GeneGorman
Short Read Mapper

Delaney Miller, Gal Kadmon, Henry Tran,
Lauren Hu, Yanni (Michelle) Pang

Gene Gorman Team Members & Roles

Gal Kadmon

Role 1: Project Manager
Role 2: Documentation

Lauren Hu

Role 1: Testing
Role 2: Input/Output

Delaney Miller

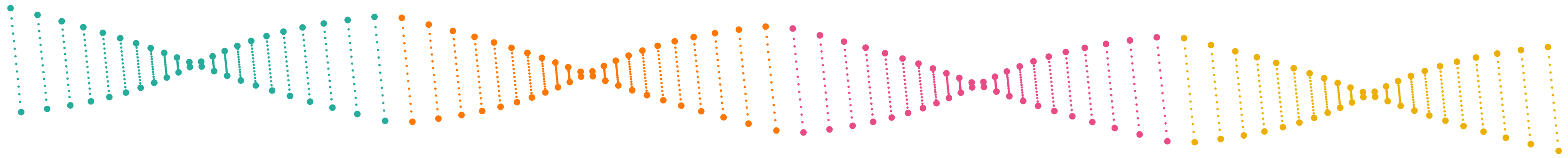
Role 1: Documentation
Role 2: Project Manager

Yanni Pang

Role 1: Input/Output
Role 2: Algorithms

Henry Tran

Role 1: Algorithms
Role 2: Testing



Communication

Slack

Resources, non urgent updates

Text

Urgent Questions

Drive

Collaborative document storage

High-Level Overview of our Approach

Preprocessing

Handles parsing the input reads & building the suffix array which is used in mapper

Read Mapper

Splits reads into seeds and finds exact matches using a suffix array. Then, uses a matching approach that allows for missing seeds to find the best mapping

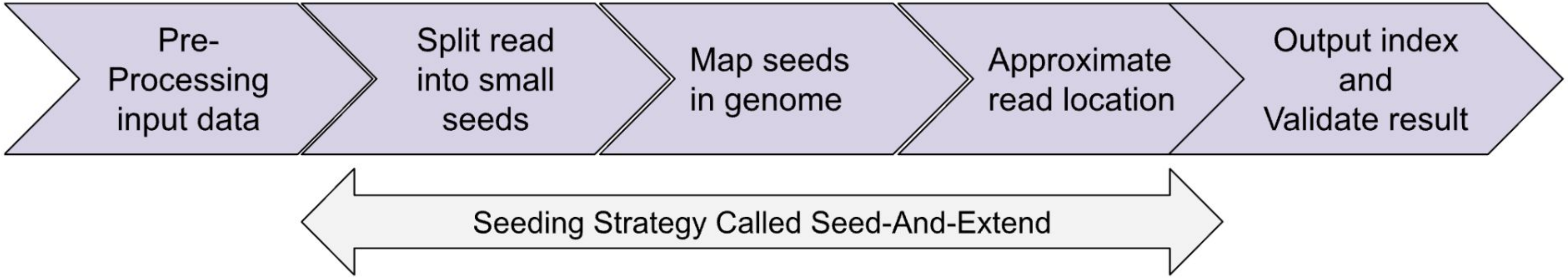
Validator

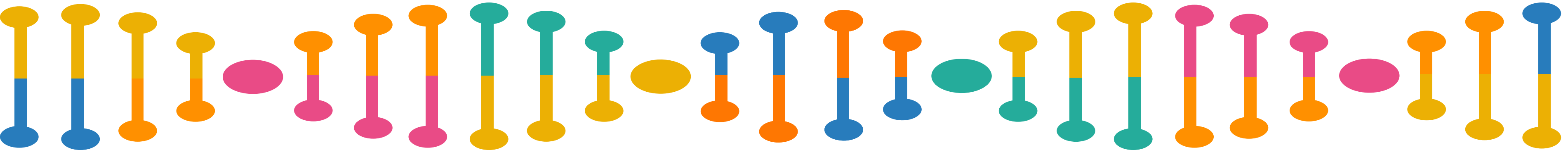
Compares our predicted mappings with the ground truth table and calculates metrics to assess the quality, time, and memory usage of our read mapping.

Output

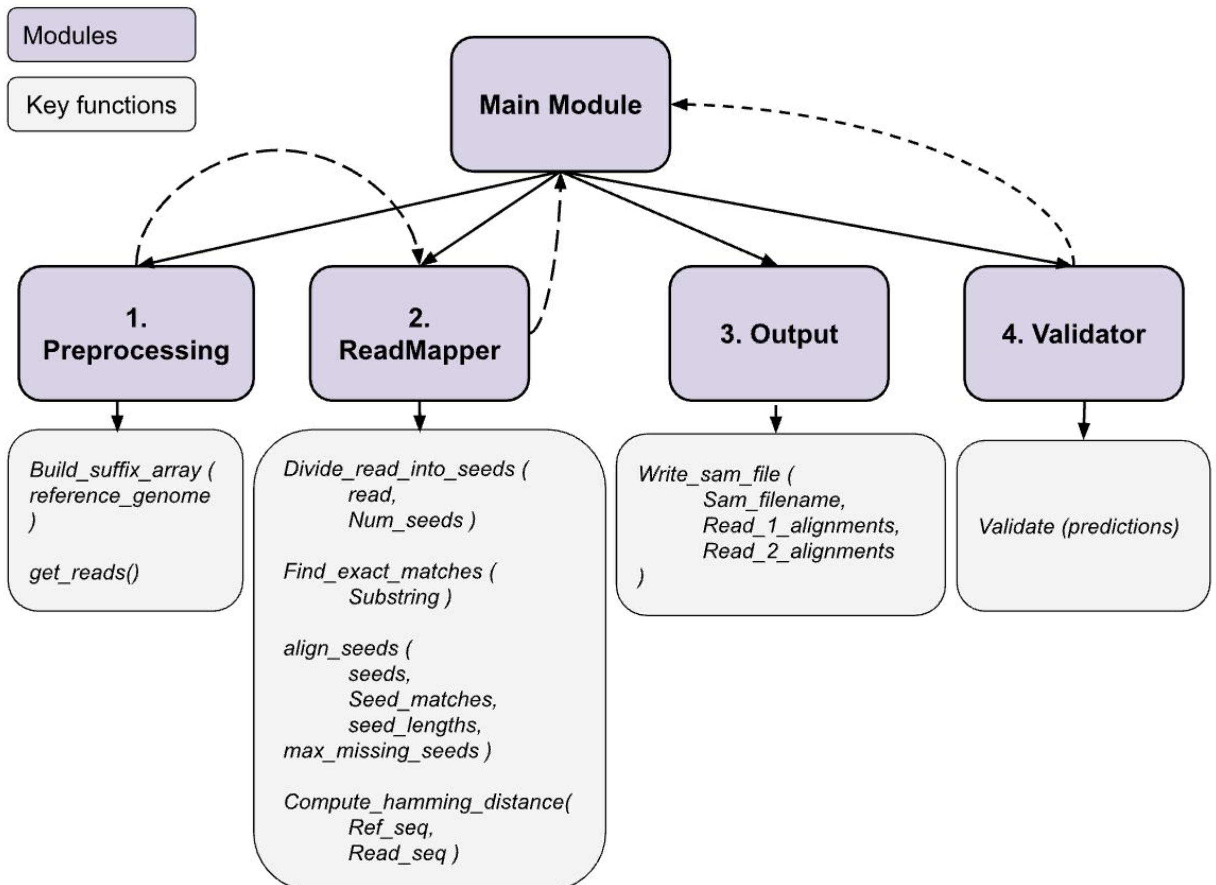
Formats our mapping into the correctly formatted SAM file

High Level Diagram


Pre-Processing input data
Split read into small seeds
Map seeds in genome
Approximate read location
Output index and Validate result
Seeding Strategy Called Seed-And-Extend



Data Flow Diagram



```
Build_suffix_array (
reference_genome
)

get_reads()
```

```
Divide_read_into_seeds (
        read,
        Num_seeds )

Find_exact_matches (
        Substring )

align_seeds (
        seeds,
        Seed_matches,
        seed_lengths,
max_missing_seeds )

Compute_hamming_distance(
        Ref_seq,
        Read_seq )
```

```
Write_sam_file (
        Sam_filename,
        Read_1_alignments,
        Read_2_alignments
)
```

```
Validate (predictions)
```

Strategy in Preprocessor

Step 1 Define the preprocessor class with methods to be used in main.

Step 2 Validate input files to check if they are empty or improperly formatted.

Step 3 Parse the input files to extract the reads and format them into dictionaries mapping read IDs to the corresponding sequences

Step 4 Build the suffix array from the reference genome

Strategy in Readmapper

Step 1 Define the readmapper class with methods to be used in main

Step 2 For each read, divide it into seeds. We found that 25 was a good balance between speed and accuracy

Step 3 Search for exact matches of every seed in the reference genome by searching the suffix array

Step 4 To find exact matches in the suffix array quickly, we use a binary search algorithm

Strategy in Readmapper

Step 5

For all exact matches of these seeds, we calculate the corresponding potential alignment region

Step 6

For each potential alignment region, we calculate how many seeds matched to that same region. Regions with more matching seeds are more likely to be correct

Step 7

We set a minimum threshold of required seeds, so the mapping quality stays high. For all potential alignments that surpass this threshold, we take the alignment with the smallest hamming distance.

Step 8

For alignments that do not pass the threshold, we considered it an unaligned read. We do this for both the forward and reverse strands, and return the best.

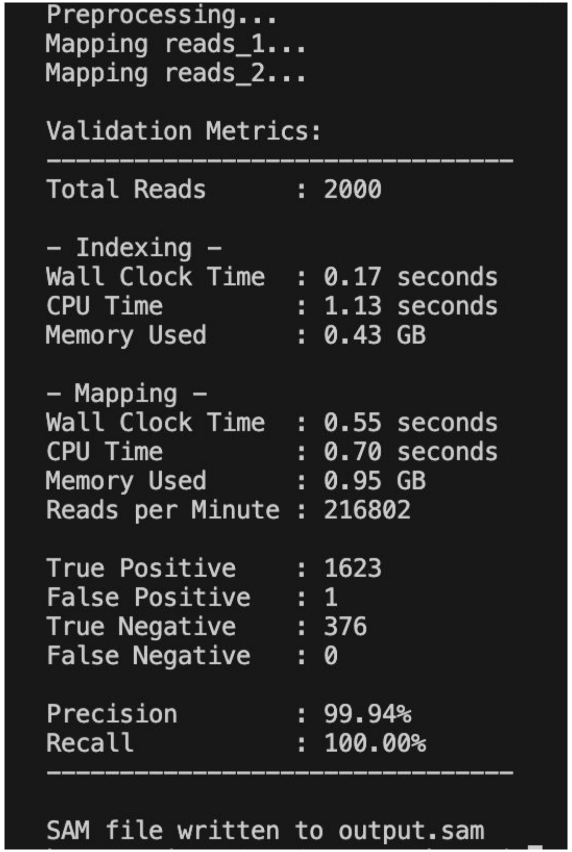
Input/Output & Parsing Libraries Used

FastQ

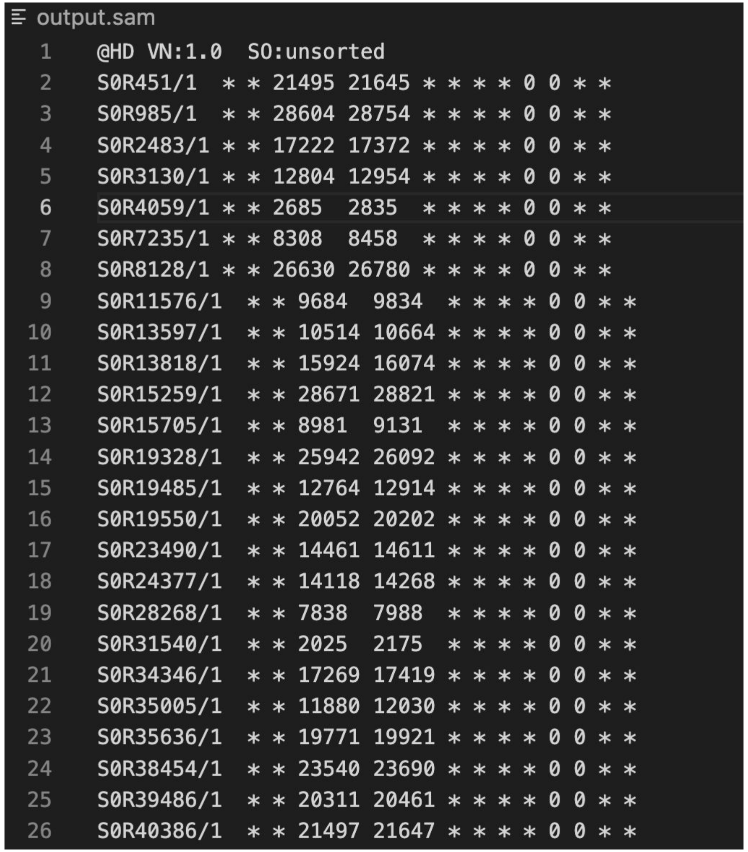
SeqIO

sys

Current Metrics and Output



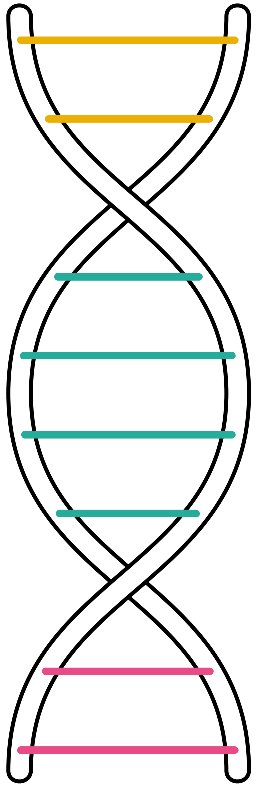
```
Preprocessing...
Mapping reads_1...
Mapping reads_2...

Validation Metrics:
--------------------------------
Total Reads       : 2000

- Indexing -
Wall Clock Time   : 0.17 seconds
CPU Time          : 1.13 seconds
Memory Used       : 0.43 GB

- Mapping -
Wall Clock Time   : 0.55 seconds
CPU Time          : 0.70 seconds
Memory Used       : 0.95 GB
Reads per Minute  : 216802

True Positive     : 1623
False Positive    : 1
True Negative     : 376
False Negative    : 0

Precision         : 99.94%
Recall            : 100.00%
--------------------------------

SAM file written to output.sam
```



output.sam

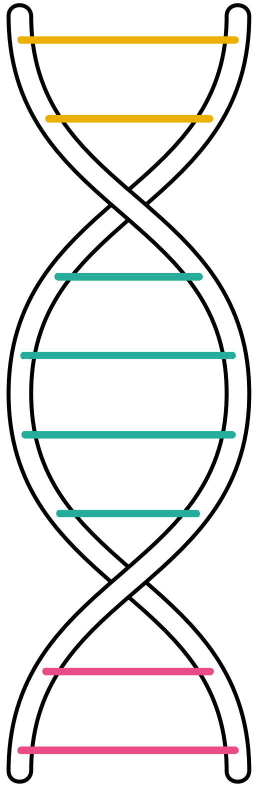
```
@HD VN:1.0  SO:unsorted
S0R451/1  * * 21495 21645 * * * * 0 0 * *
S0R985/1  * * 28604 28754 * * * * 0 0 * *
S0R2483/1 * * 17222 17372 * * * * 0 0 * *
S0R3130/1 * * 12804 12954 * * * * 0 0 * *
S0R4059/1 * * 2685  2835  * * * * 0 0 * *
S0R7235/1 * * 8308  8458  * * * * 0 0 * *
S0R8128/1 * * 26630 26780 * * * * 0 0 * *
S0R11576/1  * * 9684  9834  * * * * 0 0 * *
S0R13597/1  * * 10514 10664 * * * * 0 0 * *
S0R13818/1  * * 15924 16074 * * * * 0 0 * *
S0R15259/1  * * 28671 28821 * * * * 0 0 * *
S0R15705/1  * * 8981  9131  * * * * 0 0 * *
S0R19328/1  * * 25942 26092 * * * * 0 0 * *
S0R19485/1  * * 12764 12914 * * * * 0 0 * *
S0R19550/1  * * 20052 20202 * * * * 0 0 * *
S0R23490/1  * * 14461 14611 * * * * 0 0 * *
S0R24377/1  * * 14118 14268 * * * * 0 0 * *
S0R28268/1  * * 7838  7988  * * * * 0 0 * *
S0R31540/1  * * 2025  2175  * * * * 0 0 * *
S0R34346/1  * * 17269 17419 * * * * 0 0 * *
S0R35005/1  * * 11880 12030 * * * * 0 0 * *
S0R35636/1  * * 19771 19921 * * * * 0 0 * *
S0R38454/1  * * 23540 23690 * * * * 0 0 * *
S0R39486/1  * * 20311 20461 * * * * 0 0 * *
S0R40386/1  * * 21497 21647 * * * * 0 0 * *
```

Results for Indexing (2000 reads)



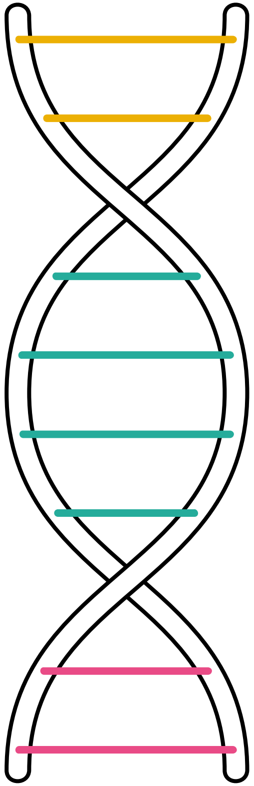
1	Wall Clock Time	0.17 seconds
2	CPU Time	1.13 seconds
3	Memory Usage	0.43 GB

Results for Mapping (2000 reads)



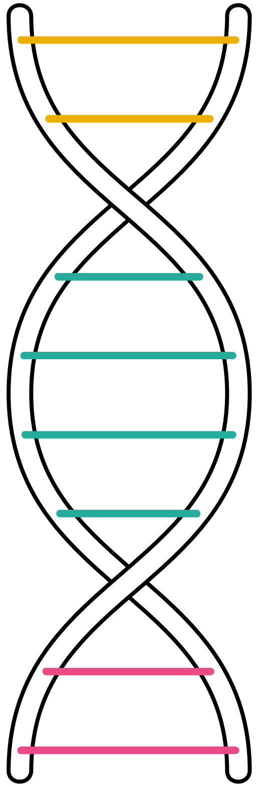
1	Wall Clock Time	0.55 seconds
2	CPU Time	0.70 seconds
3	Memory Usage	0.95 GB

Precision & Recall (2000 reads)



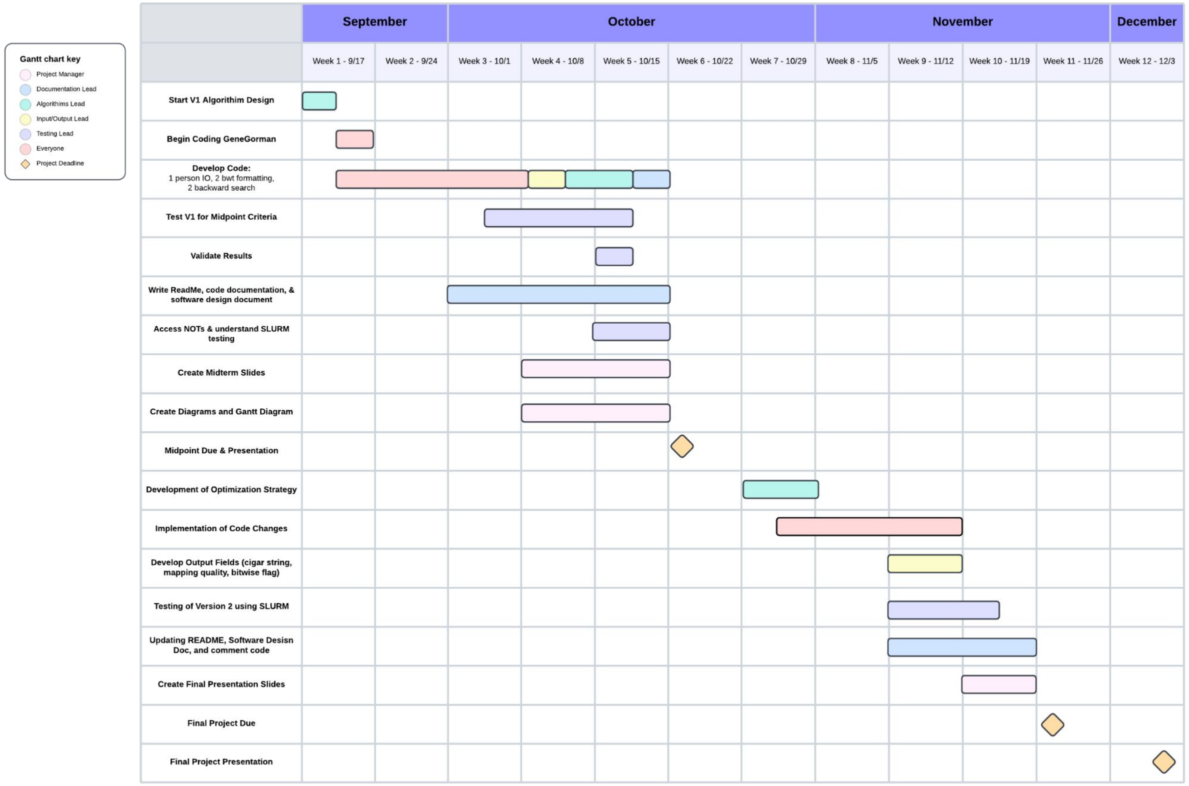
1	Precision	99.4%
2	Recall	100%
3	Reads per Minute	216,802

True/False Positives & Negatives (2000 Reads)



1	True Positive	1623
2	False Positive	1
3	True Negative	376
4	False Negative	0

Next Steps - Gantt Chart

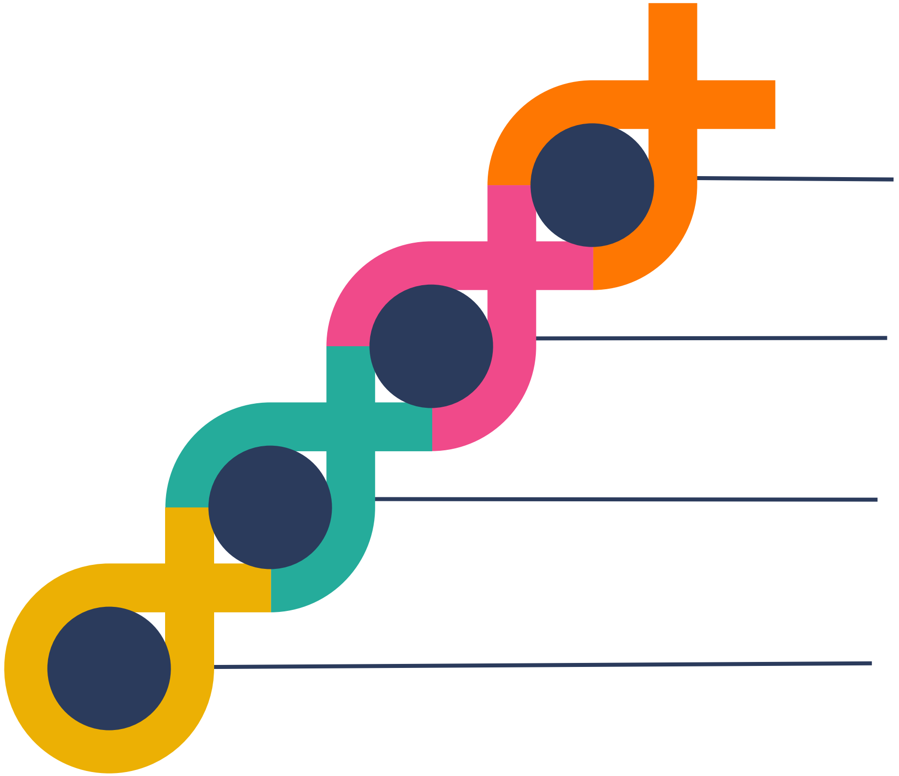


COMP 416: GeneGorman

Gantt chart key
- ⬜ Project Manager
- 🟦 Documentation Lead
- 🟩 Algorithims Lead
- 🟨 Input/Output Lead
- 🟪 Testing Lead
- 🟥 Everyone
- 🔶 Project Deadline

	September		October					November				December
	Week 1 - 9/17	Week 2 - 9/24	Week 3 - 10/1	Week 4 - 10/8	Week 5 - 10/15	Week 6 - 10/22	Week 7 - 10/29	Week 8 - 11/5	Week 9 - 11/12	Week 10 - 11/19	Week 11 - 11/26	Week 12 - 12/3
Start V1 Algorithim Design	🟩											
Begin Coding GeneGorman	🟥											
Develop Code: 1 person IO, 2 bwt formatting, 2 backward search	🟥	🟥	🟥	🟨🟩	🟩🟦							
Test V1 for Midpoint Criteria			🟪	🟪	🟪							
Validate Results					🟪							
Write ReadMe, code documentation, & software design document			🟦	🟦	🟦							
Access NOTs & understand SLURM testing					🟪							
Create Midterm Slides				⬜	⬜							
Create Diagrams and Gantt Diagram				⬜	⬜							
Midpoint Due & Presentation						🔶						
Development of Optimization Strategy							🟩					
Implementation of Code Changes							🟥	🟥	🟥			
Develop Output Fields (cigar string, mapping quality, bitwise flag)									🟨			
Testing of Version 2 using SLURM									🟪	🟪		
Updating README, Software Desisn Doc, and comment code									🟦	🟦		
Create Final Presentation Slides										⬜		
Final Project Due											🔶	
Final Project Presentation												🔶

Next Steps - Algorithm Development



01

BWT

We plan to replace our usage of a suffix array and instead use a BWT structure.

02

Parallelization

Right now, algorithm runs sequentially, but adding parallelization will save time.

03

Cigar String & Output

Filling in the SAM file fields - mapping quality, cigar string, and bitwise flag

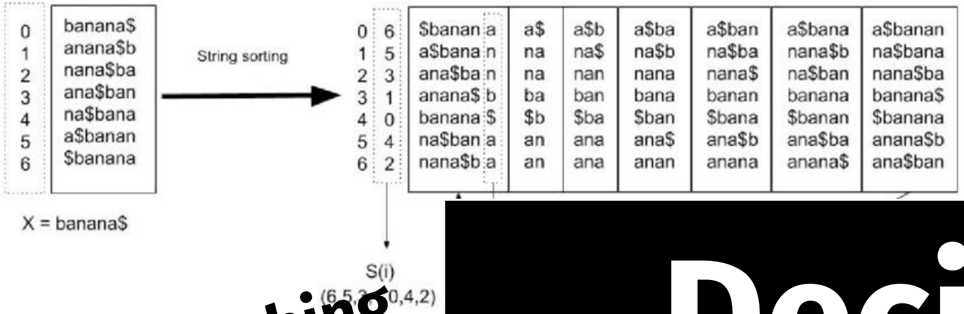
04

Weigh Alignments by Quality Scores

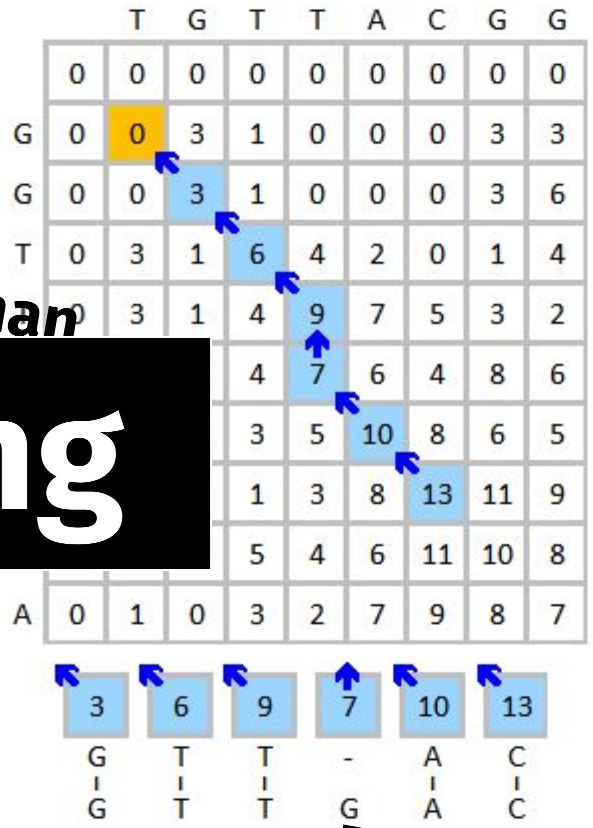
Right now we use generic hamming distance, but we want to consider using quality scores to weigh the distance calculation

Challenges?

BWT



DP?



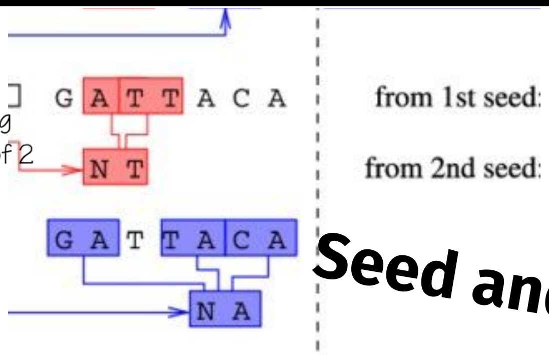
Smith-waterman

Hashing

.... Deciding

Hamming

```
G A G A C T C A T
X         X
T A G A C G C A T
```



Seed and extend

BOWTIE

Thank You & Questions?