

# Read Mapping

Michael Schatz

Feb 13, 2013

Lecture 6: Applied Comparative Genomics



# Assignment 2: Genome Assembly

Due Wednesday Feb 13 @ 11:59pm

- 1. Setup Docker/VirtualBox/Ubuntu**
- 2. Initialize Tools**
- 3. Download Reference Genome & Reads**
- 4. Decode the secret message**
  1. Estimate coverage, check read quality
  2. Check kmer distribution
  3. Assemble the reads with spades
  4. Align to reference with MUMmer
  5. Extract foreign sequence
  6. `dna-encode.pl -d`

<https://github.com/schatzlab/appliedgenomics2019/blob/master/assignments/assignment2/README.md>



# Assignment 3: Due Monday Feb 25

## Assignment 3: Coverage, Genome Assembly, and the BWT

Assignment Date: Monday, Feb. 18, 2019

Due Date: Monday, Feb. 25, 2019 @ 11:59pm

### Question 1. Coverage simulator [10 pts]

- Q1a. How many 100bp reads are needed to sequence a 1Mbp genome to 5x coverage?
- Q1b. In the language of your choice, simulate sequencing 5x coverage of a 1Mbp genome and plot the histogram of coverage. Note you do not need to actually output the sequences of the reads, you can just randomly sample positions in the genome and record the coverage. You do not need to consider the strand of each read. The start position of each read should have a uniform random probability at each possible starting position (1 through 999,900). You can record the coverage in an array of 1M positions. Overlay the histogram with a Poisson distribution with  $\lambda=5$
- Q1c. Using the histogram from 1b, how much of the genome has not been sequenced (has 0x coverage). How well does this match Poisson expectations?
- Q1d. Now repeat the analysis with 15x coverage: 1. simulate the appropriate number of reads, 2. make a histogram, 3. overlay a Poisson distribution with  $\lambda=15$ , 4. compute the number of bases with 0x coverage, and 5. evaluate how well it matches the Poisson expectation.

### Question 2. de Bruijn Graph construction [10 pts]

- Q2a. Draw (by hand or by 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)

ATTC  
ATTG  
CATT  
CTTA  
GATT  
TATT  
TCAT  
TCTT

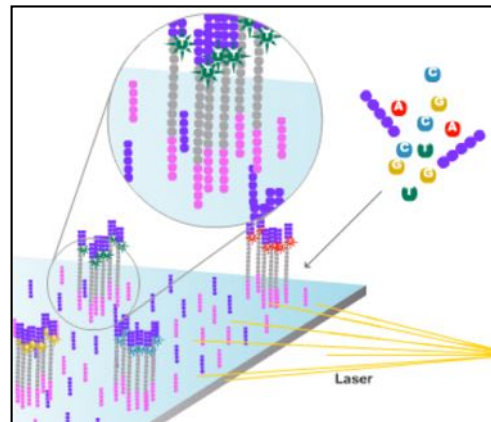
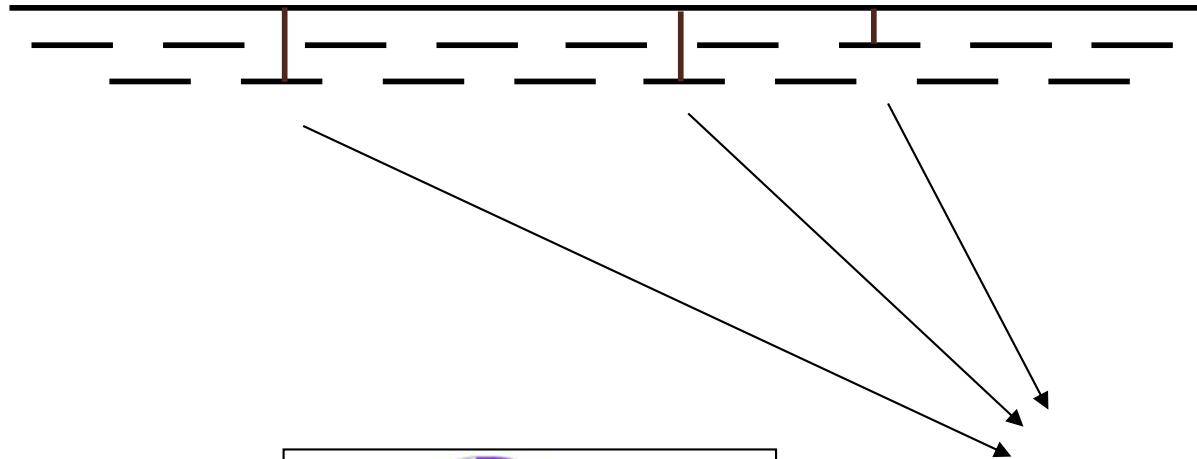




# Part I: Suffix Arrays

# Personal Genomics

How does your genome compare to the reference?



Heart Disease

Cancer

Presidential smile

# Brute Force Analysis



- Brute Force:
  - At every possible offset in the genome:
    - Do all of the characters of the query match?
- Analysis
  - Simple, easy to understand
  - Genome length =  $n$  [3B]
  - Query length =  $m$  [7]
  - Comparisons:  $(n-m+1) * m$  [21B]
- Overall runtime:  $O(nm)$ 
  - [How long would it take if we double the genome size, read length?]
  - [How long would it take if we double both?]

# Brute Force Reflections

Why check every position?

- GATTACA can't possibly start at position 15

[WHY?]

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

- Improve runtime to  $O(n + m)$

[3B + 7]

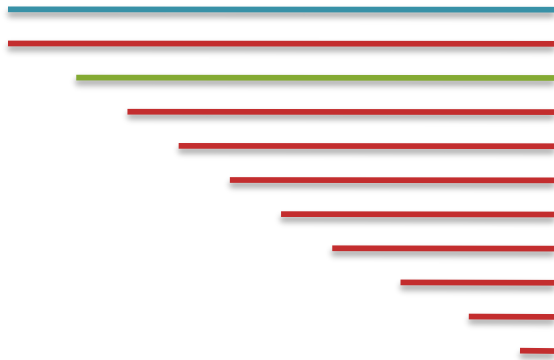
- If we double both, it just takes twice as long
- Knuth-Morris-Pratt, 1977
- Boyer-Moyer, 1977, 1991

- For one-off scans, this is the best we can do (optimal performance)

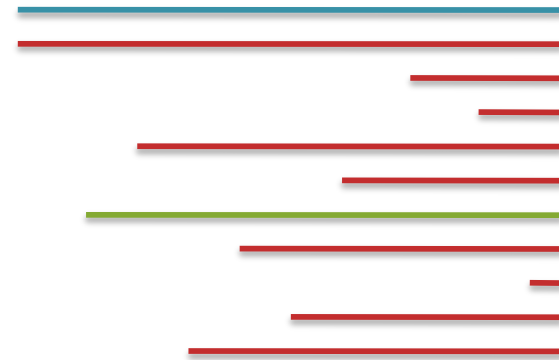
- We have to read every character of the genome, and every character of the query
- For short queries, runtime is dominated by the length of the genome

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

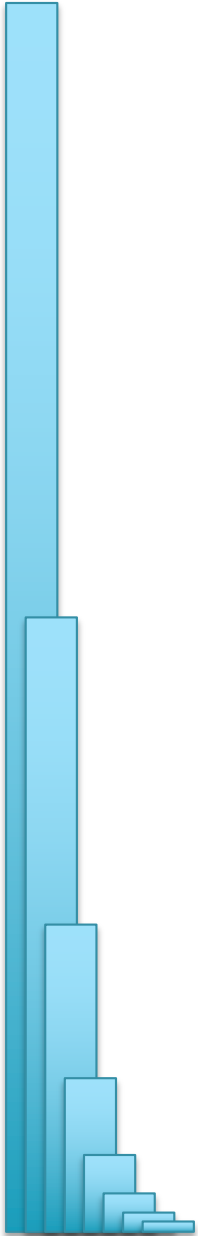
[32]

- 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?]



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

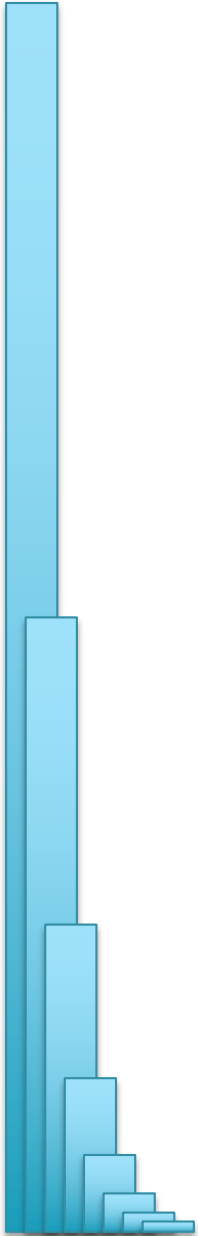
[32]

- Total Runtime:  $O(m \lg n)$

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

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

Can be reduced to  $O(m + \lg n)$   
using an auxiliary data structure called the LCP array



# 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

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