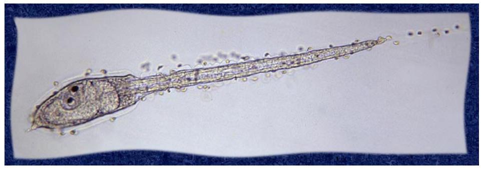
Human Genome Resequencing

Which human did we sequence?

Answer one:

Answer two: "it doesn't matter"

Polymorphism rate: number of letter changes between two different members of a species

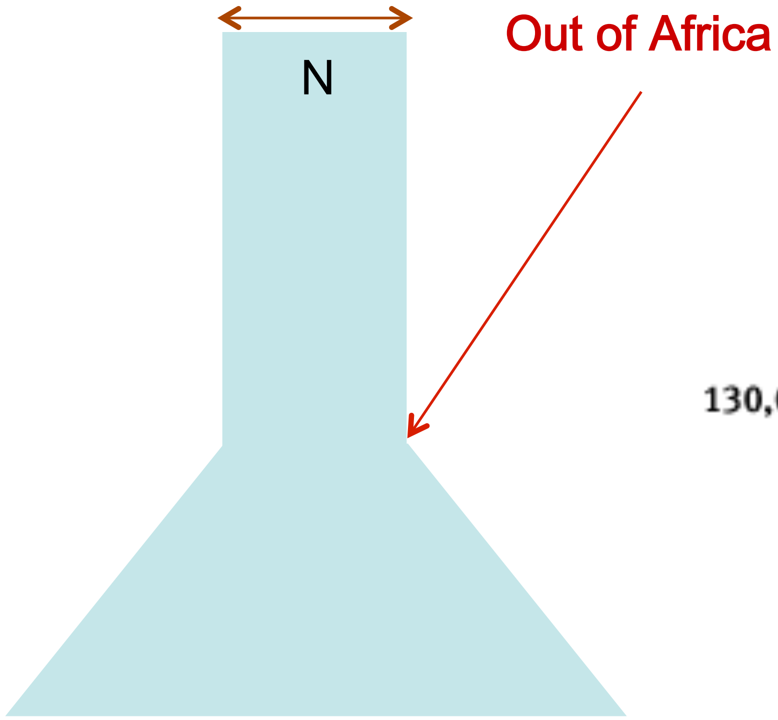


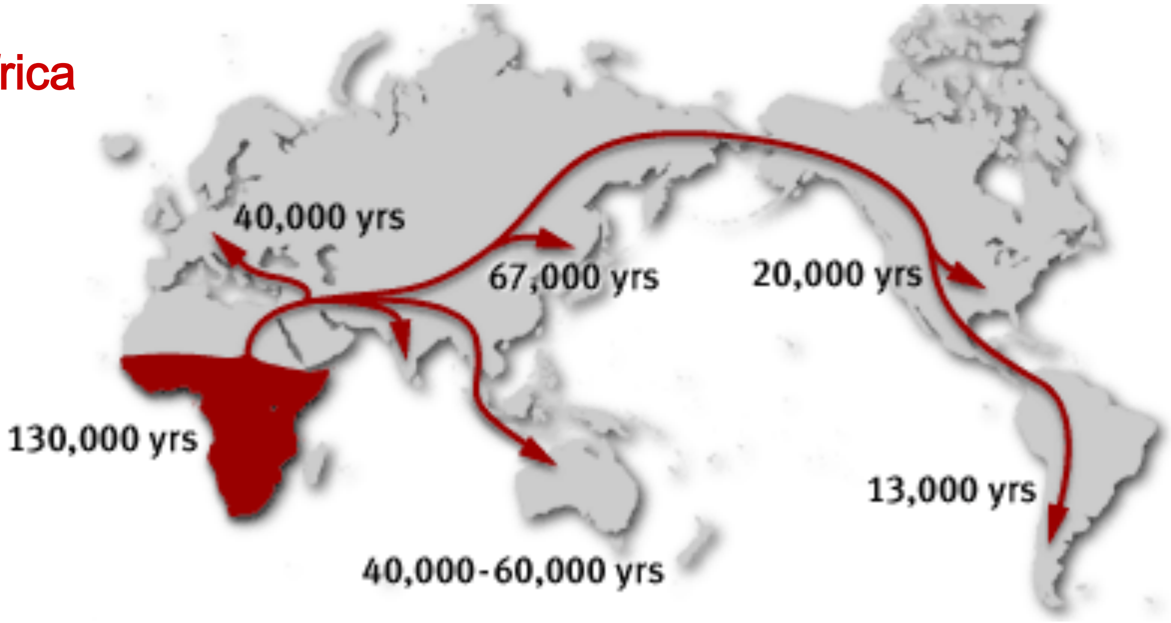
Humans: ~1/1,000

Other organisms have much higher polymorphism rates

- Population size!

Why humans are so similar





Heterozygosity: H

$H = 4Nu/(1 + 4Nu)$

$u \sim 10^{-8}$, $N \sim 10^{4}$

$\Rightarrow H \sim 4\times10^{-4}$

A small population that interbred reduced the genetic variation

Out of Africa ~ 40,000 years ago

DNA Sequencing

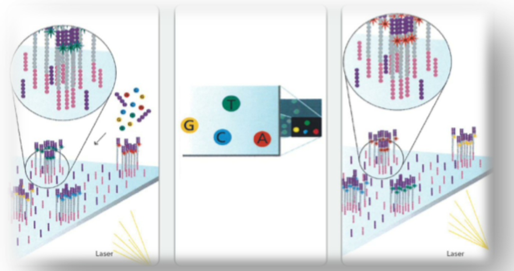
Goal:

Find the complete sequence of A, C, G, T's in DNA

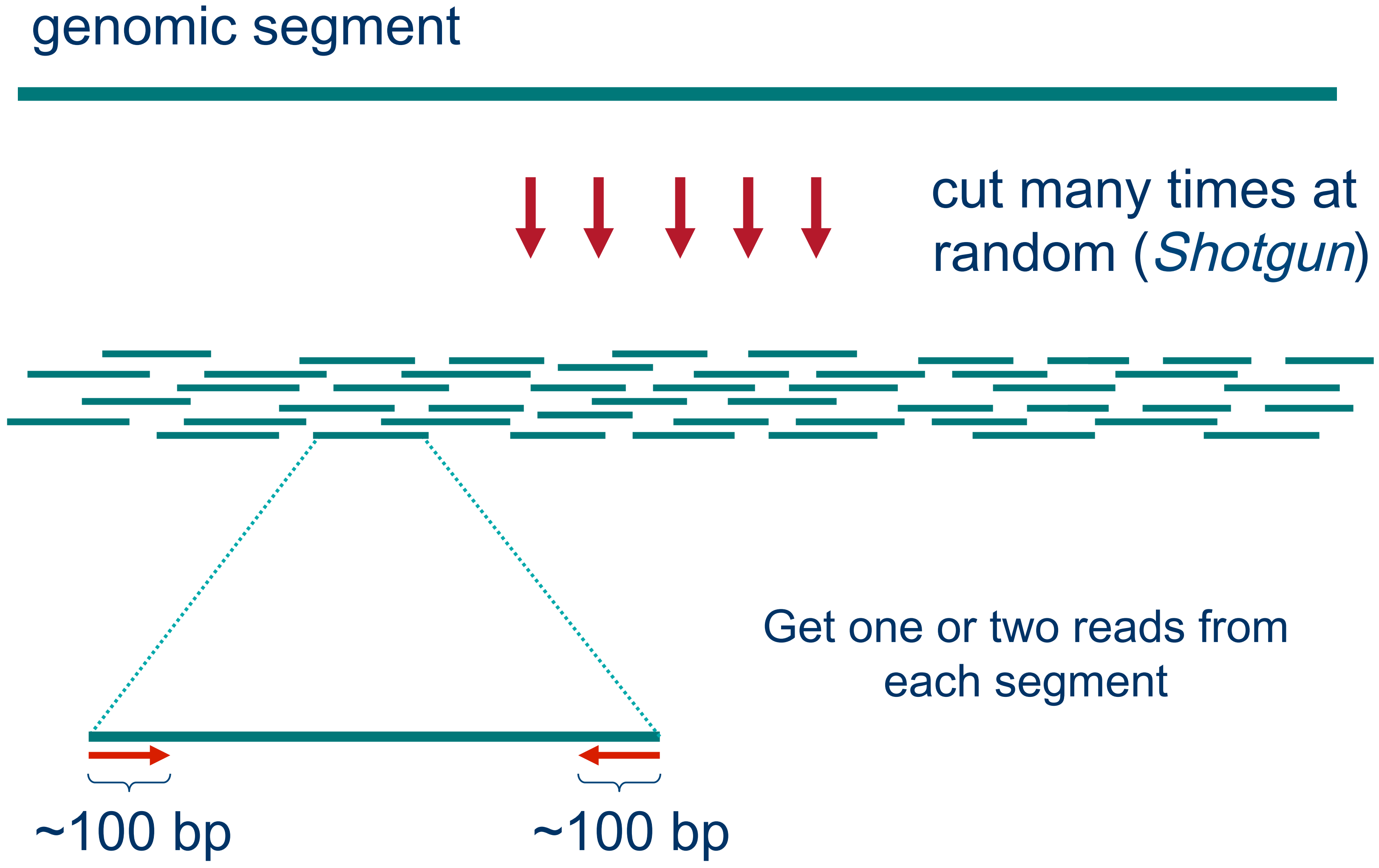
Challenge:

There is no machine that takes long DNA as an input, and gives the complete sequence as output

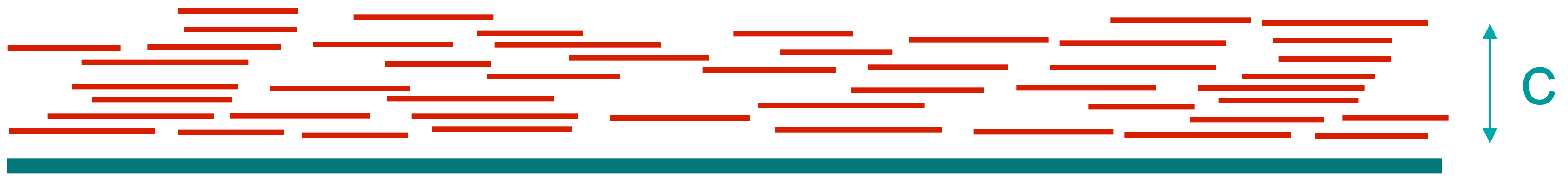
Can only sequence ~150 letters at a time



Method to sequence longer regions


genomic segment
cut many times at random (*Shotgun*)
Get one or two reads from each segment
~100 bp
~100 bp

Definition of Coverage



Length of genomic segment:	**G**
Number of reads:	**N**
Length of each read:	**L**

Definition: Coverage **C = N L / G**

How much coverage is enough?

Lander-Waterman model: Prob[not covered bp] = e^{-C}

Assuming uniform distribution of reads, C=10 results in 1 gapped region /1,000,000 nucleotides

Two main assembly problems

- De Novo Assembly

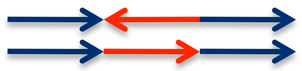
- Resequencing

Human Genome Variation

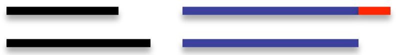
SNP

TGCTGAGA
TGCCGAGA

Inversion



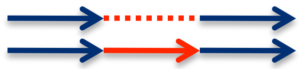
Translocation



Microdeletion

TGC - - AGA
TGCCGAGA

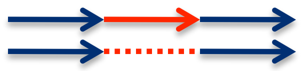
Large Deletion



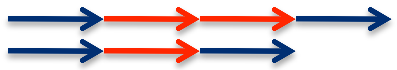
Novel Sequence

TGCTCGGAGA
TGC - - - GAGA

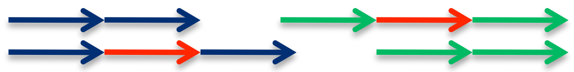
Mobile Element or Pseudogene Insertion



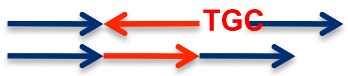
Tandem Duplication



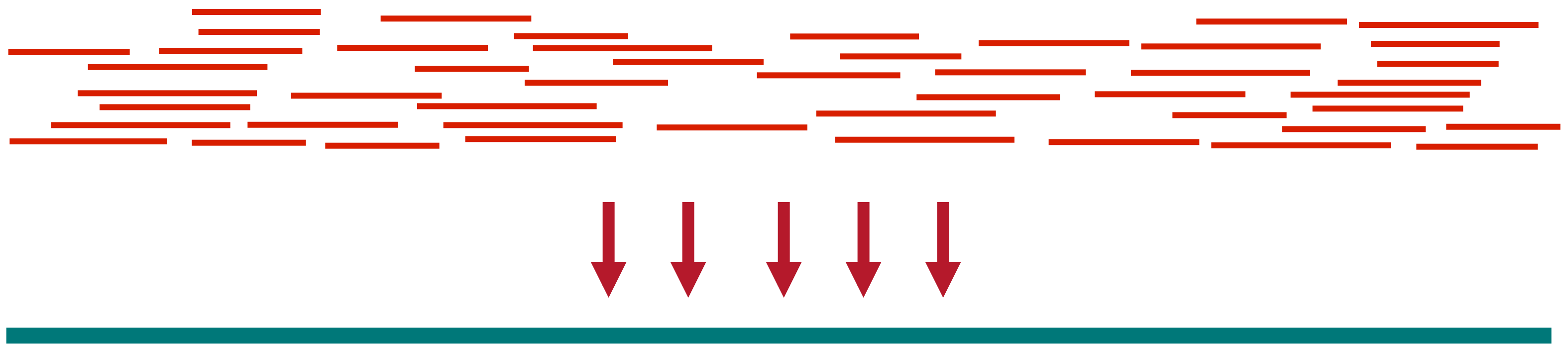
Transposition



Novel Sequence at Breakpoint



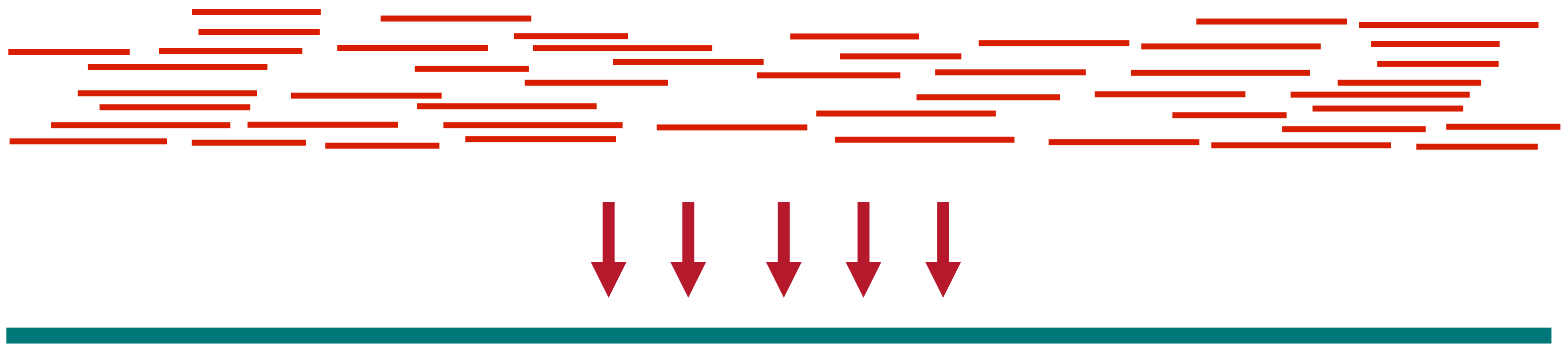
Read Mapping



```
                  CATCGACCGAGCGCGATGCTAGCTAGGTGATCGT......
             TGCCGCATCGACCGAGCGCGATGCTAGCTAGGTGATCGT...
          GCATGCCGCATCGACCGAGCGCGATGCTAGCTAGGTGATCGT
        GTGCATGCCGCATCGACCGAGCGCGATGCTAGCTAGGTGATC
......AGGTGCATGCCGCATCGATCGAGCGCGATGCTAGCTAGCTGATCGT......
```

- Want ultra fast, highly similar alignment
- Detection of genomic variation

Read Mapping – Burrows-Wheeler Transform



```
                   CATCGACCGAGCGCGATGCTAGCTAGGTGATCGT......
              TGCCGCATCGACCGAGCGCGATGCTAGCTAGGTGATCGT...
           GCATGCCGCATCGACCGAGCGCGATGCTAGCTAGGTGATCGT
         GTGCATGCCGCATCGACCGAGCGCGATGCTAGCTAGGTGATC
.......AGGTGCATGCCGCATCGATCGAGCGCGATGCTAGCTAGCTGATCGT......
```

- Modern fast read aligners: BWT, Bowtie, SOAP
  - Based on ***Burrows-Wheeler transform***

Burrows-Wheeler Transform

ANA

$\downarrow$

X = **BANANA**

BANANA
ANANA
NANA
ANA
NA
A

suffixes of BANANA

BANANA
ANANA
NANA
ANA
NA
A

Burrows-Wheeler Transform

ANA

↓

X = **BANANA$**

BANANA$	BANANA$
ANANA$	**ANA**NA$
NANA$	NANA$
ANA$	**ANA**$
NA$	NA$
A$	A$
$	$

Burrows-Wheeler Transform

ANA

$\downarrow$

X = BANANA$

BANANA$	BANANA$	BANANA$
ANANA$B	**ANA**NA$B	ANANA$B
NANA$BA	NANA$BA	NANA$BA
ANA$BAN	**ANA**$BAN	ANA$BAN
NA$BANA	NA$BANA	NA$BANA
A$BANAN	A$BANAN	A$BANAN
$BANANA	$BANANA	$BANANA

Burrows-Wheeler Transform

ANA

$\downarrow$

X = BANANA$

BANANA$	BANANA$	$BANANA
ANANA$B	ANANA$B	A$BANAN
NANA$BA	NANA$BA	ANA$BAN
ANA$BAN	ANA$BAN	ANANA$B
NA$BANA	NA$BANA	BANANA$
A$BANAN	A$BANAN	NA$BANA
$BANANA	$BANANA	NANA$BA

Burrows-Wheeler Transform

	ANA	BANANA$	BANANA$	$BANANA
		ANANA$B	ANANA$B	A$BANAN
	$\downarrow$	NANA$BA	NANA$BA	ANA$BAN
		ANA$BAN	ANA$BAN	ANANA$B
		NA$BANA	NA$BANA	BANANA$
X =	BANANA$	A$BANAN	A$BANAN	NA$BANA
		$BANANA	$BANANA	NANA$BA

Burrows-Wheeler Transform

ANA

$\downarrow$

X = BANANA$

```
BANANA$
ANANA$B
NANA$BA
ANA$BAN
NA$BANA
A$BANAN
```

```
BANANA$
ANANA$B
NANA$BA
ANA$BAN
NA$BANA
A$BANAN
```

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string ‘BANANA’

BWT(BANANA) = ANNB$AA

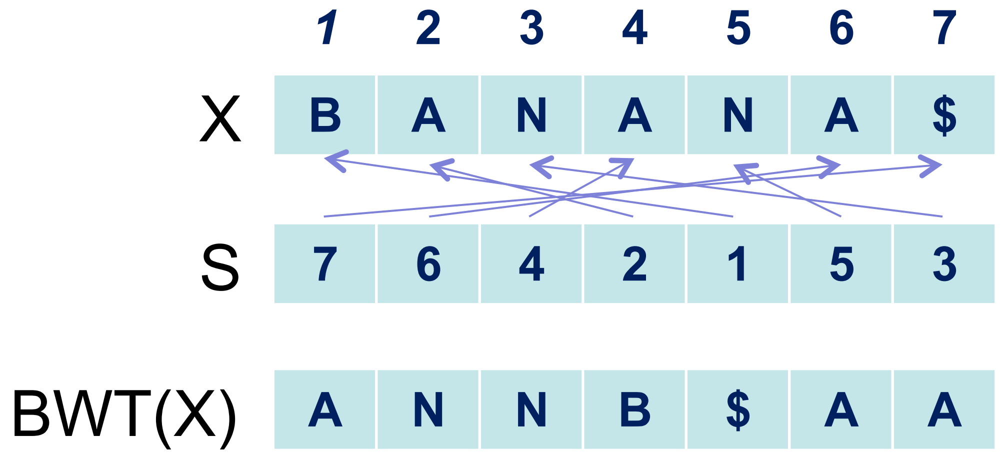
Suffix Arrays

$BANANA	1	$BANANA
A$BANAN	2	A$BANAN
ANA$BAN	3	ANA$BAN
ANANA$B	4	ANANA$B
BANANA$	5	BANANA$
NA$BANA	6	NA$BANA
NANA$BA	7	NANA$BA

Suffixes are sorted in the BWT matrix

<u>Define suffix array S:</u>

S(i) = j, where $X_j \ldots X_n$ is the i-th suffix lexicographically



	1	2	3	4	5	6	7
X	B	A	N	A	N	A	$
S	7	6	4	2	1	5	3
BWT(X)	A	N	N	B	$	A	A

BWT(X) constructed from S:
At each position, take the letter to the left of the one pointed by S

Reconstructing BANANA

$BANANA	A	$	A$	$B	A$B	$BA
A$BANAN	N	A	NA	A$	NA$	A$B
ANA$BAN	N	A	NA	AN	NAN	ANA
ANANA$B	B	A	BA	AN	BAN	ANA
BANANA$	$	B	$B	BA	$BA	BAN
NA$BANA	A	N	AN	NA	ANA	NA$
NANA$BA	A	N	AN	NA	ANA	NAN

BWT matrix of string 'BANANA'

$\rightarrow$ sort $\rightarrow$ append BWT $\rightarrow$ sort $\rightarrow$ append BWT $\rightarrow$ sort

Reconstructing BANANA - faster

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string 'BANANA'

Lemma. The i-th occurrence of character c in last column is the same text character as the i-th occurrence of c in the first column

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

Reconstructing BANANA - faster

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string 'BANANA'

Lemma. The i-th occurrence of character c in last column is the same text character as the i-th occurrence of c in the first column

```
A $BANAN
N A$BANA
N ANA$BA
B ANANA$
$ BANANA
A NA$BAN
A NANA$B
```

Reconstructing BANANA - faster

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string 'BANANA'

Lemma. The i-th occurrence of character c in last column is the same text character as the i-th occurrence of c in the first column

```
A $BANAN
N A$BANA
N ANA$BA
B ANANA$
$ BANANA
A NA$BAN
A NANA$B
```

```
A$BANAN
ANA$BAN
ANANA$B
```

Same words, same sorted order

Reconstructing BANANA - faster

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string ‘BANANA’

Lemma. The i-th occurrence of character ‘a’ in last column is the same text character as the i-th occurrence of ‘a’ in the first column

LF(): Map the i-th occurrence of character ‘a’ in last column to the first column

LF(r): Let row r contain the i-th occurrence of ‘a’ in last column

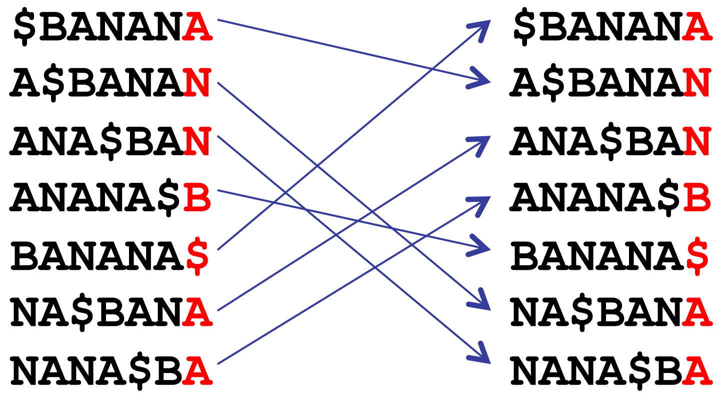
Then, LF(r) = r’; r’: i-th row starting with ‘a’

Reconstructing BANANA - faster

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string 'BANANA'

LF(r): Let row r be the i-th occurrence of 'a' in last column
Then, LF(r) = r'; r': i-th row starting with 'a'



LF[] = [2, 6, 7, 5, 1, 3, 4]

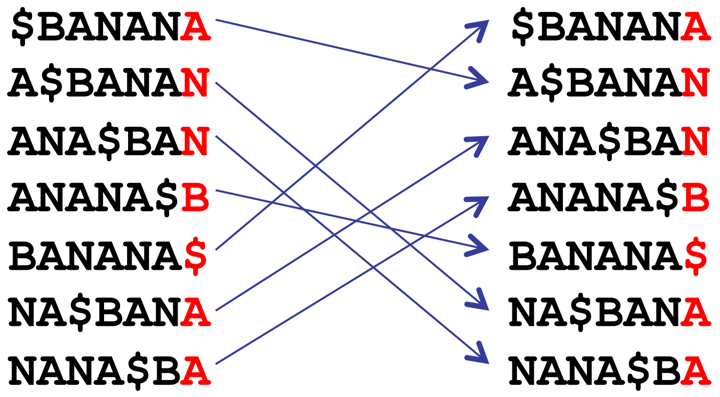
Row LF(r) is obtained by rotating row r one position to the right

Reconstructing BANANA - faster

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string ‘BANANA’

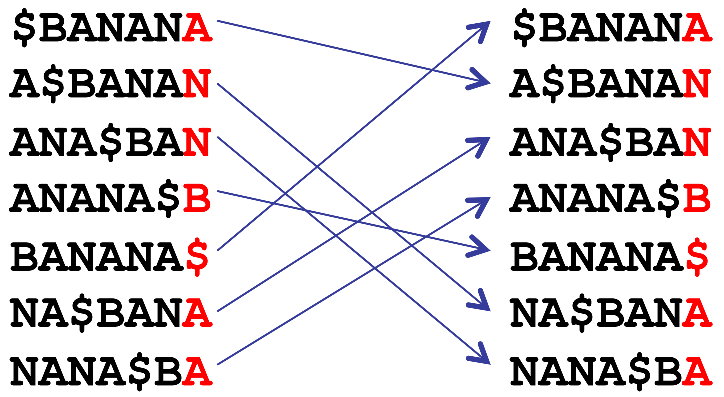
LF(r): Let row r be the i-th occurrence of ‘a’ in last column
Then, LF(r) = r’; r’: i-th row starting with ‘a’



LF[] = [2, 6, 7, 5, 1, 3, 4]

Therefore, the last character in row LF(r) is the character before the last character in row r

Reconstructing BANANA - faster



```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string ‘BANANA’

LF[] = [2, 6, 7, 5, 1, 3, 4]

Computing LF() is easy:

Let C(a): # of characters smaller than ‘a’

Example: C($) = 0; C(A) = 1; C(B) = 4; C(N) = 5

Let row r end with the i-th occurrence of ‘a’ in last column

Then, LF(r) = C(a) + i (why?)

Reconstructing BANANA - faster

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string 'BANANA'

	A	N	N	B	$	A	A	
C()	1	5	5	4	0	1	1	C() copied for convenience
index i	1	1	2	1	1	2	3	indicating this is i-th occurrence of 'c'
LF()	2	6	7	5	1	3	4	LF() = C() + i

Reconstruct BANANA:

```
S := ""; r := 1; c := BWT[r];
UNTIL c = '$' {
        S := cS;
        r := LF(r);
        c := BWT(r); }
```

Credit: Ben Langmead thesis

Searching for query “ANA”

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string ‘BANANA’

L(W): lowest index in BWT matrix where W is prefix
U(W): highest index in BWT matrix where W is prefix

Example:
L(“NA”) = 6
U(“NA”) = 7

<u>Lemma (prove as exercise)</u>
L(aW) = C(a) + i +1,
where i = # ‘a’s up to L(W) – 1 in BWT(X)
U(aW) = C(a) + j,
where j = # ‘a’s up to U(W) in BWT(X)

Example:
L(“ANA”) = C(‘A’) + # ‘A’s up to (L(“NA”) – 1) + 1
= 1 + (# ‘A’s up to 5) + 1
= 1 + 1 + 1 = 3
U(“ANA”) = 1 + # ‘A’s up to U(“NA”) = 1 + 3 = 4

Searching for query "ANA"

```
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA
```

BWT matrix of string 'BANANA'

```
Let
LFC(r, a) = C(a) + i, where i = #'a's up to r in BWT

ExactMatch(W[1…k]) {

    a := W[k];
    low := C(a) +1;
    high := C(a+1);    // a+1: lexicographically next char
    i := k – 1;
    while (low <= high && i >= 1) {
            a = W[i];
            low = LFC(low – 1, a) + 1;
            high = LFC(high, a);
            i := i – 1; }
    return (low, high);
}
```

Credit: Ben Langmead thesis

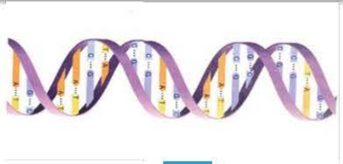
Summary of BWT algorithm

Suffix array of string X:

S(i) = j, where $X_j \ldots X_n$ is the j-th suffix lexicographically

- BWT follows immediately from suffix array
  - Suffix array construction possible in O(n), many good O(n log n) algorithms
- Reconstruct X from BWT(X) in time O(n)
- Search for all exact occurrences of W in time O(|W|)
- BWT(X) is easier to compress than X

BWT Index Construction



Reference Sequence Construction

BWT Construction

BWT-auxiliary Structure Construction (C & O arrays) and Compression

.bwt

Reference

TTATTT...ATGTGCCTTTGAAA...GTTAAACCT...AAATT.....AATTT...AGGTTTAAAC..TTTCAAAGGCACAT..AAATAA$

Forward *Reverse Complement*

BWT

ACGTTA..TTCTGAATGTGACC…TCCAGACGA…CCATT.....AGTTC…CGGATT AGAT…AAGTACCGTGTGAT…CCAGAT

Compressed BWT (4 bases/byte)

TTATTT...ATGTGCCTT............$GTTGGTTAATAA

C-array

$:	0
T:	55000
C:	1044814
G:	7814189
A:	1

O-array

	T	G	C	A
0:	1	0	0	0
1:	2	0	0	0
2:	....			
3:	....			
..	....			
|G|-1:	....			

SA

0:	|G|-1
1:	64
2:	144814
3:	781414689
..:	…
|G|-1:	1484

Memory Consumption

For a genome of length n:

-- occurrence array O(.,.) needs 4nlogn bits

→ sampling: store only O(.,k) for e.g. k = 128

→ use BWT to compute missing counts

-- suffix array SA(.) needs nlogn bits

→ sampling: store SA(k) for e.g. k = 32

→ use inverse compressed suffix array

BWA Inexact Matching

Allow up to *n* mismatches/gaps.

Backwards-search extension:
Given read W, keep track of multiple possible partial alignments of W

Partial alignment 4-tuple: (i, z, L , U)

```
$I \leftarrow \emptyset$
$I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z-1, k, l)$
**for** each $b \in \{A, C, G, T\}$ **do**
    $k \leftarrow C(b) + O(b, k-1) + 1$
    $l \leftarrow C(b) + O(b, l)$
    **if** $k \leq l$ **then**
        $I \leftarrow I \cup \textsc{InexRecur}(W, i, z-1, k, l)$
        **if** $b = W[i]$ **then**
            $I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z, k, l)$
        **else**
            $I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z-1, k, l)$
```

BWA Inexact Matching

```
$I \leftarrow \emptyset$
$I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z-1, k, l)$
for each $b \in \{A, C, G, T\}$ do
    $k \leftarrow C(b) + O(b, k-1) + 1$
    $l \leftarrow C(b) + O(b, l)$
    if $k \leq l$ then
        $I \leftarrow I \cup \textsc{InexRecur}(W, i, z-1, k, l)$
        if $b = W[i]$ then
            $I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z, k, l)$
        else
            $I \leftarrow I \cup \textsc{InexRecur}(W, i-1, z-1, k, l)$
```

W = ACTGTGT

Partial alignment 4-tuple: (i = 4, z = 3, L , U)

Recursive step:

A	C	T	G	gap-ref	gap-read
AGT	CGT	TGT	GGT	~~T~~GT	*GT
z-1	z-1	z	z-1	z-1	z-1
i-1	i-1	i-1	i-1	i-1	i
L^AU^A	L^CU^C	L^TU^T	L^GU^G	LU	L^AU^A L^CU^C L^TU^T L^GU^G
…GAGT	…GCGT	…GTGT	…GGGT	…G-GT	…GT[A/C/T/G]GT
…GTGT	…GTGT	…GTGT	…GTGT	…GTGT	…GT - GT

L^A=C(A) + O(A, L-1) + 1

U^A=C(A) + O(A, L)

BWA Heuristics

- Lower bound array *D*, where *D(i)* := LB on number of differences of exactly matching R[0,i] with the reference (can be computed in O(|R|) time → check $n < D(i)$ instead of $n < 0$)
- Process best partial alignments first: use a *min*-priority heap to store alignment entries (instead of recursion)
- Prune out alignments considered sub-optimal (although they might have fewer than *n* differences):

  dynamically adjust search parameters (e.g. *n*):

  (1) stop if # top hits exceeds a threshold (=30),

  (2) set $n = nbest + 1$, where *nbest* is the # of differences in top hit
- Seeding: limit the number of differences in the *seed* sequence (first *k* bp)
- Disallow indels at the ends of the read

Li H, Durbin R.
Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics, 2009. 7154 cites

Langmead B, Salzberg SL.
Fast gapped-read alignment with Bowtie2. Nature Methods, 2012. 3017 cites

Li H
Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM