

Introduction to Sequence Alignment

Manpreet S. Katari

Outline

1. Global vs. local approaches to aligning sequences
 1. Dot Plots
2. BLAST
 1. Dynamic Programming
3. Hash Tables
 1. BLAT
4. BWT (Burrow Wheeler's Transformation)
 1. Bowtie, BWA
 2. Some slides borrowed from Stratos
5. Trimming sequences
 1. Trimmomatic

Global and local approaches to aligning sequences

GLOBAL: Attempt to “match” and assess similarity between two entire sequences

LOCAL: Find subsequences of high similarity

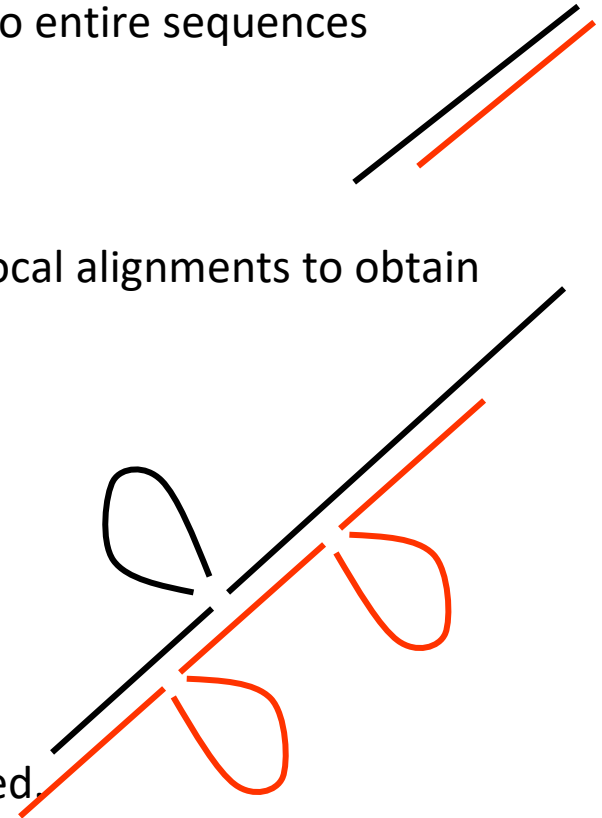
... and then possibly “stick” (chain, net, thread) together local alignments to obtain an overall comparison of the original sequences.

The second approach is more meaningful

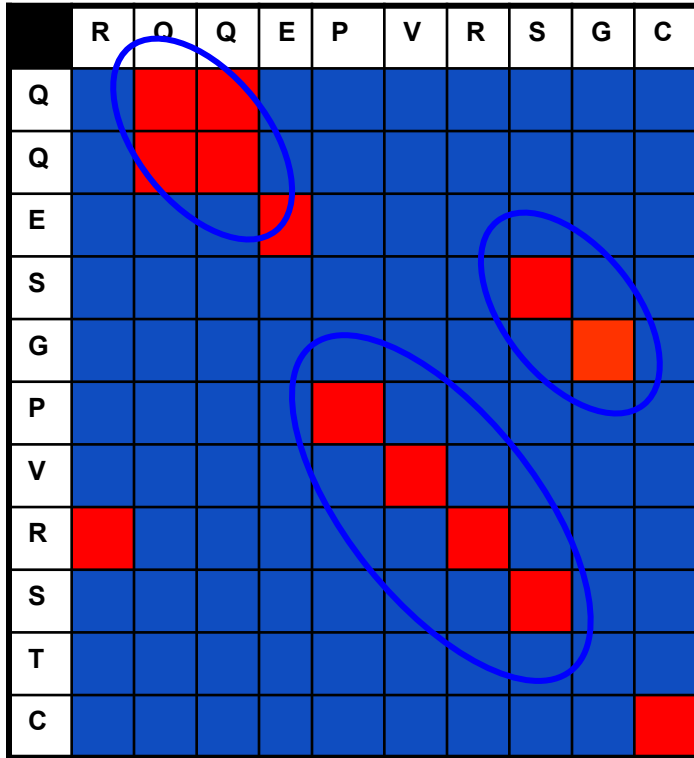
(especially for long sequences, of different lengths, like whole genomes)

Two protein or DNA sequences are unlikely to present a straightforward overall “match”, even if they are closely related.

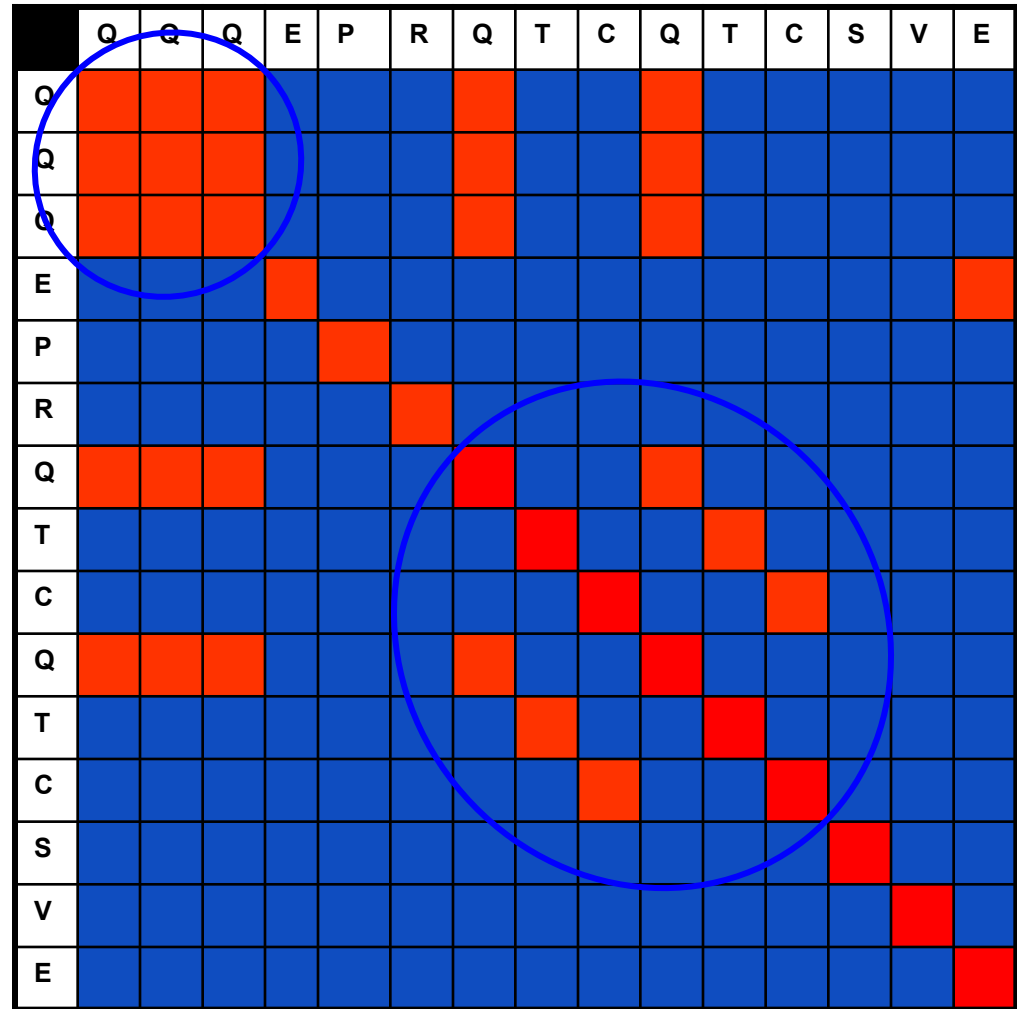
Why? Substitutions are not the only process by which they diverge: insertions, deletions and rearrangements



Visualizing pair-wise alignments: Dot plots (matrices)



A dot for every match... visualize regions of alignment between two sequences, even when not in same position.

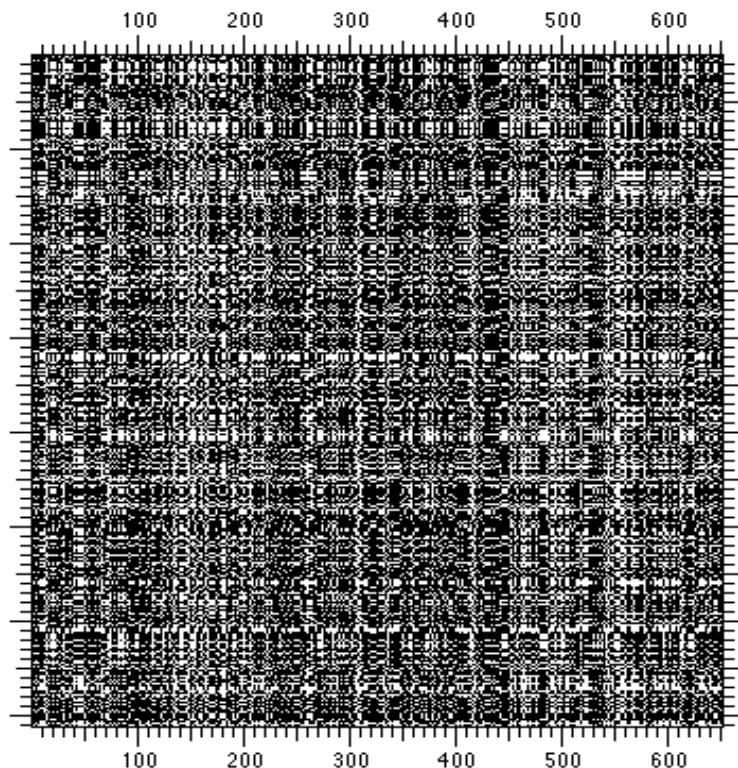


A sequence against itself... low complexity regions; repeats (may want to “mask out” prior to aligning the sequence with other sequences).

A real example: (dot-plots with tuning parameters)

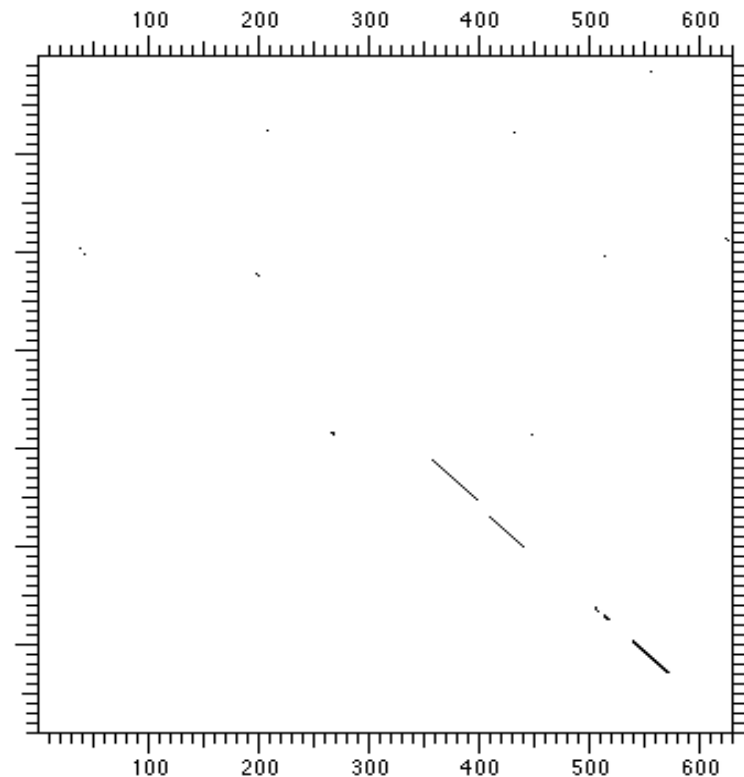
- every point corresponds to a window on the sequence, of size **W**
- a dot is drawn if there are at least **S** (stringency) matches in the window

W=1 (single position); S=1



too much detail to detect structure visually...

W=23; S=15



... appropriately tuning the parameters the alignment structure appears

Dynamic programming: Pairwise sequence alignment

Create a **matrix** table for comparing sequences with one sequence along each axis (size $m+1, n+1$)

Fill in **partial alignment scores** until score for entire sequence has been calculated:

- **Assign score for each position i,j** , progressing from top left to bottom right
 1. Take value from left, assign gap penalty (gap along left axis)
 2. Take value from top, assign gap penalty (gap along top axis)
 3. Take value from diagonal above left, assign

match/mismatch score

- **Repeat** until table is completed.

Use **trace-back** to obtain full alignment:

AC--TCG
ACAGTAG

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

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

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

Heuristic (vs. optimal) methods for pair-wise alignment: the BLAST family of algorithms

Given a *scoring system* (scoring matrix and gap penalties), alignment can be evaluated *quantitatively*, and *optimal* alignments can be sought with *Dynamic Programming* algorithms:

- **GLOBAL**: Needleman-Wunsch-Gotoh algorithm
- **LOCAL**: Smith-Waterman algorithm

These have high *algorithmic complexity ($O(N^2)$)*, and are replaced by *heuristic procedures* in most practical applications.

Most commonly used: *BLAST family of algorithms* (local alignment).

BLAST: heuristic database search using local alignment

Basic Local Alignment Search Tool

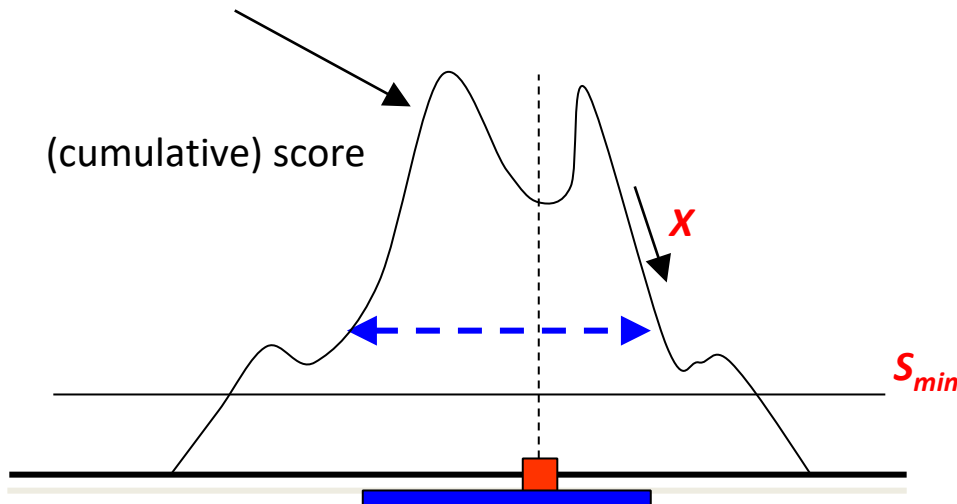
...T L S R D Q H A W R L S query sequence

QW (query word), size $W=3$.

{ (RDQ,16) (RBQ, 14) ... (REQ, 12)... (RDB, 11) }

...T L S **R D Q** H A W R L S

...R L S **R E Q** H T W R S S



A cartoon for BLAST (parameters)

For each QW , use the **scoring matrix** to form a **neighborhood** $NB = \{\text{all words of size } W \text{ with a score } \geq T=11\}$

Find match(es) to words belonging to NB in the subject (target) sequence

For each match, use the **scoring matrix** and **gap penalties** to produce an **HSP** (High scoring Segment Pair), then extend alignment on both sides, until

- drop is $> X$, or
- score goes below S_{min} (minimum hit score)

How likely is it to find a match by chance?

“Given a set of sequences NOT related to the query sequence (or even random sequences), what is the probability of finding a match with alignment score S simply by chance?”

Score will depend on:

- Length and composition of the query and target sequence
- Scoring matrix

Statistical significance is given by **probability** and **expectation values**:

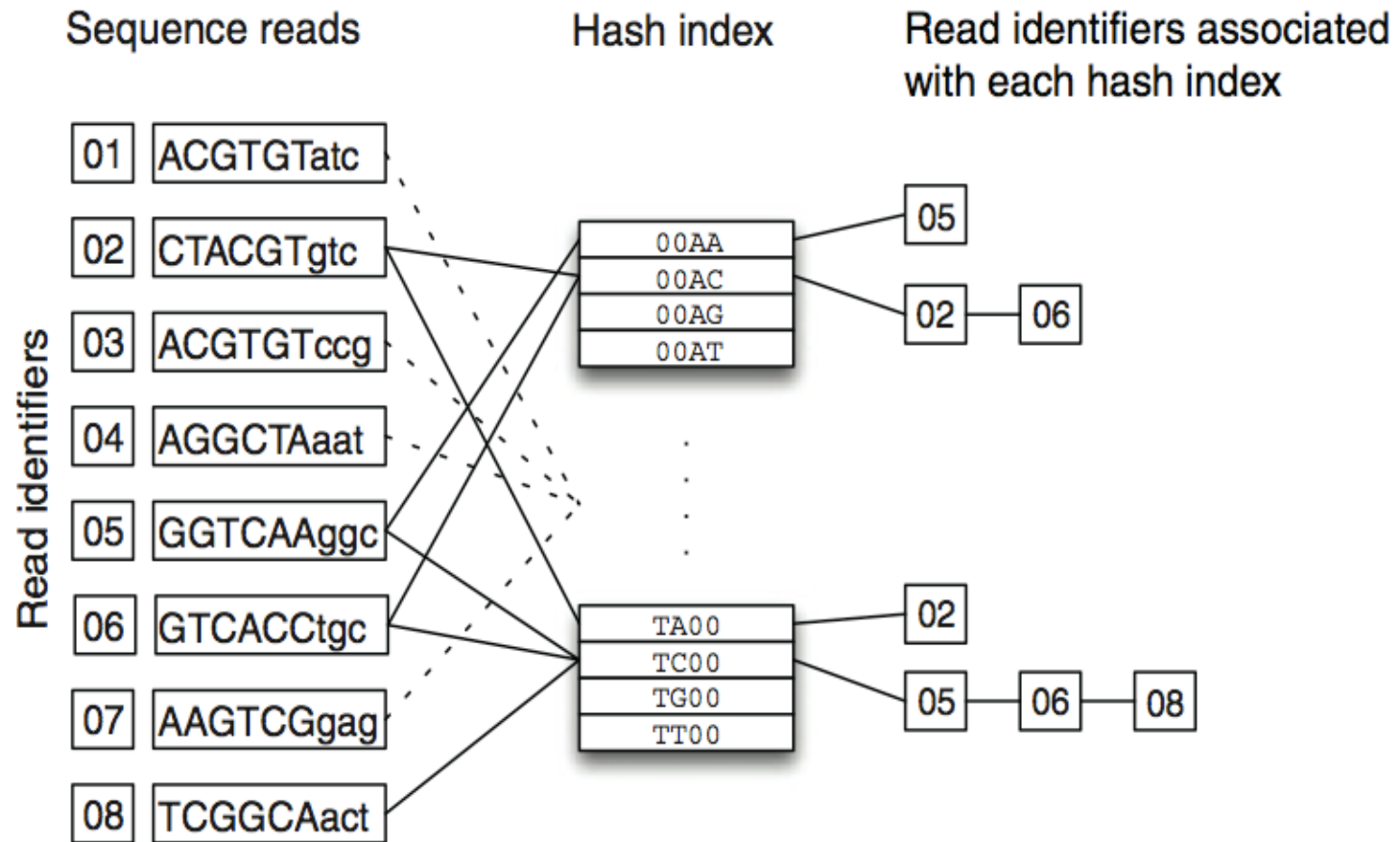
P-value: probability of finding one or more sequences of score $\geq S$
by random chance

E-value: expected number of sequences of score $\geq S$
that would be found by random chance

About Blat (from genome.ucsc.edu)

- “BLAT on DNA is designed to quickly find sequences of 95% and greater similarity of length 25 bases or more. “
- “It may miss more divergent or shorter sequence alignments. It will find perfect sequence matches of 20 bases. “
- “BLAT is not BLAST.”
- “DNA BLAT works by keeping an index of the entire genome in memory. The index consists of all overlapping 11-mers stepping by 5 except for those heavily involved in repeats.”
- “The index takes up about 2 gigabytes of RAM. The genome itself is not kept in memory, allowing BLAT to deliver high performance on a reasonably priced Linux box. “
- “The index is used to find areas of probable homology, which are then loaded into memory for a detailed alignment.”

Hash Table (BLAT)



Short Read Applications

```
...CCATAG      TATGCGCCC      CGGAAATTT      GGTATAC...
...CCAT      CTATATGCG      TCGGAAATT      CGGTATAC
...CCAT      GGCTATATG      CTATCGGAAA      CGGTATAC
...CCA      AGGCTATAT      CCTATCGGA      TCGGTATA      C...
...CCA      AGGCTATAT      GCCCTATCG      TTTGCGGT      C...
...CC      AGGCTATAT      GCCCTATCG      AAATTTGC      ATAC...
...CC      TAGGCTATA      GCGCCCTA      AAATTTGC      GTATAC...
```

...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...

```
      GAAATTTGC
      GGAAATTTG
      CGGAAATTT
      CGGAAATTT
      TCGGAAATT
      CTATCGGAAA
      CCTATCGGA      TTTGCGGT
      GCCCTATCG      AAATTTGC
      GCCCTATCG      AAATTTGC      ATAC...
```

...CC

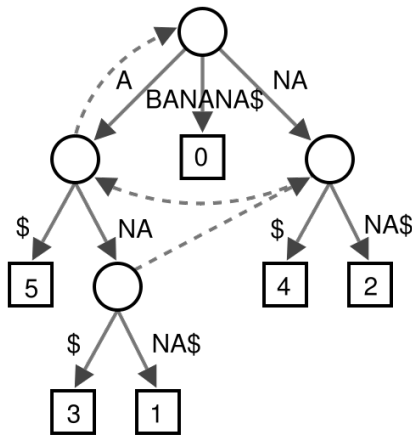
...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...

Finding the alignments is typically the performance bottleneck

- New alignment algorithms must address the requirements and characteristics of NGS reads
 - Millions of reads per run (30x of genome coverage)
 - Short Reads (as short as 36bp)
 - Different types of reads (single-end, paired-end, mate-pair, etc.)
 - Base-calling quality factors
 - Sequencing errors (~ 1%)
 - Repetitive regions
 - Sequencing organism vs. reference genome
 - Must adjust to evolving sequencing technologies and data formats

Indexing

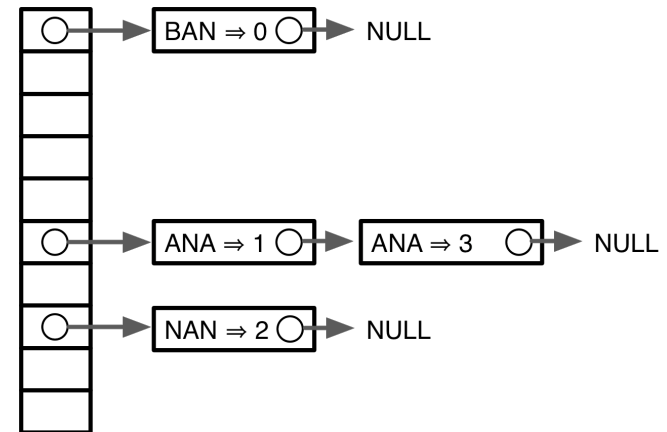
- Genomes and reads are too large for direct approaches like dynamic programming
- *Indexing* is required



Suffix tree

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

Suffix array



Seed hash tables

Many variants, incl. spaced seeds

- Choice of index is key to performance

Suffix Array

[00] gggtaaagctataactattgatcaggcggtt

Find "ctat" in the reference

Suffix Array

[00] gggtaaagctataactattgatcaggcggt
[01] ggtaaagctataactattgatcaggcggt
[02] gtaaagctataactattgatcaggcggt
[03] taaagctataactattgatcaggcggt
[04] aaagctataactattgatcaggcggt
[05] aagctataactattgatcaggcggt
[06] agctataactattgatcaggcggt
[07] gctataactattgatcaggcggt
[08] ctataactattgatcaggcggt
[09] tataactattgatcaggcggt
[10] ataactattgatcaggcggt
[11] taactattgatcaggcggt
[12] aactattgatcaggcggt
[13] actattgatcaggcggt
[14] ctattgatcaggcggt
[15] tattgatcaggcggt
[16] attgatcaggcggt
[17] ttgatcaggcggt
[18] tgatcaggcggt
[19] gatcaggcggt
[20] atcaggcggt
[21] tcaggcggt
[22] caggcggt
[23] aggcgtt
[24] ggcgtt
[25] gcgtt
[26] cggt
[27] gtt
[28] tt
[29] t

Find "ctat" in the reference

[04] aaagctataactattgatcaggcggt
[12] aactattgatcaggcggt
[05] aagctataactattgatcaggcggt
[13] actattgatcaggcggt
[06] agctataactattgatcaggcggt
[23] aggcgtt
[10] ataactattgatcaggcggt
[20] atcaggcggt
[16] attgatcaggcggt
[22] caggcggt
[26] cggt
[08] ctataactattgatcaggcggt
[14] ctattgatcaggcggt
[19] gatcaggcggt
[25] gcgtt
[07] gctataactattgatcaggcggt
[24] ggcgtt
[00] gggtaaagctataactattgatcaggcggt
[01] ggtaaagctataactattgatcaggcggt
[02] gtaaagctataactattgatcaggcggt
[27] gtt
[29] t
[03] taaagctataactattgatcaggcggt
[11] taactattgatcaggcggt
[09] tataactattgatcaggcggt
[15] tattgatcaggcggt
[21] tcaggcggt
[18] tgatcaggcggt
[28] tt
[17] ttgatcaggcggt

NGS Read Alignment

Burrows Wheeler Transformation (BWT)

- Invented by David Wheeler in 1983 (Bell Labs). Published in 1994.
“A Block Sorting Lossless Data Compression Algorithm”
Systems Research Center Technical Report No 124. Palo Alto, CA: Digital Equipment Corporation, Burrows M, Wheeler DJ. 1994
- Originally developed for compressing large files (*bzip2*, etc.)
- Lossless, Fully Reversible
- Alignment Tools based on BWT: *bowtie*, *BWA*, *SOAP2*, etc.
- Approach:
 - Align reads on the transformed reference genome, using an efficient index (FM index)
 - Solve the simple problem first (align one character) and then build on that solution to solve a slightly harder problem (two characters) etc.
- Results in great speed and efficiency gains (a few GigaByte of RAM for the entire H. Genome). Other approaches require tens of GigaBytes of memory and are much slower.

Burrows Wheeler Transformation

Text

=

c	t	g	a	a	a	c	t	g	g	t	\$
1	2	3	4	5	6	7	8	9	10	11	0

➤ Introduce **\$** at the end and construct all cyclic permutations of Text

c	t	g	a	a	a	c	t	g	g	t	\$
t	g	a	a	a	c	t	g	g	t	\$	c
g	a	a	a	c	t	g	g	t	\$	c	t
a	a	a	c	t	g	g	t	\$	c	t	g
a	a	c	t	g	g	t	\$	c	t	g	a
a	c	t	g	g	t	\$	c	t	g	a	a
c	t	g	g	t	\$	c	t	g	a	a	a
t	g	g	t	\$	c	t	g	a	a	a	c
g	g	t	\$	c	t	g	a	a	a	c	t
g	t	\$	c	t	g	a	a	a	c	t	g
t	\$	c	t	g	a	a	a	c	t	g	g
\$	c	t	g	a	a	a	c	t	g	g	t

Burrows Wheeler Transformation

- Sort rows alphabetically, keeping of which row went where

\$	c	t	g	a	a	a	c	t	g	g	t
a	a	a	c	t	g	g	t	\$	c	t	g
a	a	c	t	g	g	t	\$	c	t	g	a
a	c	t	g	g	t	\$	c	t	g	a	a
c	t	g	a	a	a	c	t	g	g	t	\$
c	t	g	g	t	\$	c	t	g	a	a	a
g	a	a	a	c	t	g	g	t	\$	c	t
g	g	t	\$	c	t	g	a	a	a	c	t
g	t	\$	c	t	g	a	a	a	c	t	g
t	\$	c	t	g	a	a	a	c	t	g	g
t	g	a	a	a	c	t	g	g	t	\$	c
t	g	g	t	\$	c	t	g	a	a	a	c

Burrows Wheeler Matrix

BWT (Text)

=

t	g	a	a	\$	a	t	t	g	g	c	c
11	3	4	5	0	6	2	8	9	10	1	7

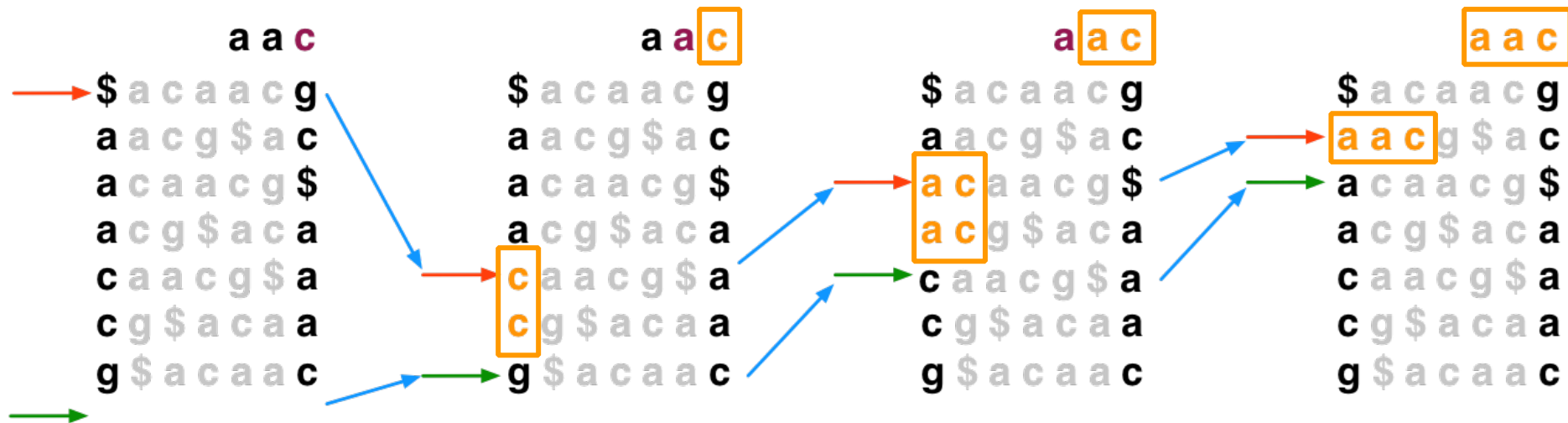
Exercise

For BWT(T) =

t	g	a	a	\$	a	t	t	g	g	c	c
11	3	4	5	0	6	2	8	9	10	1	7

Align Q = **ctgg**

Exact Matching with FM Index



- In progressive rounds, **top** & **bot** delimit the range of rows beginning with progressively longer suffixes of Q

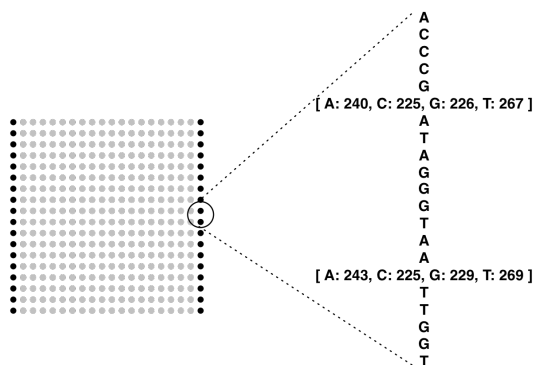
FM Index is Small

- Entire FM Index on DNA reference consists of:
 - BWT (same size as T)
 - Checkpoints (~15% size of T)
 - SA sample (~50% size of T)
- Total: ~1.65x the size of T

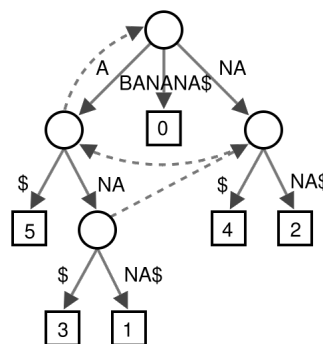
Assuming 2-bit-per-base encoding and no compression, as in Bowtie

Assuming a 16-byte checkpoint every 448 characters, as in Bowtie

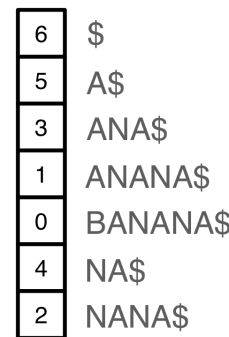
Assuming Bowtie defaults for suffix-array sampling rate, etc



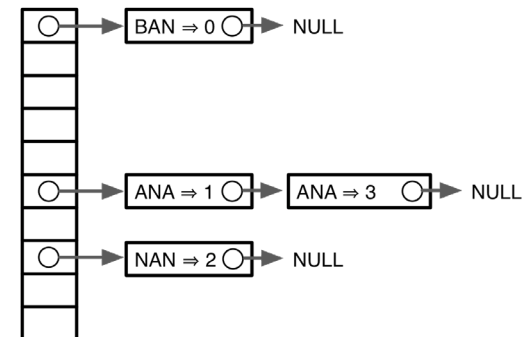
~1.65x



>45x



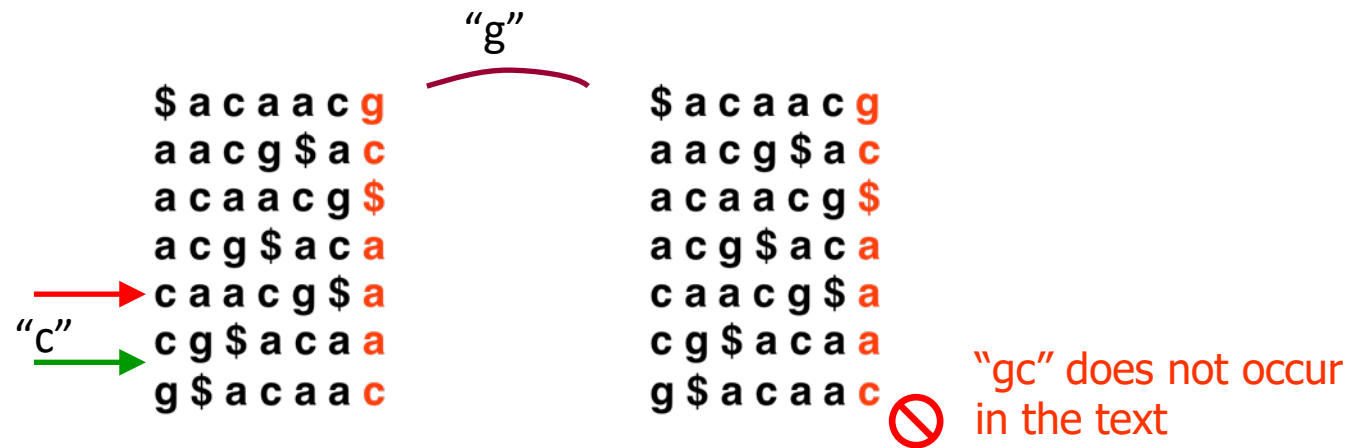
>15x



>15x

Backtracking

- Backtracking attempt for Q = "agc", T = "acaacg":



Substitution → "a"

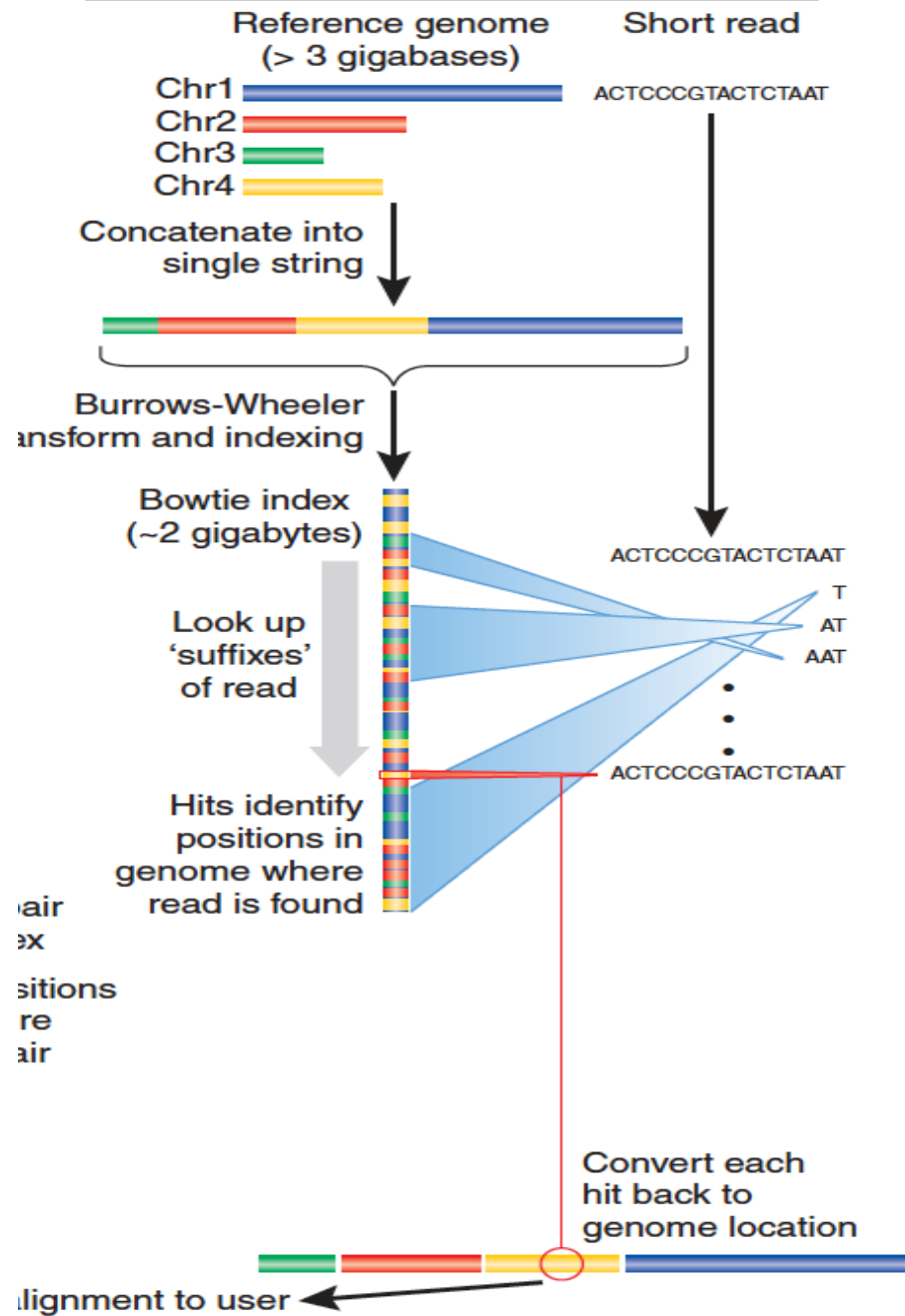


Found this alignment:

acaacg
| |
agc

b

Burrows-Wheeler



Mapping Reads from RNA molecules

- What is the advantage of mapping reads from RNA to the genome sequenced instead of a database of all predicted RNA molecules?
 - We are not depending on the quality of annotation.
 - We are not assuming that we know about all of the RNA molecules in the cell.
- How can we find reads mapping to spliced junctions?
 - Create a separate database of all possible spliced junctions
 - Split reads in half and map them separately.

Output of Aligners (SAM)

<https://learn.gencore.bio.nyu.edu/ngs-file-formats/sambam-format/>

Col	Field	Type	Regex/Range	Brief description
1	QNAME	String	[!-?A-~]{1,255}	Query template NAME
2	FLAG	Int	[0,2 ¹⁶ -1]	bitwise FLAG
3	RNAME	String	* [!-()+-<>-~] [!-~]*	Reference sequence NAME
4	POS	Int	[0,2 ²⁹ -1]	1-based leftmost mapping POSition
5	MAPQ	Int	[0,2 ⁸ -1]	MAPping Quality
6	CIGAR	String	* ([0-9]+[MIDNSHPX=])+	CIGAR string
7	RNEXT	String	* = [!-()+-<>-~] [!-~]*	Ref. name of the mate/next fragment
8	PNEXT	Int	[0,2 ²⁹ -1]	Position of the mate/next fragment
9	TLEN	Int	[-2 ²⁹ +1,2 ²⁹ -1]	observed Template LENgth
10	SEQ	String	* [A-Za-z=.]+	fragment SEQUENCE
11	QUAL	String	[!-~]+	ASCII of Phred-scaled base QUALity+33

TRIMMING: Why do we want to trim?

- Low quality sequences
- Adapter sequences
- Crop head or tail of the sequence
- Remove reads that end up to be too small
- For pair-end sequences, make sure the pairs are both in the files.

Trimmomatic

<http://www.usadellab.org/cms/?page=trimmomatic>

Paired End:

Command to execute
Trimmomatic

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

Trimmomatic

Paired End:

Telling Trimmomatic that it
is Pair-end data

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

Trimmomatic

Paired End:

Specify the quality scores

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

Trimmomatic

Paired End:

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

The input sequences,
each file represents one
member of the pair.

Trimmomatic

Paired End:

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

If both members of the pair survive the filtering then put them here.

Trimmomatic

Paired End:

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

Reads that survive filtering
but the mate did not.

Trimmomatic

Paired End:

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

Trimmomatic

Paired End:

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

Trimmomatic

Paired End:

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)

Trimmomatic

Paired End:

```
java -jar trimmomatic-0.35.jar PE -phred33 input_forward.fq.gz  
input_reverse.fq.gz output_forward_paired.fq.gz  
output_forward_unpaired.fq.gz output_reverse_paired.fq.gz  
output_reverse_unpaired.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

This will perform the following:

- Remove adapters (ILLUMINACLIP:TruSeq3-PE.fa:2:30:10)
- Remove leading low quality or N bases (below quality 3) (LEADING:3)
- Remove trailing low quality or N bases (below quality 3) (TRAILING:3)
- Scan the read with a 4-base wide sliding window, cutting when the average quality per base drops below 15 (SLIDINGWINDOW:4:15)
- Drop reads below the 36 bases long (MINLEN:36)