CS 68: BIOINFORMATICS

Prof. Sara Mathieson
Swarthmore College
Spring 2018

Outline: Feb 12

- Feedback from survey on Friday
- Python style and pair programming recap
- Continue: BWT and application to read mapping

Notes:

- Office hours today: 3-5pm
- Fill in partner form for Lab 4

Feedback from survey

Survey feedback: understand vs. confusing

- Understand well:
  - *DBGs*
  - *overlap graphs*
  - *Eulerian walks*
  - *dynamic programming*
  - *global/local alignment*
  - *BWT process*

- Most confusing:
  - *BWT theory and read mapping*
  - *global vs. local alignment (many people)*
  - *dynamic programming*
  - *runtimes*
  - *Velvet & DBG failures/issues*
  - *graphing (?)*

Survey feedback: in-class preferences

- Slides less I more IIIIIIIII
- Board less IIII more IIIIII
- Group work less I more IIII
- Handouts less more IIIIIIIIII
- Other: implementation discussion

Survey feedback: other

- **Office hours**: you can always make an appointment if you can't make office hours (I am doing research off campus on Tuesdays)
- **Open door policy**: yes! You are welcome to come in if my door is open. If closed: on a deadline or skype research meeting
- **Labs**: command line input preferred over interactive
- **Piazza**: more student answers and discussion would be great
- **Lecture recording**:... maybe a future semester!
- **Slides before class**:... not usually!

Notes about pair programming
+
Python style

Pair programming

- Make font sizes A LOT larger (editor and terminal)
- Face machine toward both partners
- Work together on the assignment as much as possible
- Try to attend office hours together if possible (or make an appointment)
- Share lab feedback with your partner

Python style

- More comments
- More line breaks
- More function/method headers

```
"""
Run the central dogma on an input DNA sequence. Allow the user
to convert between DNA, RNA, and protein sequences.
Author: Sara Mathieson
Date: 1/22/18
"""
```

```
START_CODON = 'ATG'
STOP_CODONS = ['TAG', 'TGA', 'TAA']
```

```
def read_fasta(filename):
    """Read a fasta file to produce a single strand of DNA.
    Parameters: input filename (string)
    Return: DNA object
    """
    fasta_file = open(filename,'r')
    fasta_file.readline() # header
    strand = ""
    for line in fasta_file:
        strand += line.strip()
    fasta_file.close()
    return DNA(strand)
```

- Avoid while True
- Avoid global variables
- Avoid long lines
- Avoid long main

Big picture – people asked:

- How do people come up with these algorithms?

- Where is this all going?

Big picture – people asked:

- How do people come up with these algorithms?
  - *Understand and re-implement existing algorithms from the literature*
  - *Apply to many different problems and edge cases*
  - *Find areas of improvement (runtime, accuracy, ease of use, etc)*

- Where is this all going?

Big picture – people asked:

- How do people come up with these algorithms?
  - *Understand and re-implement existing algorithms from the literature*
  - *Apply to many different problems and edge cases*
  - *Find areas of improvement (runtime, accuracy, ease of use, etc)*

- Where is this all going?

First part of the semester
(up through week 4):
Algorithms for “getting the data”

Beginning next week:
Algorithms for “learning from the data”

Big picture – people asked:

- How do people come up with these algorithms?
  - *Understand and re-implement existing algorithms from the literature*
  - *Apply to many different problems and edge cases*
  - *Find areas of improvement (runtime, accuracy, ease of use, etc)*

- Where is this all going?

- DNA sequencing
- Genome assembly
- Sequence alignment
- Read mapping
- Variant calling

First part of the semester
(up through week 4):
Algorithms for "getting the data"

Beginning next week:
Algorithms for "learning from the data"

Big picture – people asked:

- How do people come up with these algorithms?
  - *Understand and re-implement existing algorithms from the literature*
  - *Apply to many different problems and edge cases*
  - *Find areas of improvement (runtime, accuracy, ease of use, etc)*

- Where is this all going?

First part of the semester
(up through week 4):
Algorithms for "getting the data"

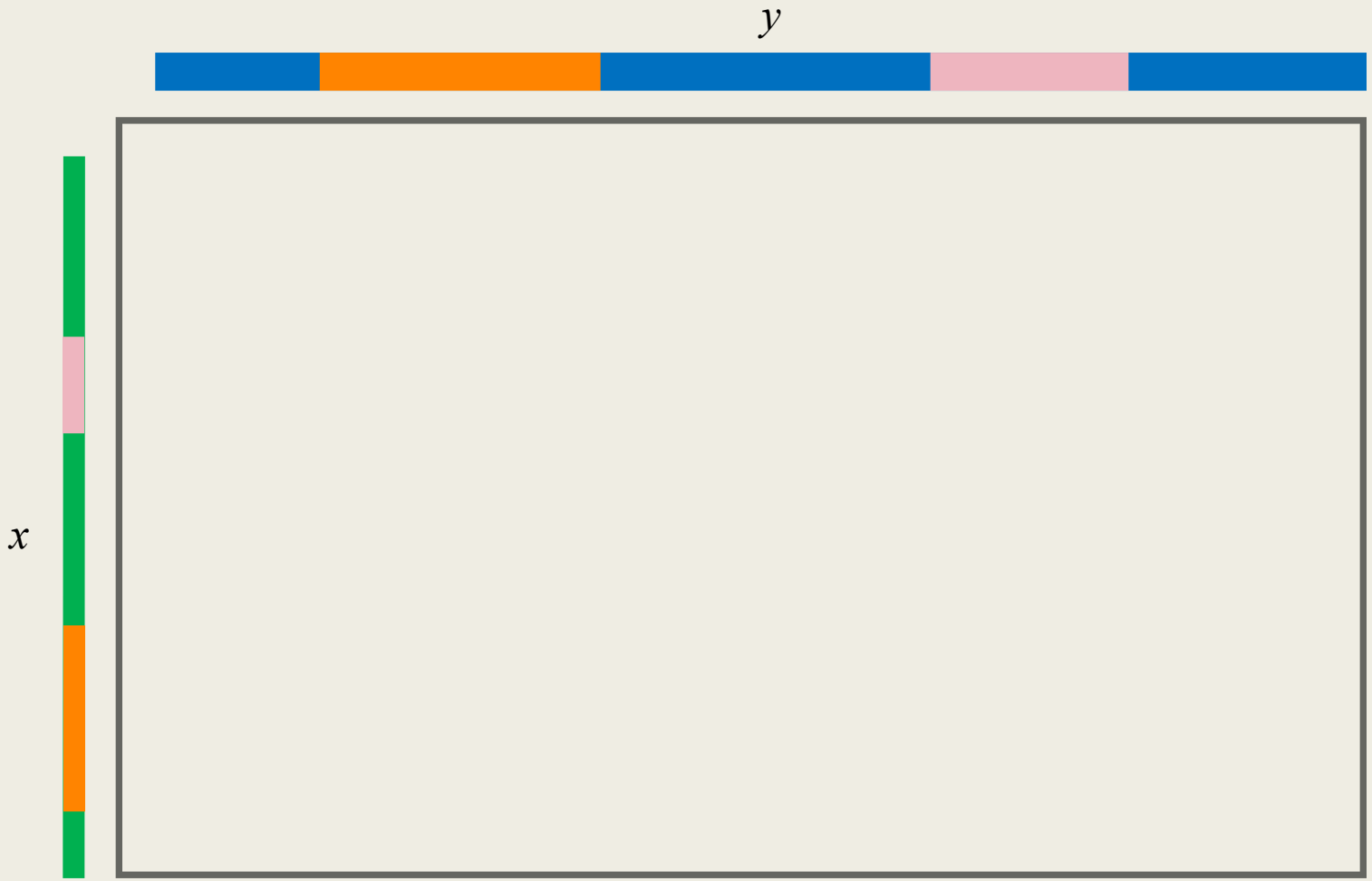
- DNA sequencing
- Genome assembly
- Sequence alignment
- Read mapping
- Variant calling

Beginning next week:
Algorithms for "learning from the data"

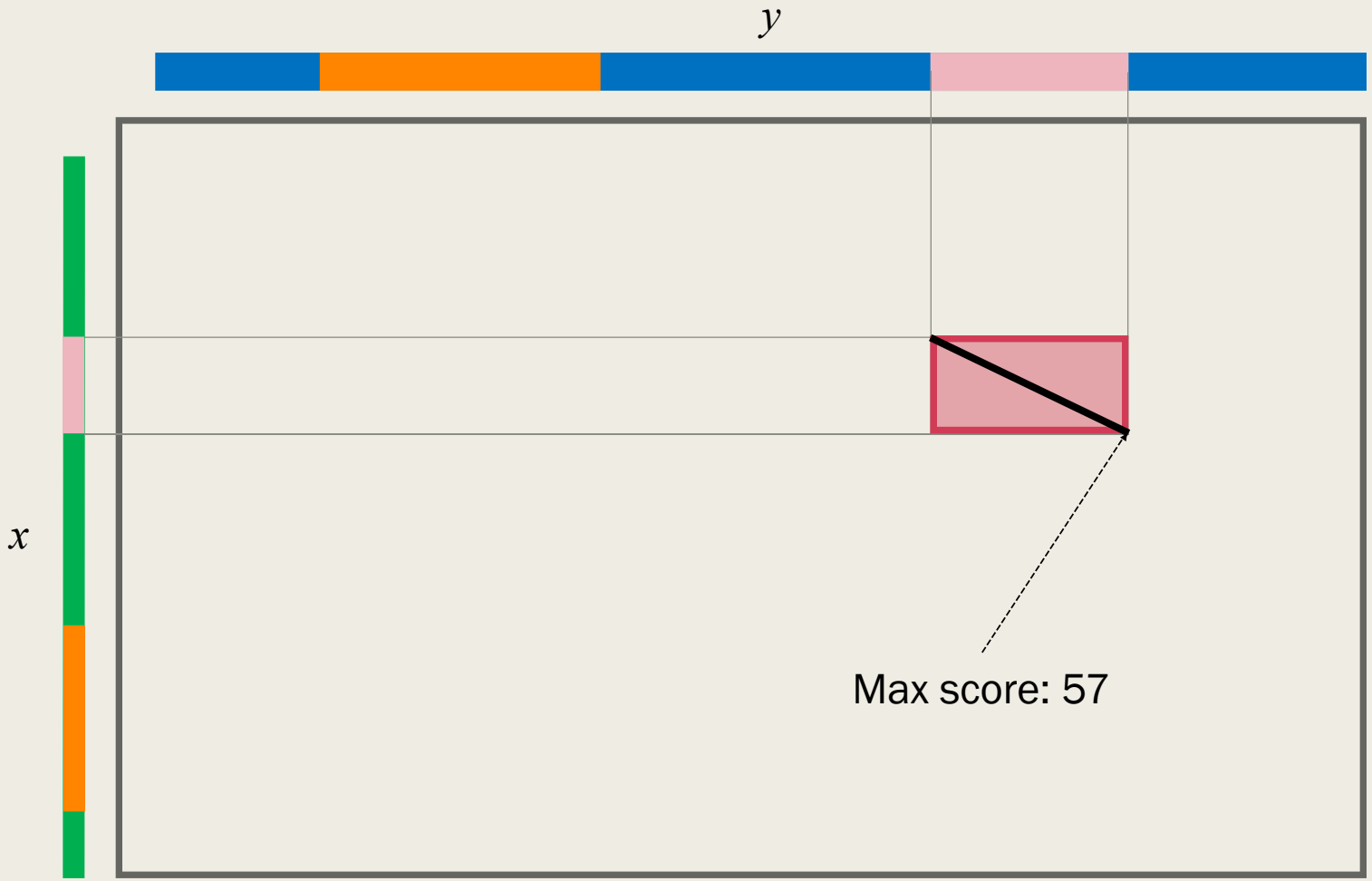
- Phylogenetic trees (speciation)
- Ancestral genome reconstruction
- Evolutionary history (natural selection, migration, admixture)
- Genomic diversity visualization and interpretation
- Cancer genetics
- Disease-gene mapping
- Pre-natal genetic testing

Global vs. Local Alignment

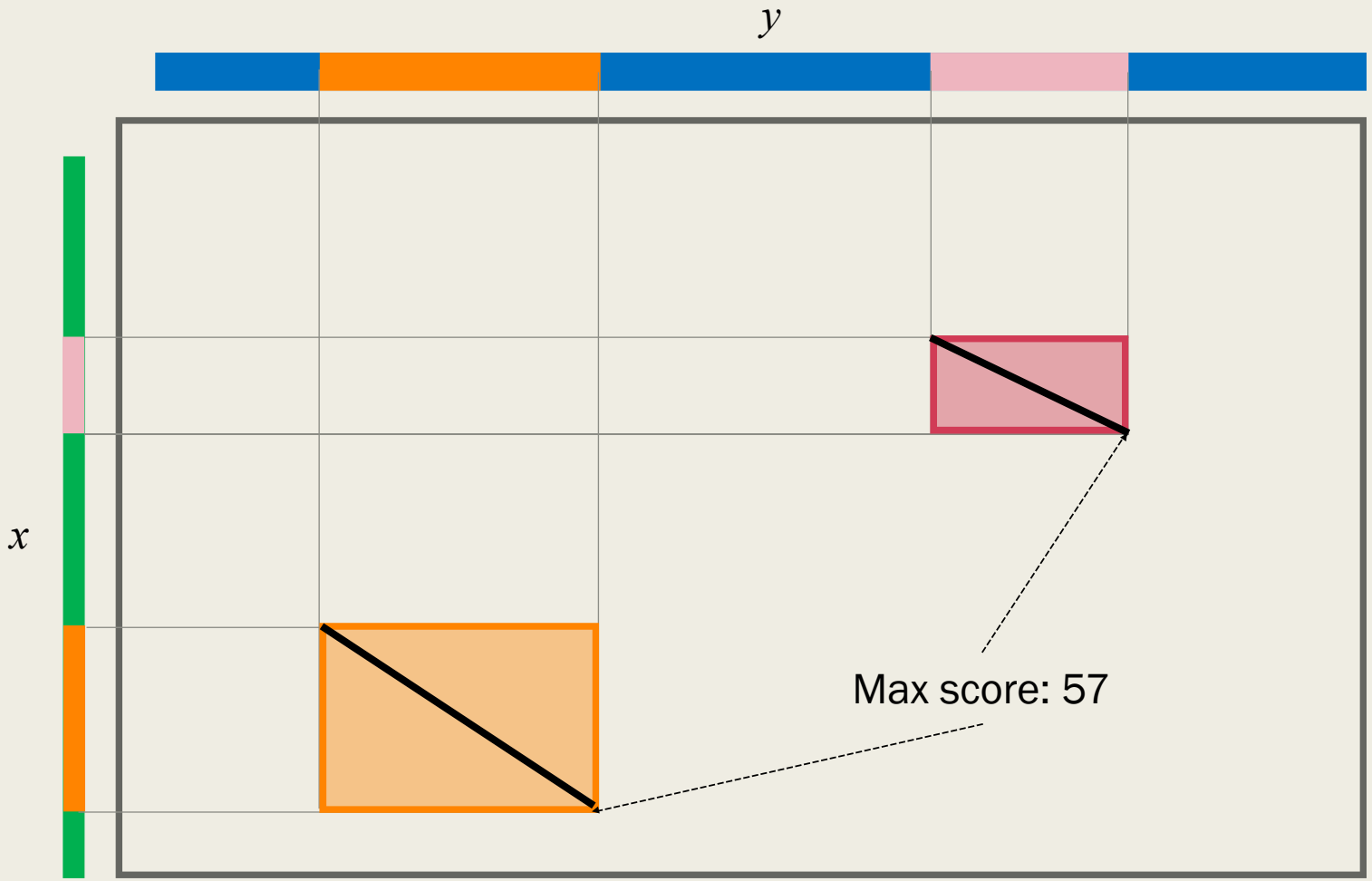
Example of local alignment picking up on small regions of high similarity



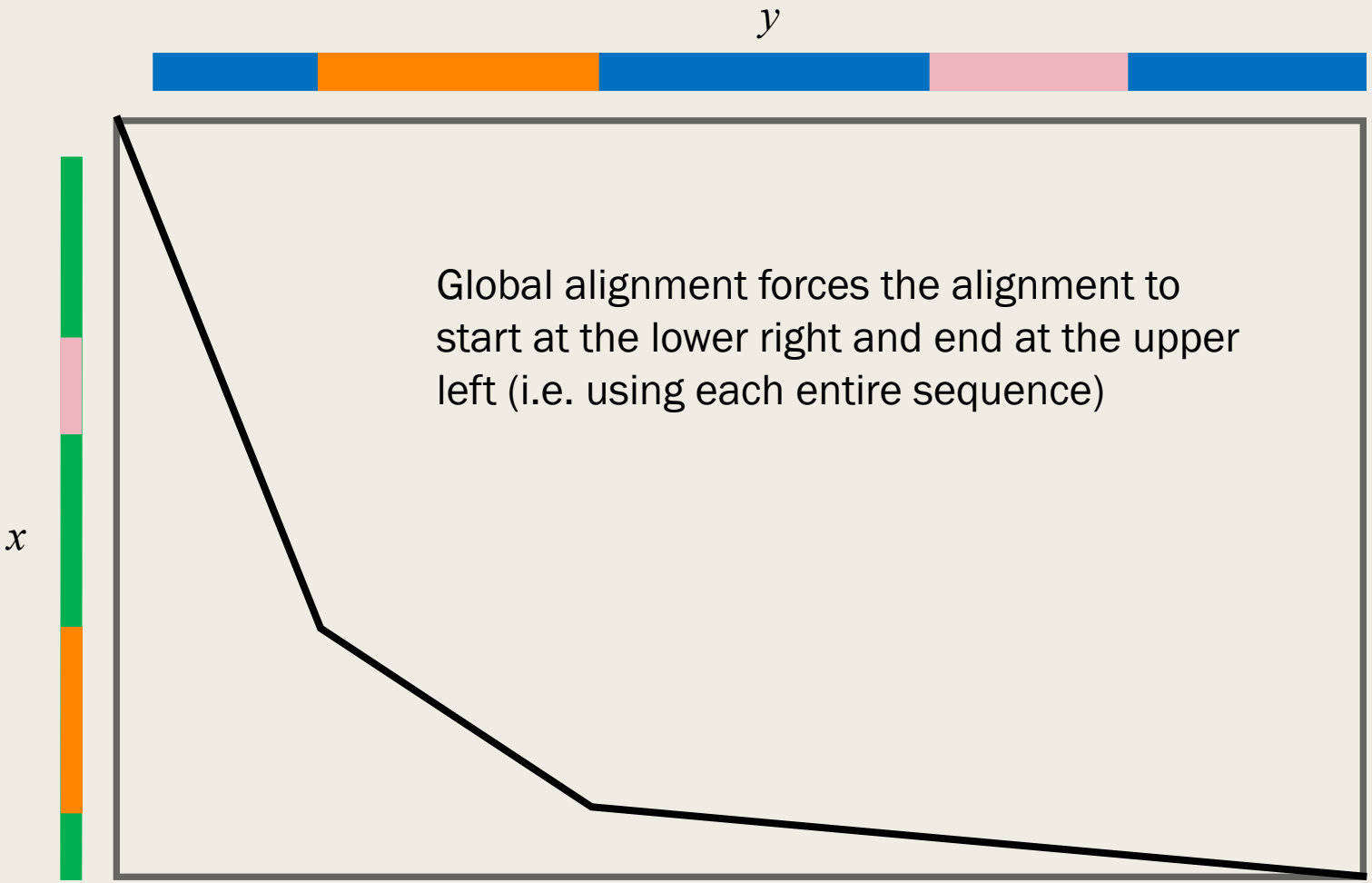
Example of local alignment picking up on small regions of high similarity



Example of local alignment picking up on small regions of high similarity



Example of a poor global alignment



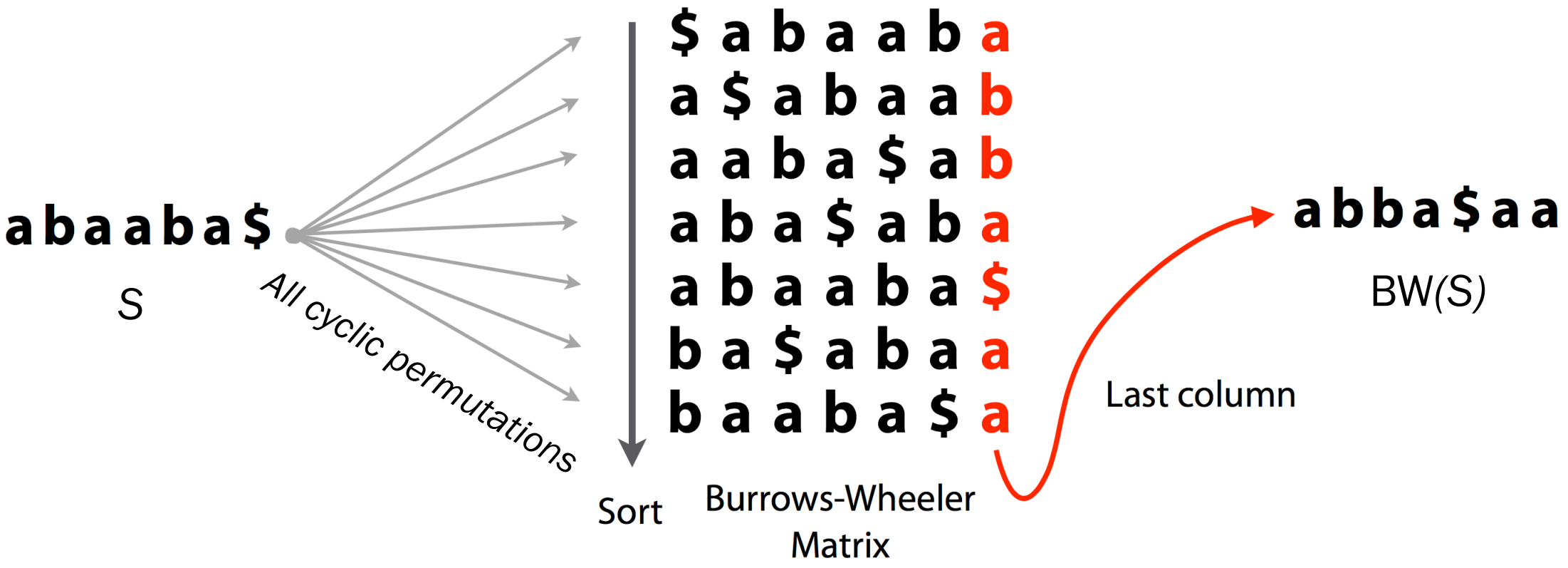
Recap BWT

Goal: read mapping

- In read mapping, y = entire genome and x = single read, and we want to know the position of the read.
- Dynamic programming approaches to alignment were designed for the case when len(x) ~= len(y) and x and y are potentially quite different
- In read mapping, not only are the lengths extremely different, but we are aligning many x's to the same y, with high similarity between x and y
- Can we do better than *O(GLn)*? Where G = len(genome), L = len(read), n = number of reads

BWT

Reversible permutation of the characters of a string, used originally for compression



Slide: adapted from Ben Langmead, John Hopkins

Example from the original BWT paper

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Figure 1: Example of sorted rotations. Twenty consecutive rotations from the sorted list of rotations of a version of this paper are shown, together with the final character of each rotation.

Burrows M, Wheeler DJ: "A block sorting lossless data compression algorithm." *Digital Equipment Corporation, Palo Alto, CA* 1994, Technical Report 124; 1994

BWT

$S = \underset{1\,2\,3\,4\,5\,6\,7}{\text{banana\$}}$

i	Sorted suffixes		A
1	\$		7
2	a\$		6
3	ana\$		4
4	anana\$		2
5	banana\$		1
6	na\$	a	5 → 4
7	nana\$	a	3 → 2

suffix array

FM-Index

Ferragina & Manzini (2000)

Example: S = "abaaba$" (genome)

1 2 3 4 5 6

P = aba (read)

① Start at end of pattern

② recurse!

all instances of "a"

1 A - C 2

i	F	L	occ(a)	occ(b)
1	$	a_1	1	0
2	a_1	b_1	1	1
3	a_2	b_2	1	2
4	a_3	a_2	2	2
5	a_4	$	2	2
6	b_1	a_3	3	2
7	b_2	a_4	4	2

BW(S)

# of b's went from $0 \to 2$ → must be b_1 & b_2

Output

* indices of pattern in F column

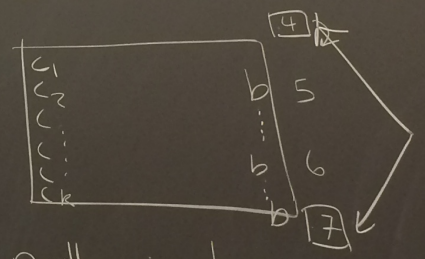
  return: $\boxed{4-5}$ inclusive

* want: return 1 & 4

  (original string indices)

Problem: Scanning is slow

Example:



b_5, b_6, b_7

pattern: ...bc

c	M[c]
\$	1
a	2
b	6

store instead of $\boxed{F}$