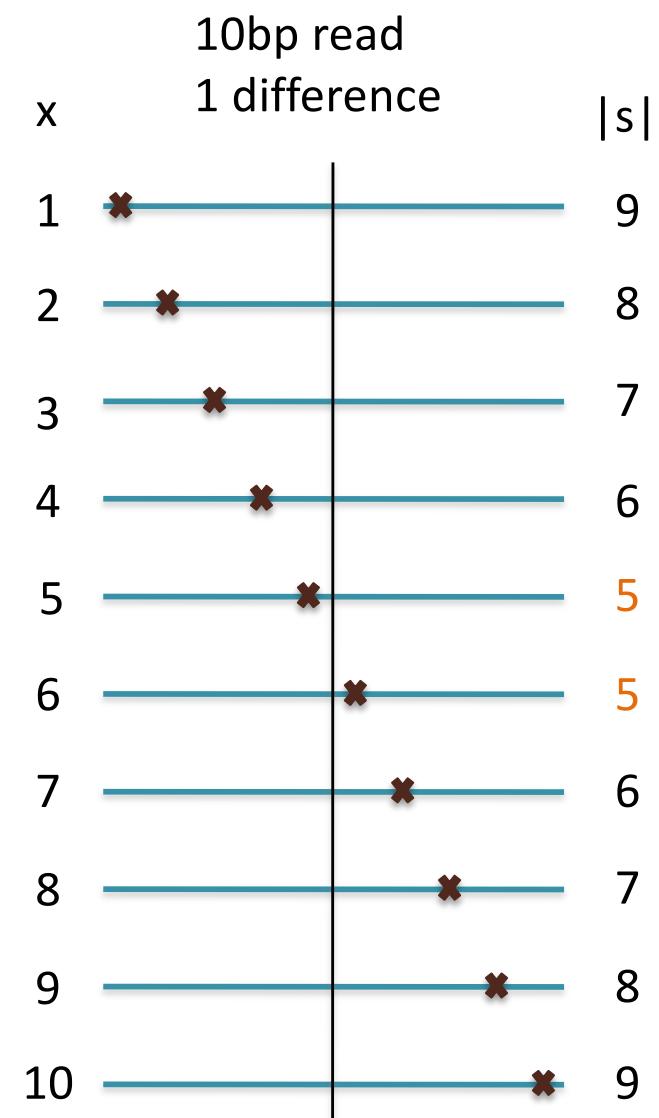
Burrows M, Wheeler DJ (1994) Digital Equipment Corporation. Technical Report 124

**What about in-exact alignment?**

# Seed-and-Extend Alignment

Theorem: An alignment of a sequence of length  $m$  with at most  $k$  differences **must** contain an exact match at least  $s=m/(k+1)$  bp long  
(Baeza-Yates and Perleberg, 1996)

- Proof: Pigeonhole principle
  - 1 pigeon can't fill 2 holes
- Seed-and-extend search
  - Use an index to rapidly find short exact alignments to seed longer in-exact alignments
    - BLAST, MUMmer, Bowtie, BWA, SOAP, ...
  - Specificity of the depends on seed length
    - Guaranteed sensitivity for  $k$  differences
    - Also finds some (but not all) lower quality alignments <- heuristic



# Similarity metrics

- Hamming distance
  - Count the number of substitutions to transform one string into another

MIKESCHATZ  
| | x | | xxxx |  
MICESHATZZ  
5

- Edit distance
  - The minimum number of substitutions, insertions, or deletions to transform one string into another

MIKESCHAT-Z  
| | x | | x | | | x |  
MICES-HATZZ

# Edit Distance Example

AGCACACCA → ACACACTA in 4 steps

AGCACACCA → (1. change G to C)

ACCACACCA → (2. delete C)

ACACACCA → (3. change A to T)

ACACACTT → (4. insert A after T)

ACACACTA → done

[Is this the best we can do?]

# Edit Distance Example

AGCACACCA → ACACACTA in 3 steps

AGCACACCA → (1. change G to C)

ACCACACCA → (2. delete C)

ACACACCA → (3. insert T after 3<sup>rd</sup> C)

ACACACTA → done

[Is this the best we can do?]

# Reverse Engineering Edit Distance

$$D(\text{AGCACACA}, \text{ ACACACTA}) = ?$$

Imagine we already have the optimal alignment of the strings, the last column can only be 1 of 3 options:

... <b>M</b>	... <b>I</b>	... <b>D</b>
...A	...-	...A
...A	...A	...-

The optimal alignment of last two columns is then 1 of 9 possibilities

... <b>MM</b>	... <b>IM</b>	... <b>DM</b>	... <b>MI</b>	... <b>II</b>	... <b>DI</b>	... <b>MD</b>	... <b>ID</b>	... <b>DD</b>
...CA	...-A	...CA	...A-	...--	...A-	...CA	...-A	...CA
...TA	...TA	...-A	...TA	...TA	...-A	...A-	...A-	...--

The optimal alignment of the last three columns is then 1 of 27 possibilities...

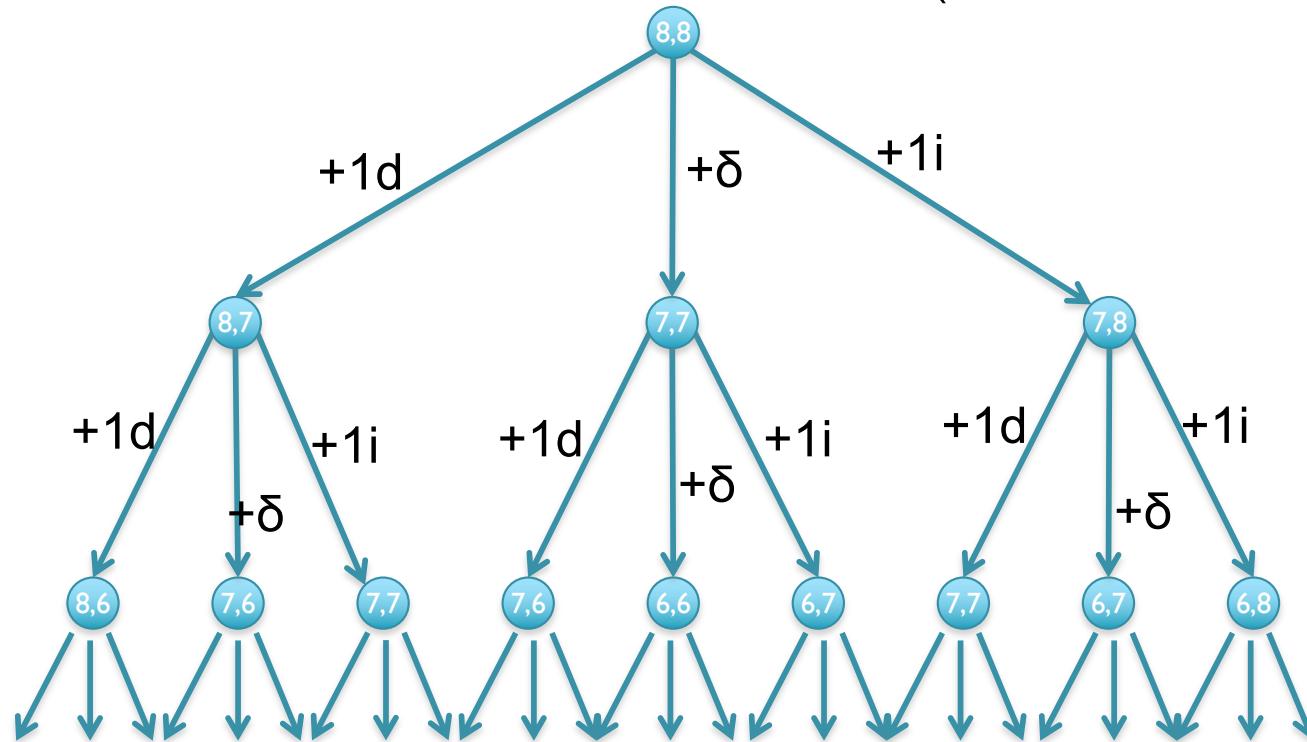
... <b>M</b> ...	... <b>I</b> ...	... <b>D</b> ...
...X...	...-...	...X...
...Y...	...Y...	...-...

Eventually spell out every possible sequence of {I,M,D}

# Recursive solution

- Computation of D is a recursive process.
  - At each step, we only allow matches, substitutions, and indels
  - $D(i,j)$  in terms of  $D(i',j')$  for  $i' \leq i$  and  $j' \leq j$ .

$$D(\text{AGCACACA}, \text{ACACACTA}) = \min\{D(\text{AGCACACA}, \text{ACACACT}) + 1, \\ D(\text{AGCACAC}, \text{ACACACTA}) + 1, \\ D(\text{AGCACAC}, \text{ACACACT}) + \delta(A, A)\}$$



[What is the running time?]

# Dynamic Programming

- We could code this as a recursive function call...  
...with an exponential number of function evaluations
- There are only  $(n+1) \times (m+1)$  pairs  $i$  and  $j$ 
  - We are evaluating  $D(i,j)$  multiple times
- Compute  $D(i,j)$  bottom up.
  - Start with smallest  $(i,j) = (1,1)$ .
  - Store the intermediate results in a table.
    - Compute  $D(i,j)$  after  $D(i-1,j)$ ,  $D(i,j-1)$ , and  $D(i-1,j-1)$

# Recurrence Relation for D

Find the edit distance (minimum number of operations to convert one string into another) in  $O(mn)$  time

- Base conditions:

- $D(i,0) = i$ , for all  $i = 0, \dots, n$
- $D(0,j) = j$ , for all  $j = 0, \dots, m$

- For  $i > 0, j > 0$ :

$$\begin{aligned} D(i,j) = \min \{ & \\ & D(i-1,j) + 1, \quad // \text{align 0 chars from S, 1 from T} \\ & D(i,j-1) + 1, \quad // \text{align 1 chars from S, 0 from T} \\ & D(i-1,j-1) + \delta(S(i), T(j)) // \text{align } i+1 \text{ chars} \end{aligned}$$

}

[Why do we want the min?]

# Dynamic Programming Matrix

		A	C	A	C	A	C	T	A
	0	I	2	3	4	5	6	7	8
A	I								
G	2								
C	3								
A	4								
C	5								
A	6								
C	7								
A	8								

[What does the initialization mean?]

# Dynamic Programming Matrix

		A	C	A	C	A	C	T	A
	0	I	2	3	4	5	6	7	8
A	I	0							
G	2								
C	3								
A	4								
C	5								
A	6								
C	7								
A	8								

$$D[A, A] = \min\{D[A, ] + 1, D[ , A] + 1, D[ , ] + \delta(A, A)\}$$

# Dynamic Programming Matrix

		A	C	A	C	A	C	T	A
	0	I	2	3	4	5	6	7	8
A	I	0	I						
G	2								
C	3								
A	4								
C	5								
A	6								
C	7								
A	8								

$$D[A, AC] = \min\{D[A, A]+1, D[AC]+1, D[A]+\delta(A, C)\}$$

# Dynamic Programming Matrix

		<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>T</b>	<b>A</b>
	0	I	2	3	4	5	6	7	8
<b>A</b>	I	0	I	2					
<b>G</b>	2								
<b>C</b>	3								
<b>A</b>	4								
<b>C</b>	5								
<b>A</b>	6								
<b>C</b>	7								
<b>A</b>	8								

$$D[A,ACA] = \min\{D[A,AC]+1, D[,ACA]+1, D[,AC]+\delta(A,A)\}$$

# Dynamic Programming Matrix

		A	C	A	C	A	C	T	A
	0	1	2	3	4	5	6	7	8
A	I	0	I	2	3	4	5	6	7
G	2								
C	3								
A	4								
C	5								
A	6								
C	7								
A	8								

$$D[A, ACACACTA] = 7$$

-----A

\*\*\*\*\* |

ACACACTA

[What about the other A?]

# Dynamic Programming Matrix

		<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>T</b>	<b>A</b>
	<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>
<b>A</b>	I	0	I	2	3	<u>4</u>	5	6	7
<b>G</b>	2	I	I	2	3	4	<u>5</u>	<u>6</u>	<u>7</u>
<b>C</b>	3								
<b>A</b>	4								
<b>C</b>	5								
<b>A</b>	6								
<b>C</b>	7								
<b>A</b>	8								

$$D[AG, ACACACTA] = 7$$

----AG--

\*\*\* | \*\*\*

ACACACTA

# Dynamic Programming Matrix

		A	C	A	C	A	C	T	A
	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
G	2	1	1	2	3	4	5	6	7
C	3	2	1	2	2	3	4	5	6
A	4	3	2	1	2	2	3	4	5
C	5	4	3	2	1	2	2	3	4
A	6	5	4	3	2	1	2	3	3
C	7	6	5	4	3	2	1	2	3
A	8	7	6	5	4	3	2	2	2

$$D[AGCACACA, ACACACTA] = 2$$

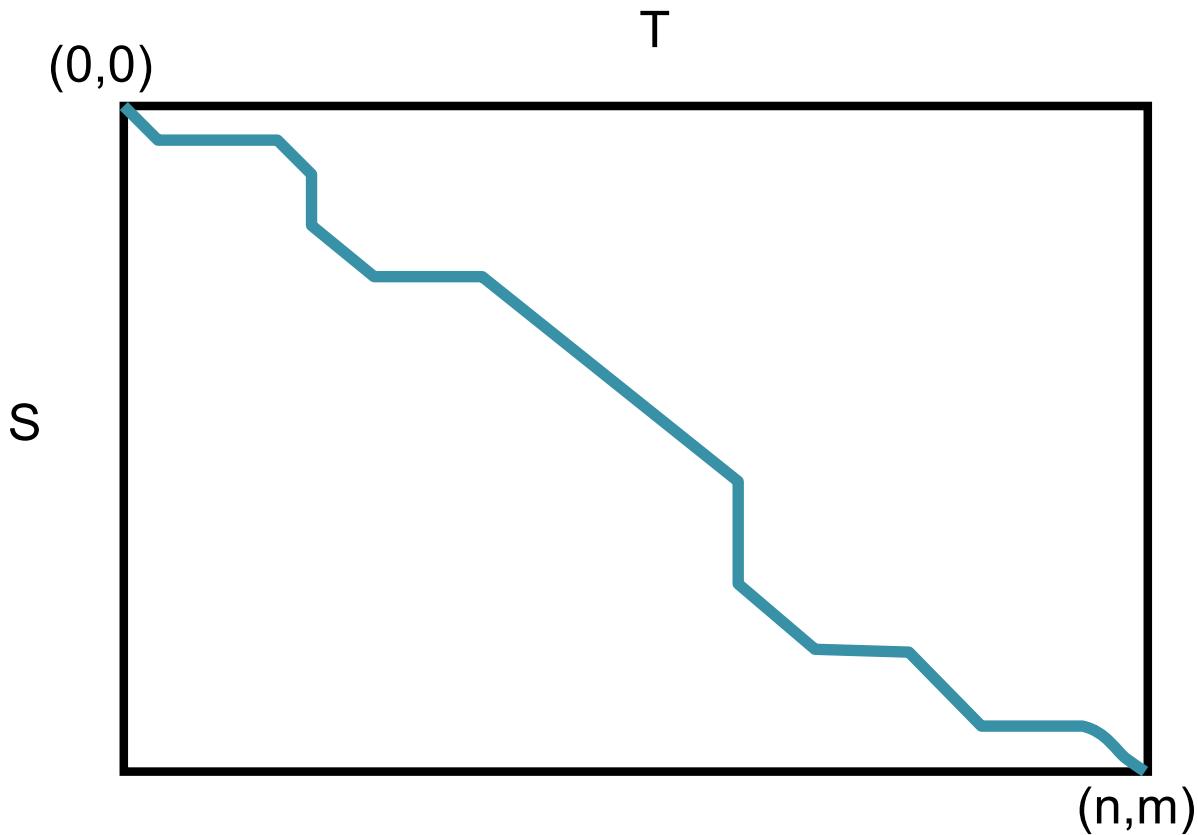
AGCACAC-A

| \* | | | | \* |

A-CACACTA

[Can we do it any better?]

# Global Alignment Schematic



- A high quality alignment will stay close to the diagonal
  - If we are only interested in high quality alignments, we can skip filling in cells that can't possibly lead to a high quality alignment
  - Find the global alignment with at most edit distance  $d$ :  $O(2dn)$

# Alignment Ambiguity

Notice that the edit distance of GATTACA and GATACA is 2,  
but there are multiple possible optimal alignments:

GATTACA

GAT--ACA

GATTACA

GA-T-ACA

GATTACA

GA--TACA

Which one is right?

# Algorithm Overview

## 1. Split read into segments

Read (reverse complement)

**Policy:** extract 16 nt seed every 10 nt

Seeds

+ , 0: <b>CCAGTAGCTCTCAGCC</b>	- , 0: <b>TACAGGCCTGGGTAAA</b>
+ , 10: <b>TCAGCCTTATTTACC</b>	- , 10: <b>GGTAAAATAAGGCTGA</b>
+ , 20: <b>TTTACCCAGGCCTGTA</b>	- , 20: <b>GGCTGAGAGCTACTGG</b>

2. Lookup each segment and prioritize

The diagram illustrates the workflow for aligning DNA sequences. It starts with a list of **Seeds** on the left, which are then processed by an **Ungapped alignment with FM Index** step in the middle. The output of this step is a sequence alignment with color-coded matches and mismatches. Finally, the aligned sequences are converted into **Seed alignments (as B ranges)** on the right.

Seeds	Ungapped alignment with FM Index	Seed alignments (as B ranges)
+, 0: <b>CCAGTAGCTCTCAGCC</b>	<b>CCAGTAGCTCTCAGCC</b>	{ [211, 212], [212, 214] }
+, 10: <b>TCAAGCCTTATTTACCC</b>	<b>TCAAGCCTTATTTACCC</b>	{ [653, 654], [651, 653] }
+, 20: <b>TTTACCCCAGGCCTGTA</b>	<b>TTTACCCCAGGCCTGTA</b>	{ [684, 685] }
-, 0: <b>TACAGGCCTGGGTAAA</b>	<b>TACAGGCCTGGGTAAA</b>	{ }
-, 10: <b>GGTAAAATAAGGCTGA</b>	<b>GGTAAAATAAGGCTGA</b>	{ }
-, 20: <b>GGCTGAGAGCTACTGG</b>	<b>GGCTGAGAGCTACTGG</b>	{ [624, 625] }

### 3. Evaluate end-to-end match

The diagram illustrates the workflow of a SIMD dynamic programming aligner. It starts with a list of extension candidates on the left, which are processed by the aligner (represented by a central box) to produce SAM alignments on the right.

**Extension candidates**

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

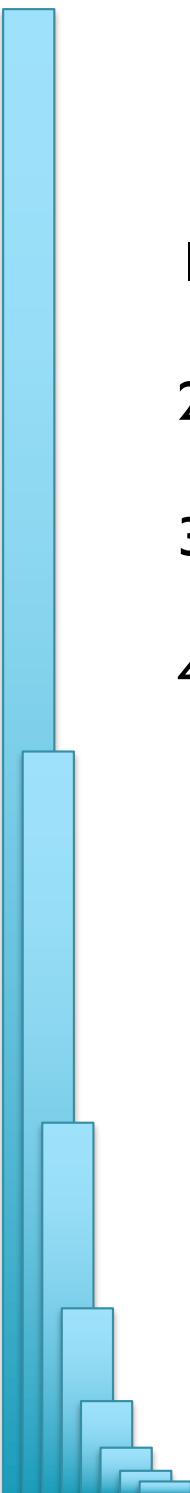
**SIMD dynamic programming aligner**

**SAM alignments**

r1	0	chr12	1936	0
	36M	*	0	0
CCAGTAGCTCTAGCCTTATTTACCCAGGCCTGTA				
II				
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 & S)



# Next Steps

1. Reflect on the magic and power of DNA 😊
2. Check out the course webpage
3. Submit HW2
4. Work on HW3