أكاديمية كاوست
KAUST ACADEMY

Advanced Bioinformatics

Instructor :Sakhaa Alsaedi
Sakhaa.Alsaedi@kaust.edu.sa

TA: Ebtihal Hani

Day 2: Genome Replication (Part 2)
26th Feb. 2024

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Outlines

- Introduction to Bio Data analysis (Python)
- Computational Application in Genome Replication
- Genome Replication Problem (Part 2)
    - Asymmetry of Replication , Skew Diagrams, Finding Frequent Words with Mismatches
- Bioinformatics Challenges with using python

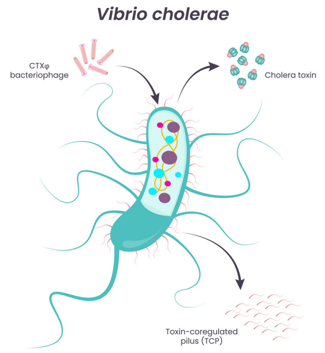
Outlines

- Introduction to Bio Data analysis (Python)
- Computational Application in Genome Replication
- Genome Replication Problem (Part 2)
- Bioinformatics Challenges with using python

Metagenomics Example

For example : *Vibrio cholerae*

OriC of VC

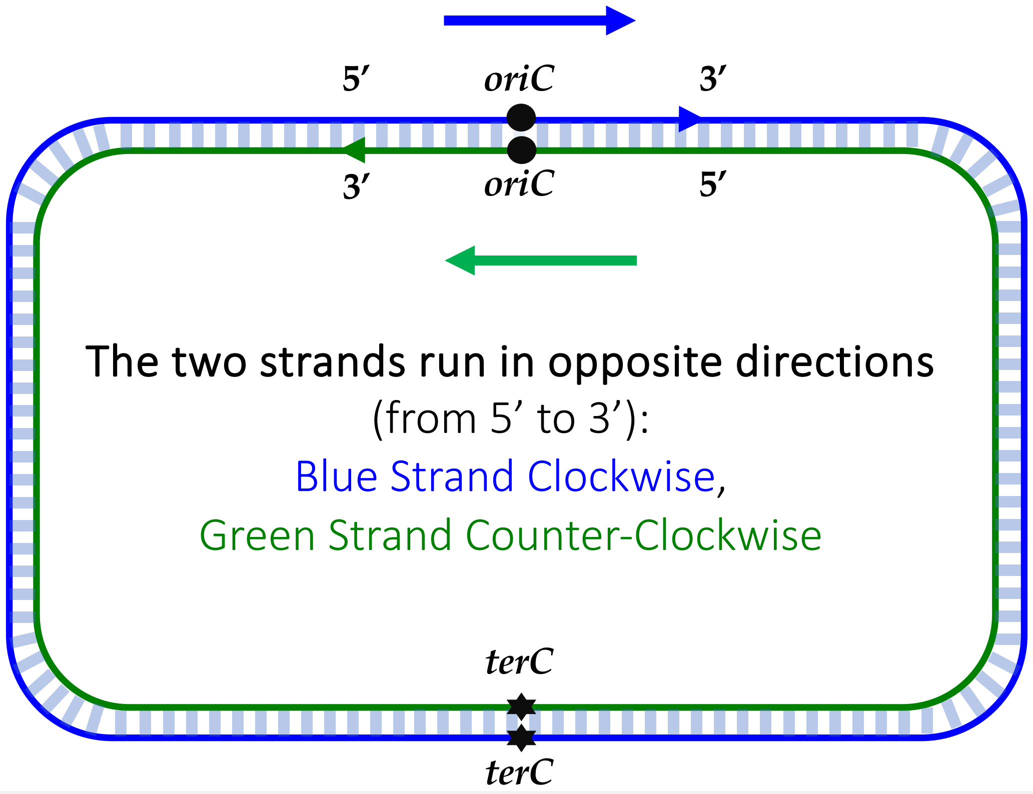


```
atcaatgatcaacgtaagcttctaagcATGATCAAGgtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagATGATCAAGagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaagaATGATCAAGctgctgctcttgatcatcgtttc
```

Outlines

- Introduction to Bio Data analysis (Python)
- Computational Application in Genome Replication
- Genome Replication Problem (Part 2)
- Bioinformatics Challenges with using python

DNA Strands Have Directions!



The two strands run in opposite directions (from 5′ to 3′):
Blue Strand Clockwise,
Green Strand Counter-Clockwise

As the strands unwind, they create two **replication forks**, which expand in both directions around the chromosome until the strands completely separate at the **replication terminus** (denoted *ter*).

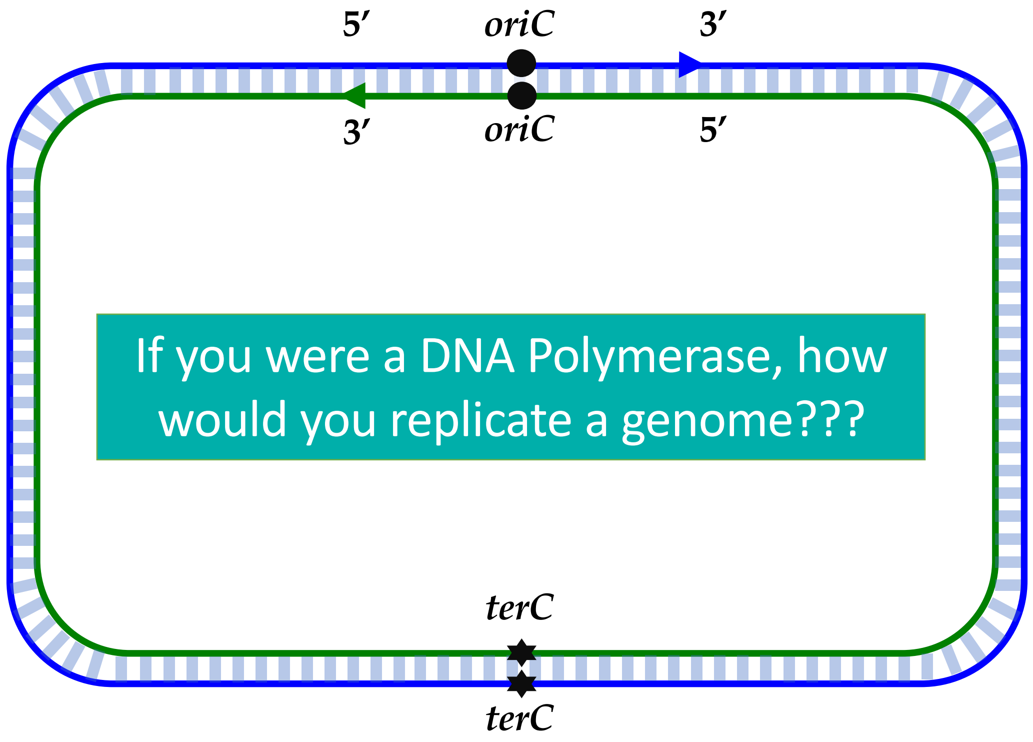
أكاديمية كاوست
KAUST ACADEMY

How DNA Polymerase replicate in a genome ?

Think and share your thoughts :)

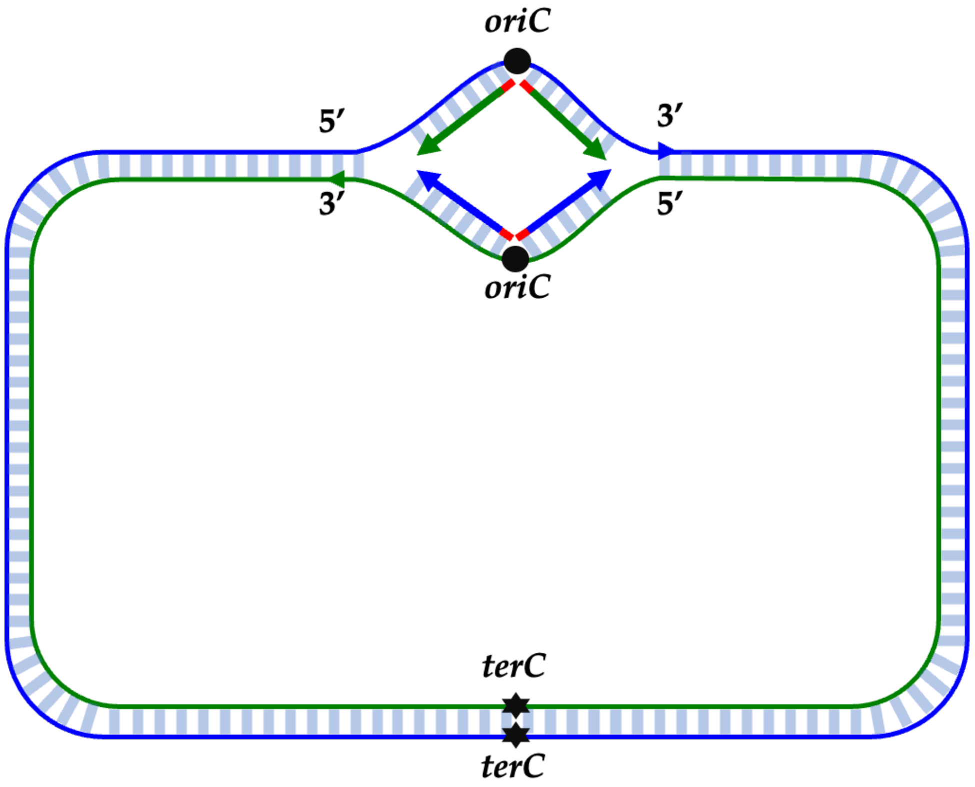
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

DNA Strands Have Directions

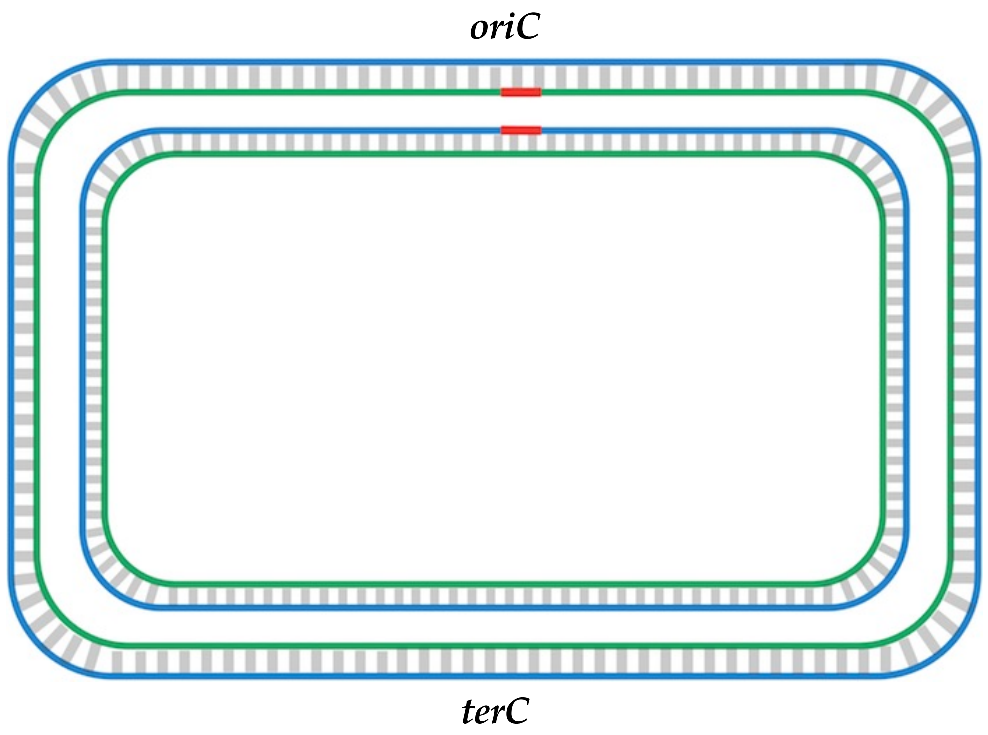


If you were a DNA Polymerase, how would you replicate a genome???

Four DNA Polymerases Do the Job

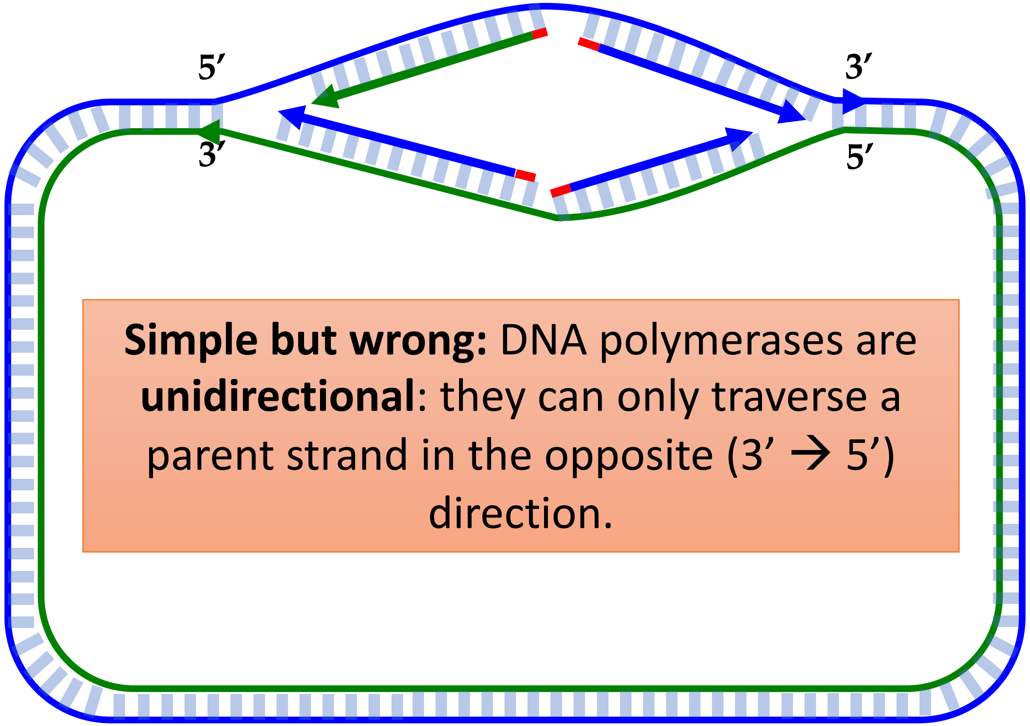


Four DNA Polymerases Do the Job



When all four DNA polymerases have reached *ter*, the chromosome's DNA will have been completely replicated, resulting in two pairs of complementary strands shown in the lower figure, and the cell is ready to divide.

Continue as Replication Fork Enlarges



Simple but wrong: DNA polymerases are **unidirectional**: they can only traverse a parent strand in the opposite (3′ → 5′) direction.

DNA polymerases are **unidirectional**, meaning that they can only traverse a template strand of DNA in the 3' → 5' direction, which is opposite from the 5′ → 3′ direction of DNA.

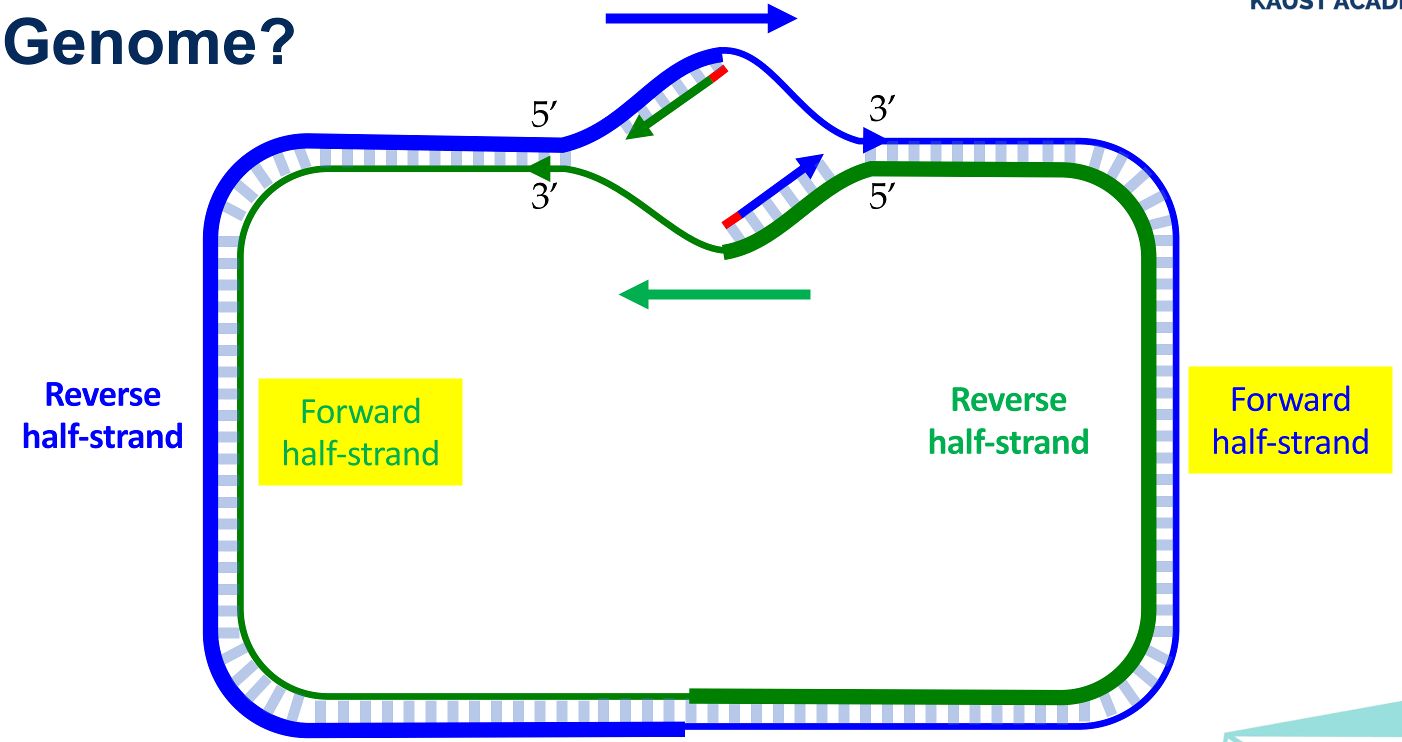
Forward and Backward Strands

- The unidirectionality of DNA polymerase requires a major revision to our naive model of replication.

- There are four different half-strands of parent DNA connecting oriC to terC, as highlighted in the following Figure.

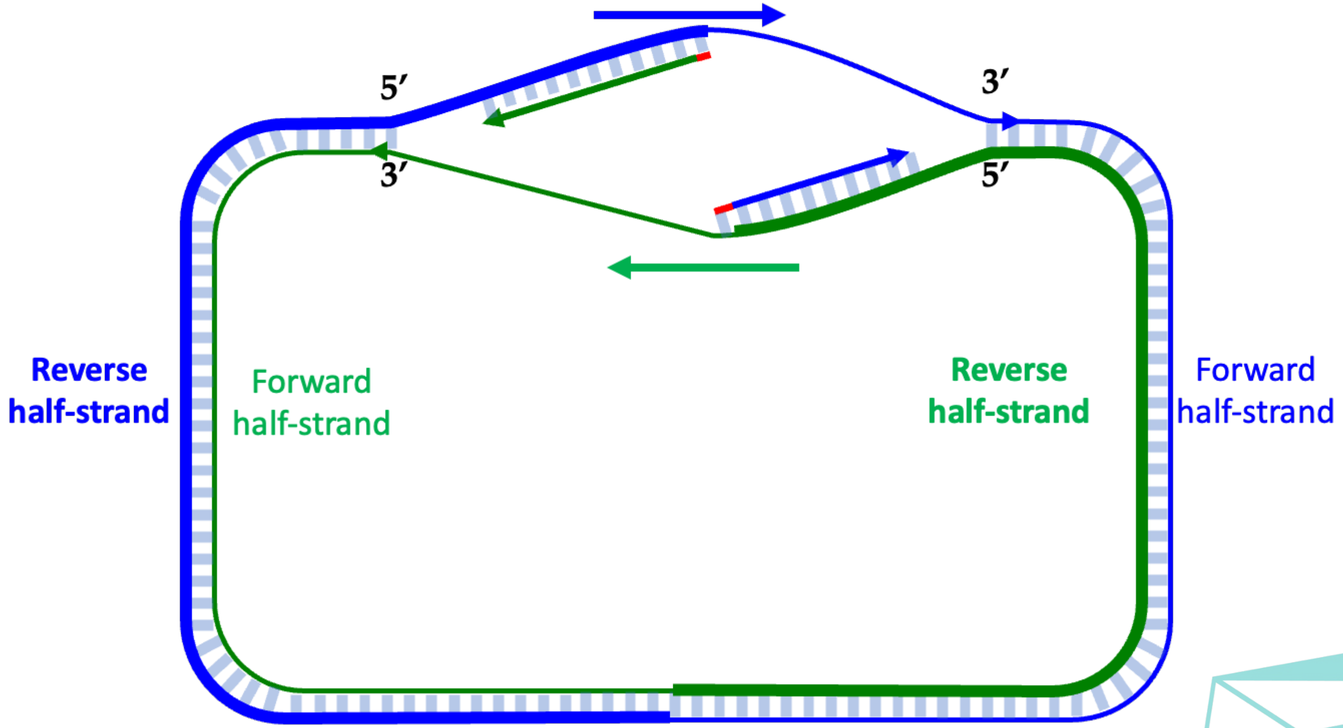
- Two of these half-strands are traversed from oriC to terC in the 5' -> 3' direction and are thus called forward half-strands.

If you Were a UNIDIRECTIONAL DNA Polymerase, how Would you Replicate a Genome?



Big problem replicating forward half-strands (thin lines).

If you Were a UNIDIRECTIONAL DNA Polymerase, How Would you Replicate a Genome???



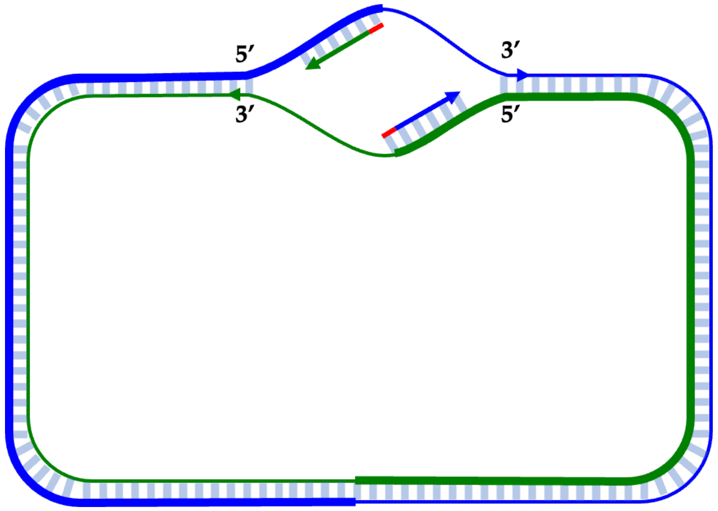
Asymmetry of Replication

- Since a DNA polymerase can only move in the reverse (3' → 5') direction, it can copy nucleotides non-stop from oriC to terC along reverse half-strands.

- However, replication on forward half-strands is very different because a DNA polymerase cannot move in the forward (5' → 3') direction; on these half-strands, a DNA polymerase must replicate backwards toward oriC.

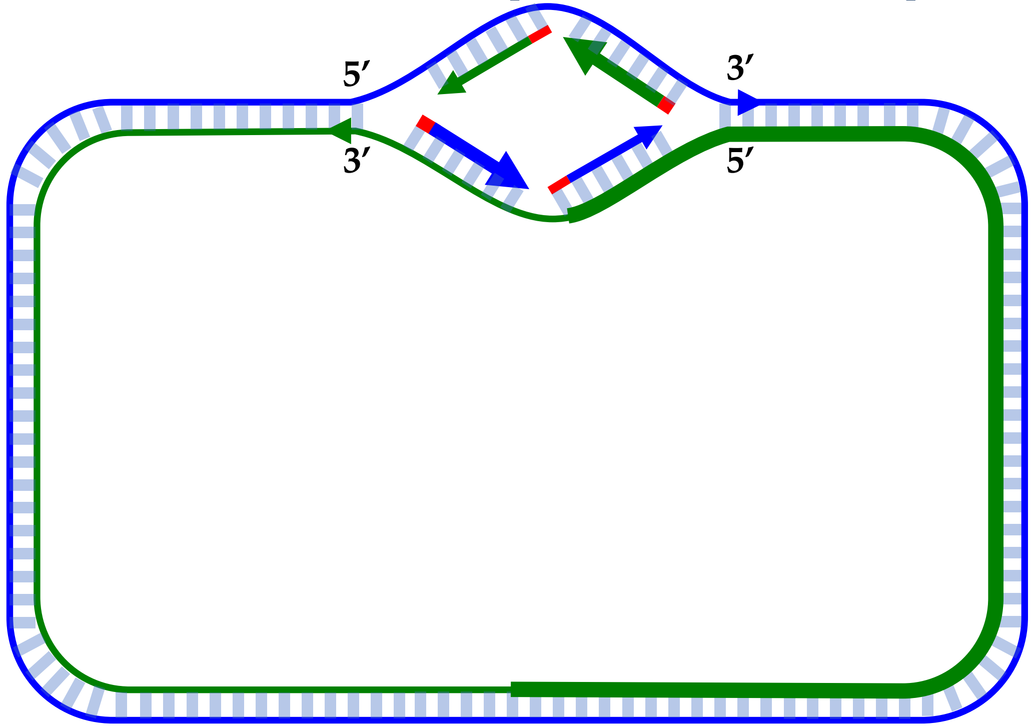
- DNA polymerase must wait for the replication fork to open a little (approximately 2,000 nucleotides) until a new primer is formed at the end of the replication fork; afterwards, the DNA polymerase starts replicating a small chunk of DNA starting from this primer and moving backward in the direction of oriC.

Wait until the Fork Opens and…



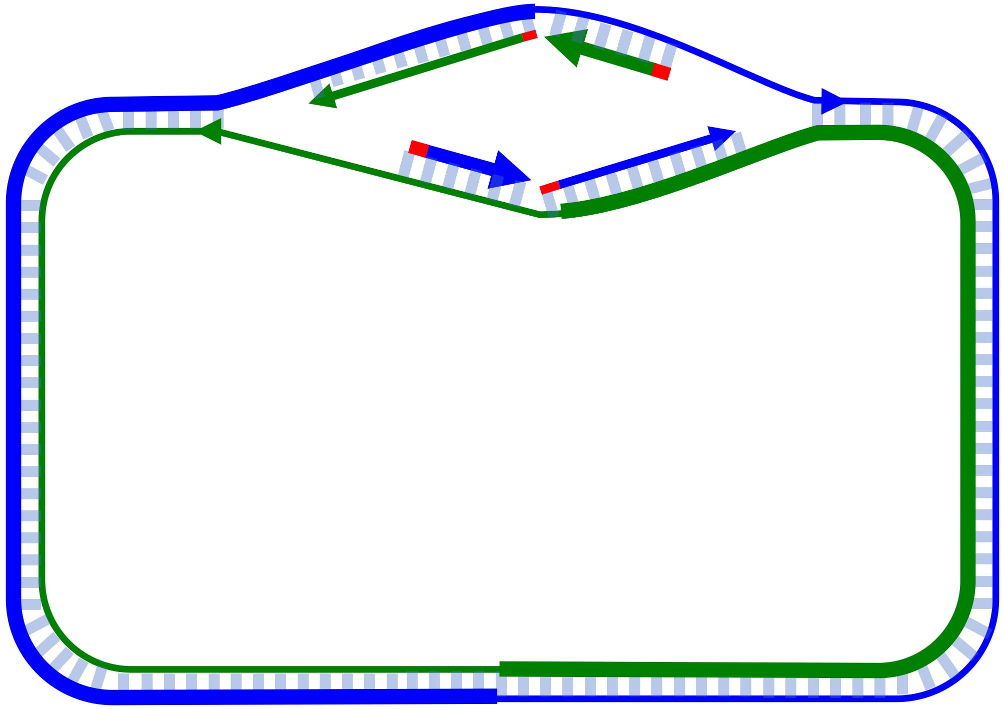
On a forward half-strand, in order to replicate DNA, a DNA polymerase must wait for the replication fork to open a little (approximately 2,000 nucleotides) until a new primer is formed at the *end* of the replication fork; afterwards, the DNA polymerase starts replicating a small chunk of DNA starting from this primer and moving *backward* in the direction of *ori*. When the two DNA polymerases on forward half-strands reach *ori*, we have the situation shown below.

Wait until the Fork Opens and Replicate

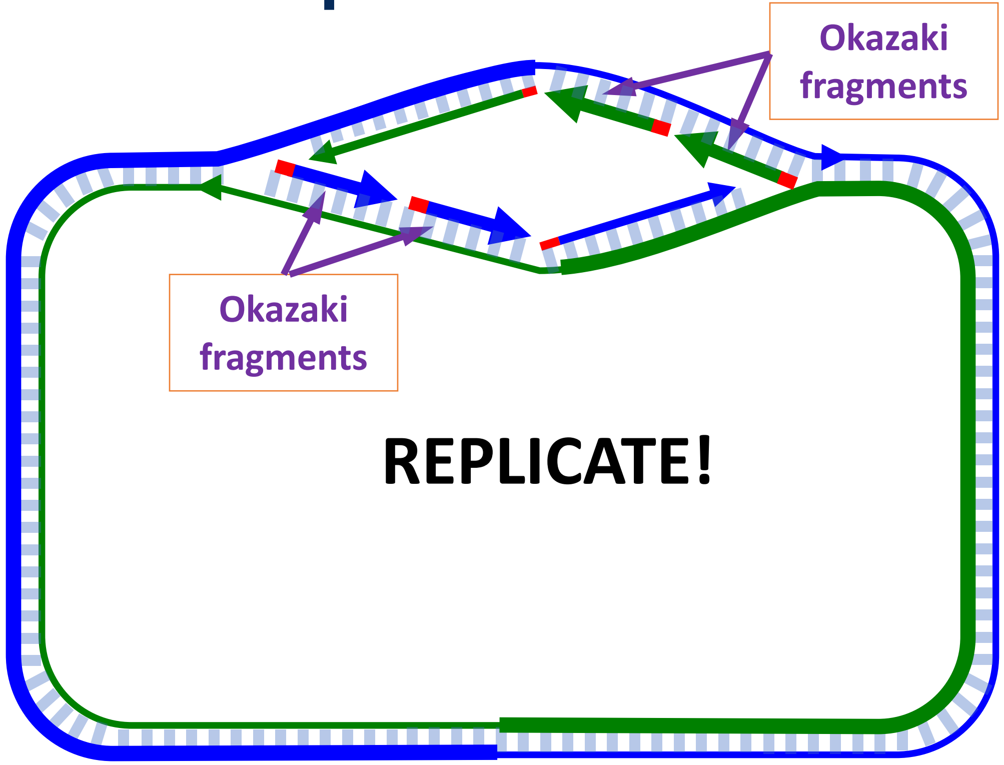


On the whole, replication on a forward half-strand requires occasional stopping and restarting, which results in the synthesis of short **Okazaki fragments** from multiple primers that are complementary to intervals on the forward half-strands

Wait until the Fork Opens and Replicate
Wait until the Fork Opens Even More and



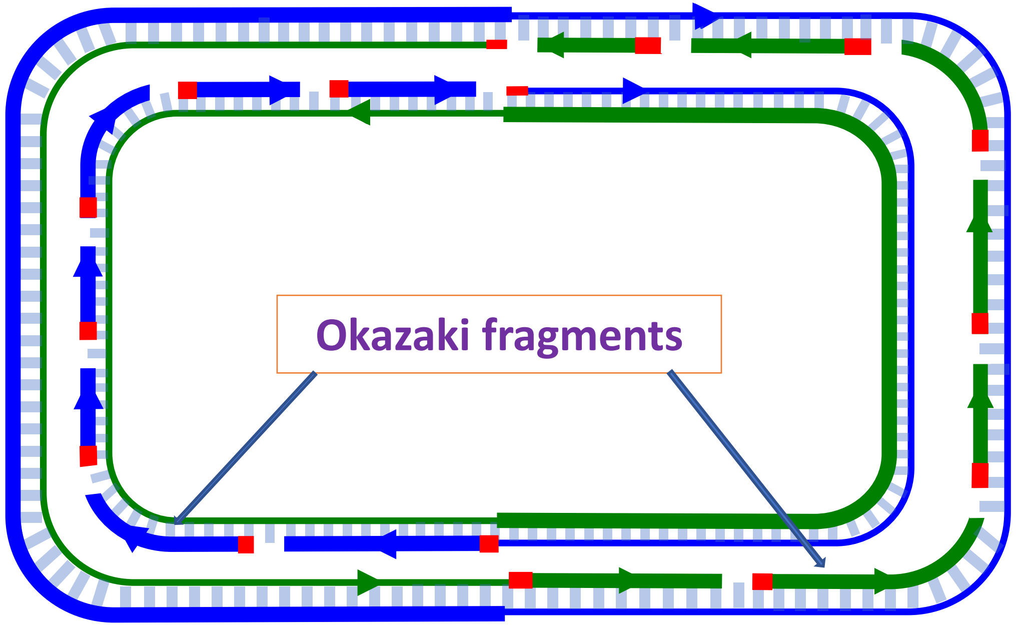
Wait until the Fork Opens and Replicate
Wait until the Fork Opens Even More and…



Instead of copying the entire half-strand, many Okazaki fragments are replicated.

Okazaki Fragments Need to be Ligated to Fill in the Gaps

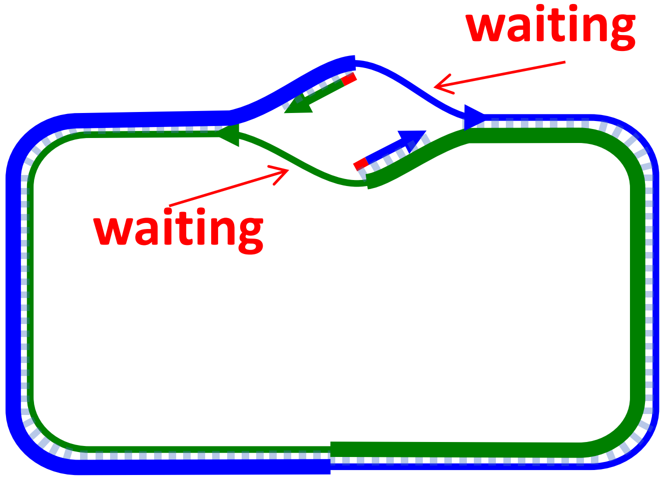
Biologists call a reverse half-strand (thick lines) a **leading half-strand** since a single DNA polymerase traverses this half-strand non-stop, and they call a forward half-strand (thin lines) a **lagging half-strand** since it is used as a template by many DNA polymerases stopping and starting replication.



The genome has been replicated!

Different Lifestyles of Reverse and Forward Half-Strands

- The **reverse half-strand** lives a **double-stranded** life most of the time.
- The **forward half-strand** spends a large portion of its life **single-stranded**, **waiting** to be replicated.



But why would a computer scientist care?

Asymmetry of Replication Affects Nucleotide Frequencies

Single-stranded DNA has a much higher mutation rate than double-stranded DNA.

Thus, if one nucleotide has a greater mutation rate, then we should observe its **shortage** on the forward half-strand that lives single-stranded life!

Which nucleotide (A/C/G/T) has the highest mutation rate? Why?

The Peculiar Statistics of #G - #C

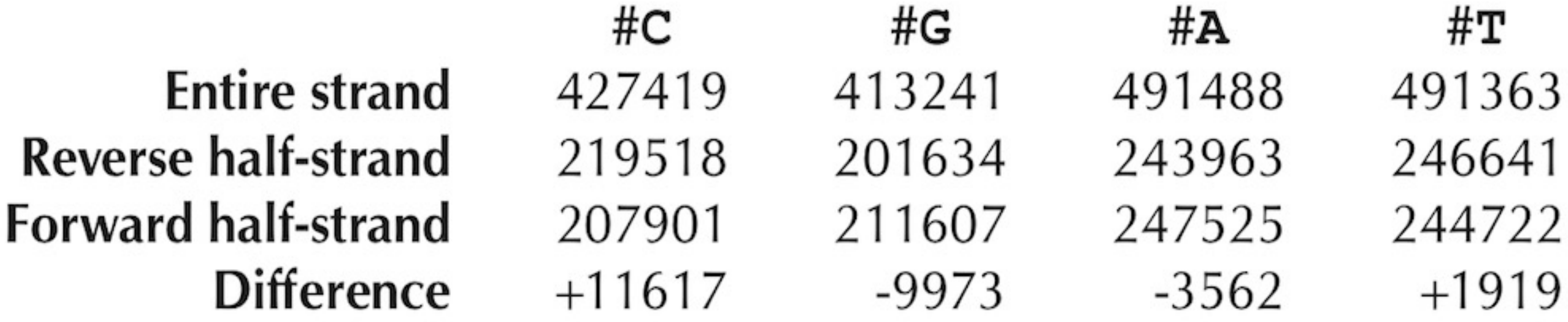
Cytosine (C) rapidly mutates into thymine (T) through **deamination**; deamination rates rise 100-fold when DNA is single stranded!

Forward half-strand (single-stranded life): **shortage of C, normal G**
Reverse half-strand (double-stranded life): **shortage of G, normal C**

	#C	#G	#G - #C
Reverse half-strand	219518	201634	**-17884**
Forward half-strand	207901	211607	**+3706**
Difference	**+11617**	**-9973**	

deamination : Spontaneous deamination con verts cytosine to uracil

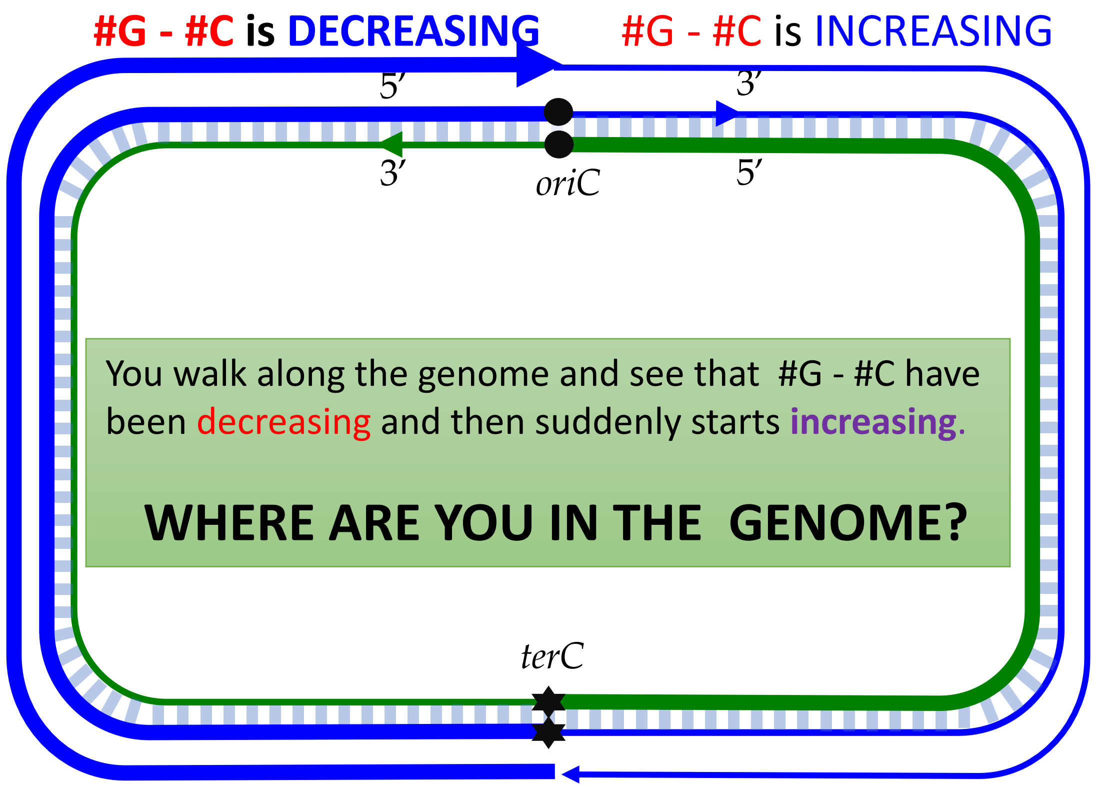
Deamination



	#C	#G	#A	#T
Entire strand	427419	413241	491488	491363
Reverse half-strand	219518	201634	243963	246641
Forward half-strand	207901	211607	247525	244722
Difference	+11617	-9973	-3562	+1919

STOP and Think: Do you notice anything about the nucleotide counts in this table?

Take a Walk Along the Genome



You walk along the genome and see that #G - #C have been decreasing and then suddenly starts increasing.

WHERE ARE YOU IN THE GENOME?

Let's see if we can take advantage of these peculiar statistics caused by deamination to locate *ori*.

C high/G low → #G - #C is DECREASING as we walk along the REVERSE half-strand

C low/G high → #G - #C is INCREASING as we walk along the FORWARD half-strand

Outlines

- Introduction to Bio Data analysis (Python)
- Computational Application in Genome Replication
- Genome Replication Problem (Part 2)
- Bioinformatics Challenges with using python

Bioinformatics Challenges

GitHub

Finding Frequent Words with Mismatches

- 1F: Minimize Skew
- 1G: Hamming Distance Between Two Strings
- 1H: Approximate Occurrences of a Pattern
- 1J: The Most Frequent Mismatches

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Bioinformatics Challenges

GitHub

Finding Frequent Words with Mismatches

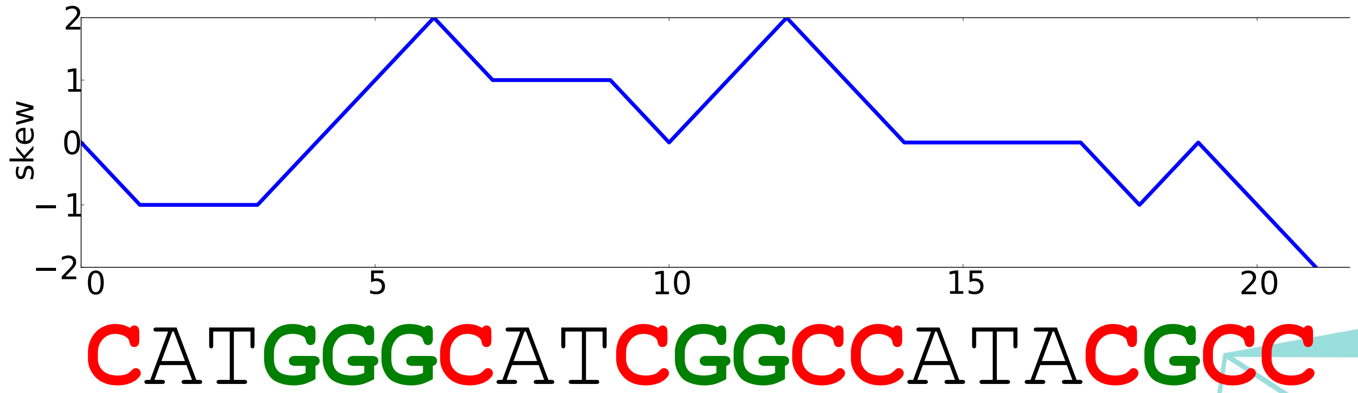
- **1F: Minimize Skew**
- 1G: Hamming Distance Between Two Strings
- 1H: Approximate Occurrences of a Pattern
- 1J: The Most Frequent Mismatches

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

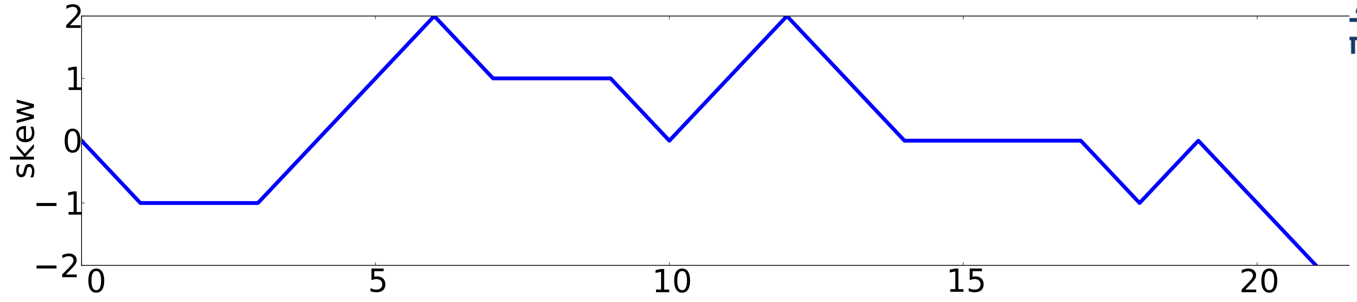
Skew Diagram

Skew(k): #G - #C for the first *k* nucleotides of *Genome*.

Skew diagram: Plot *Skew*(*k*) against *k*



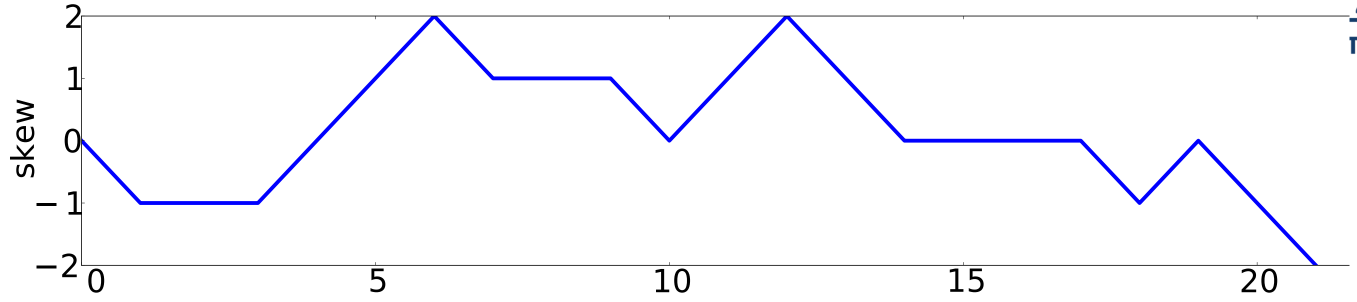
Skew Diagram



CATGGGCATCGGCCATACGCC

Since we don't know the location of *ori* in a circular genome, let's linearize it (i.e., select an arbitrary position and pretend that the genome begins here), resulting in a linear string *Genome*. We define *Skew*$_i$(*Genome*) as the difference between the total number of occurrences of G and the total number of occurrences of C in the first *i* nucleotides of *Genome*.

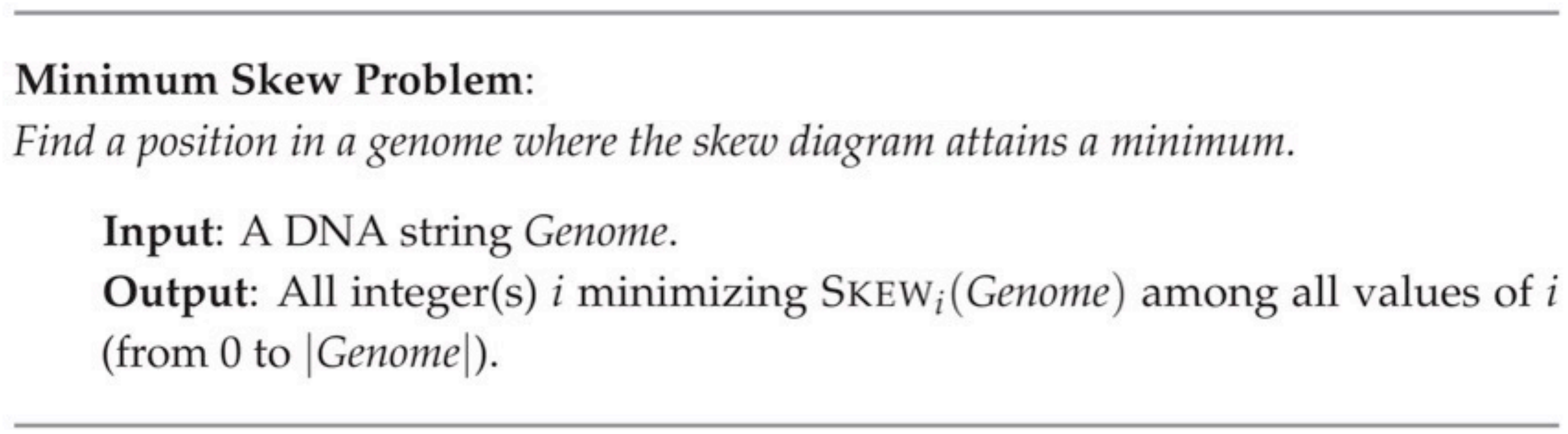
Skew Diagram



CATGGGCATCGGCCATACGCC

The **skew diagram** is defined by plotting *Skew*$_i$ (*Genome*) (as *i* ranges from 0 to |*Genome*|), where *Skew*$_0$ (*Genome*) is set equal to zero. The figure below shows a skew diagram for the DNA string CATGGGCATCGGCCATACGCC. Note that we can compute *Skew*$_{i+1}$(*Genome*) from *Skew*$_i$(*Genome*) according to the nucleotide in position *i* of *Genome*. If this nucleotide is G, then *Skew*$_{i+1}$(*Genome*) = *Skew*$_i$(*Genome*) + 1; if this nucleotide is C, then *Skew*$_{i+1}$(*Genome*)= *Skew*$_i$(*Genome*) – 1; otherwise, *Skew*$_{i+1}$(*Genome*) = *Skew*$_i$(*Genome*).

Minimum Skew Problem



Minimum Skew Problem:
Find a position in a genome where the skew diagram attains a minimum.

> **Input:** A DNA string *Genome*.
> **Output:** All integer(s) i minimizing $\textsc{Skew}_i(Genome)$ among all values of i (from 0 to $|Genome|$).

Bioinformatics Challenges

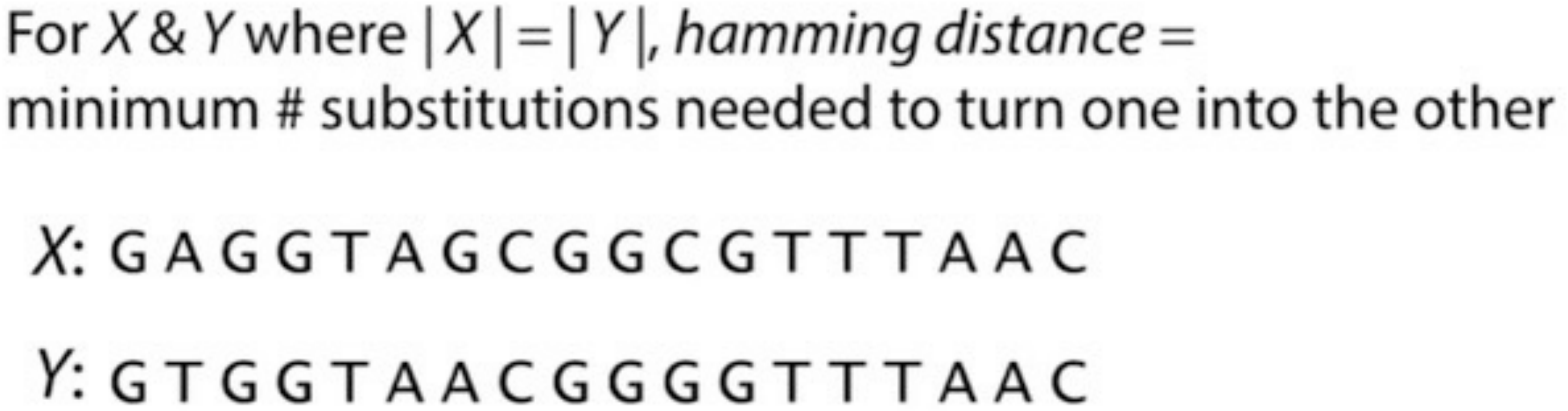
GitHub

Finding Frequent Words with Mismatches

- 1F: Minimize Skew
- **1G: Hamming Distance Between Two Strings**
- 1H: Approximate Occurrences of a Pattern
- 1J: The Most Frequent Mismatches

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Hamming Distance

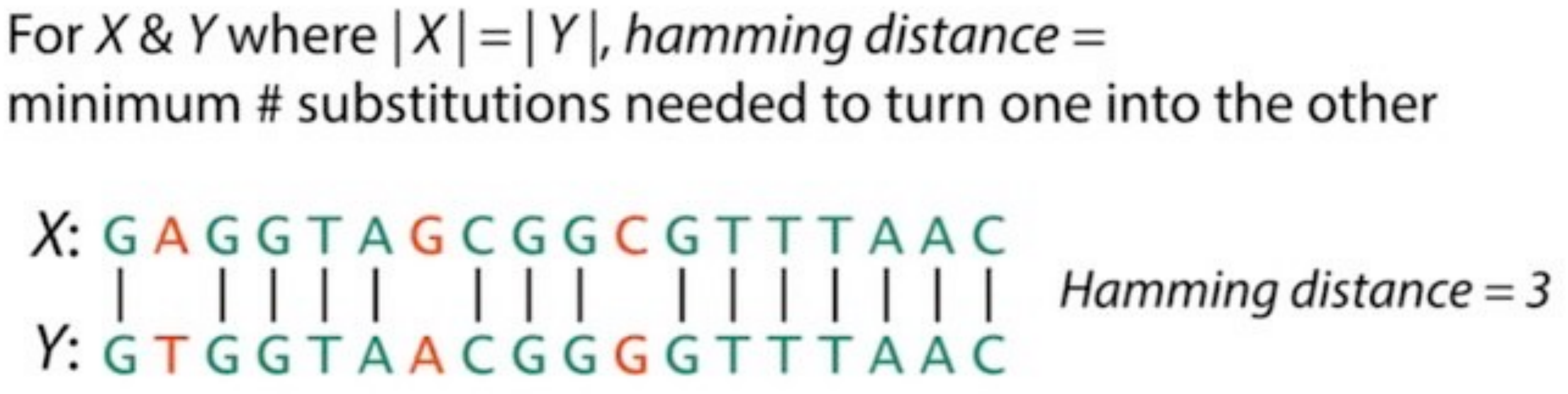


For X & Y where $|X| = |Y|$, *hamming distance* = minimum # substitutions needed to turn one into the other

X: G A G G T A G C G G C G T T T A A C

Y: G T G G T A A C G G G G T T T A A C

Hamming Distance

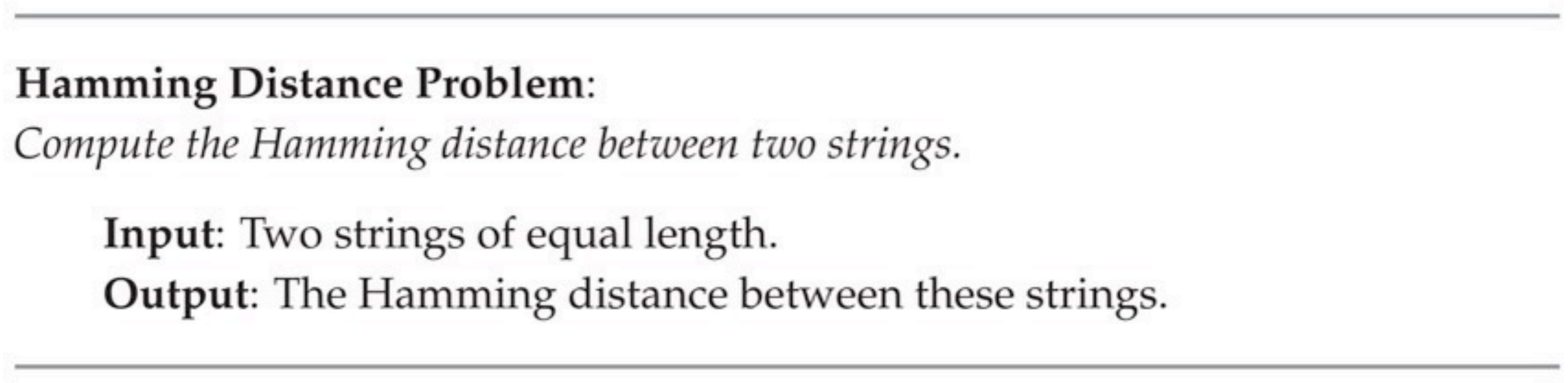


For X & Y where $|X| = |Y|$, *hamming distance* = minimum # substitutions needed to turn one into the other

```
X: G A G G T A G C G G C G T T T A A C
   |   | | | |   | | |   | | | | | | |
Y: G T G G T A A C G G G G T T T A A C
```

Hamming distance = 3

2. Hamming Distance



Hamming Distance Problem:
Compute the Hamming distance between two strings.

> **Input:** Two strings of equal length.
> **Output:** The Hamming distance between these strings.

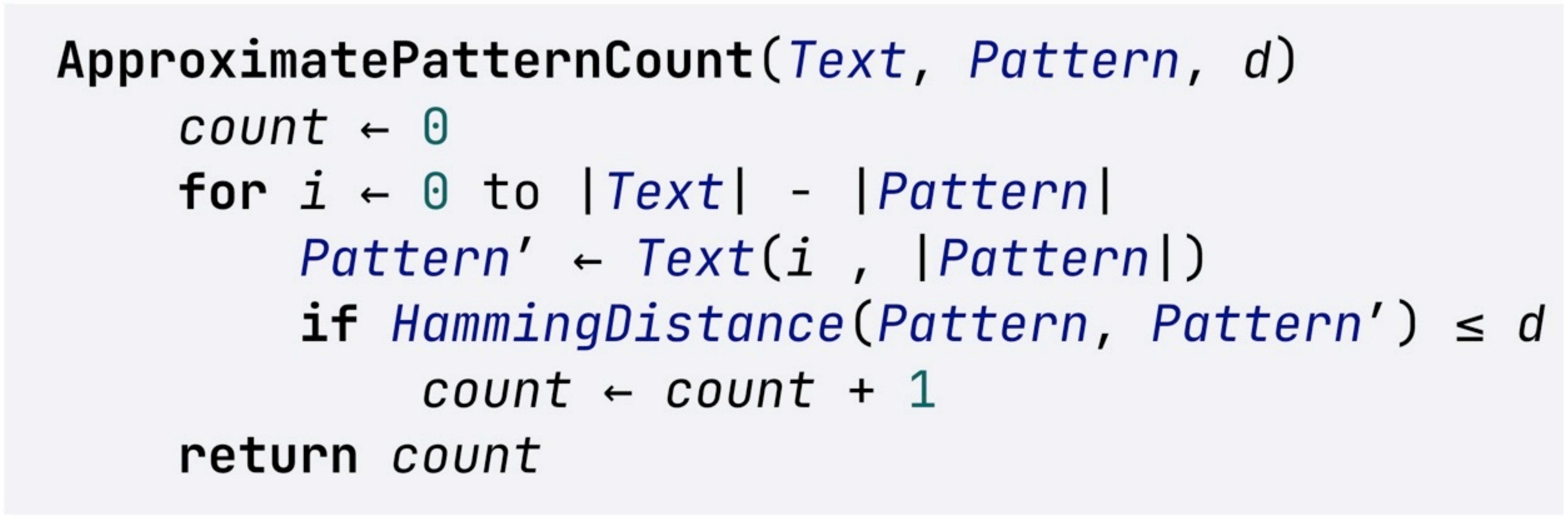
Bioinformatics Challenges

GitHub

- 1F: Minimize Skew
- 1G: Hamming Distance Between Two Strings
- **1H: Approximate Occurrences of a Pattern**
- 1J: The Most Frequent Mismatches

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

1. The Approximate Pattern Matching Problem.



```
ApproximatePatternCount(Text, Pattern, d)
    count ← 0
    for i ← 0 to |Text| - |Pattern|
        Pattern′ ← Text(i , |Pattern|)
        if HammingDistance(Pattern, Pattern′) ≤ d
            count ← count + 1
    return count
```

Code Challenge: Implement `ApproximatePatternCount()`.

Bioinformatics Challenges

GitHub

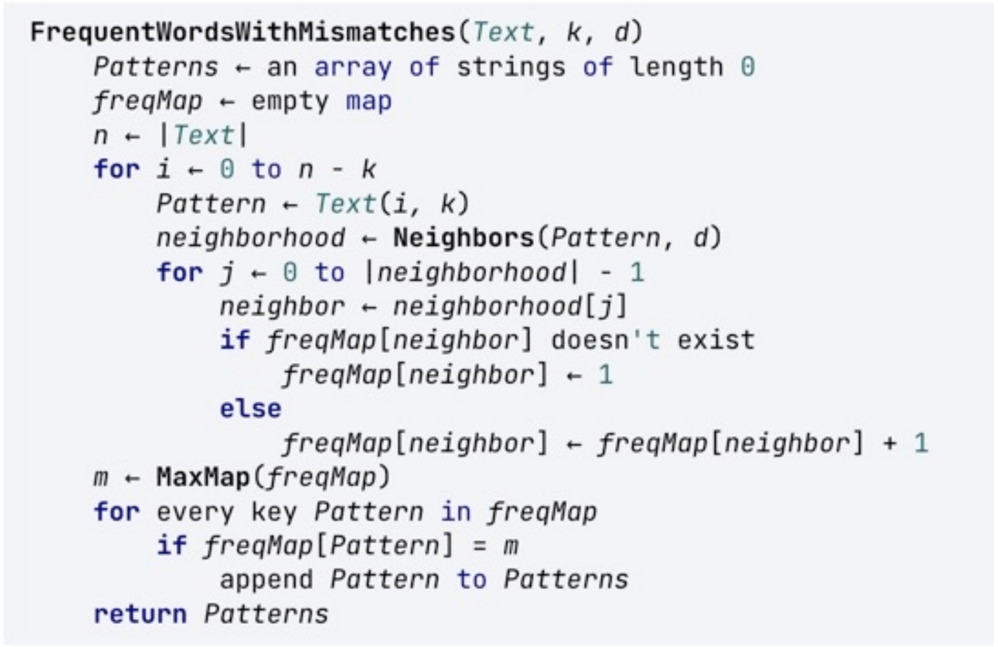
- 1F: Minimize Skew
- 1G: Hamming Distance Between Two Strings
- 1H: Approximate Occurrences of a Pattern
- **1J: The Most Frequent Mismatches**

Frequent Words with Mismatches Problem

Frequent Words with Mismatches Problem: *Find the most frequent k-mers with mismatches in a string*.
•**Input**: A string *Text* as well as integers *k* and *d*.
•**Output**: All most frequent *k*-mers with up to *d* mismatches in *Text*.

For example, to generate *Neighbors*(CAA,1), first form *Neighbors*(AA,1) = {AA, CA, GA, TA, AC, AG, AT}. The Hamming distance between AA and each of six of these neighbors is 1. Firstly, concatenating C with each of these patterns results in seven patterns (CAA, CCA, CGA, CTA, CAC, CAG, and CAT) that belong to *Neighbors*(CAA, 1). Secondly, concatenating any nucleotide with AA results in four patterns (AAA, CAA, GAA, and TAA) that belong to *Neighbors*(CAA, 1). Thus, *Neighbors*(CAA, 1) comprises eleven patterns.

Frequent Words with Mismatches Problem



```
FrequentWordsWithMismatches(Text, k, d)
    Patterns ← an array of strings of length 0
    freqMap ← empty map
    n ← |Text|
    for i ← 0 to n - k
        Pattern ← Text(i, k)
        neighborhood ← Neighbors(Pattern, d)
        for j ← 0 to |neighborhood| - 1
            neighbor ← neighborhood[j]
            if freqMap[neighbor] doesn't exist
                freqMap[neighbor] ← 1
            else
                freqMap[neighbor] ← freqMap[neighbor] + 1
    m ← MaxMap(freqMap)
    for every key Pattern in freqMap
        if freqMap[Pattern] = m
            append Pattern to Patterns
    return Patterns
```

Frequent Words with Mismatches and Reverse Complements Problem.

Frequent Words with Mismatches and Reverse Complements Problem: *Find the most frequent k-mers* (*with mismatches and reverse complements*) *in a string.*

•**Input**: A DNA string *Text* as well as integers *k* and *d*.

•**Output**: All *k*-mers *Pattern* maximizing the sum $Count_d(Text, Pattern) + Count_d(Text, Pattern_{rc})$ over all possible *k*-mers.

Done with Day 2, Heyyyyy!

Thank You !