Lecture 06

Pairwise Sequence Alignment - 2

Sep 13, 2021

Image by susannp4 from Pixabay

Arizona State University

What we have learned

- Sequence alignment
- Sequence similarity, Hamming distance
- Point mutations: substitution, insertion, and deletion
- Site states: match, mismatch, and gap
- Alignment score

Pairwise sequence alignment

Learning goals

(today and next class)

Math

Matrix representation of pairwise alignment

Practice

Perform N-W alignment by hand

Algorithm

Needleman-Wunsch global alignment algorithm

Computational complexity

Dynamic programming

Code

Python implementation of N-W algorithm

Matrix as nested list

While loop

An alignment task requires --

- Two sequences

Seq 1

ACGCGT

Seq 2

ATCGTA

- Scoring scheme

Scoring

Match	1
Mismatch	0
Gap	-1

Compare possible alignments (the intuitive way)

- Propose a few possible alignments, calculate their scores, and find the best one.

Score: 1

```
ACGCGT
|
ATCGTA
```

Score: 2

```
ACGCGT-
|  |||
AT-CGTA
```

Score: 0

```
A-CGCGT-
| ||  |
ATCG--TA
```

Winner

Mission: Move bases from sequences to alignment

- In each round, for each sequence,
- You can either move the left-most base to the alignment,
- Or not move anything (which will leave a gap in the alignment)

Alignment

ACGCGT
ATCGTA

Seq 1

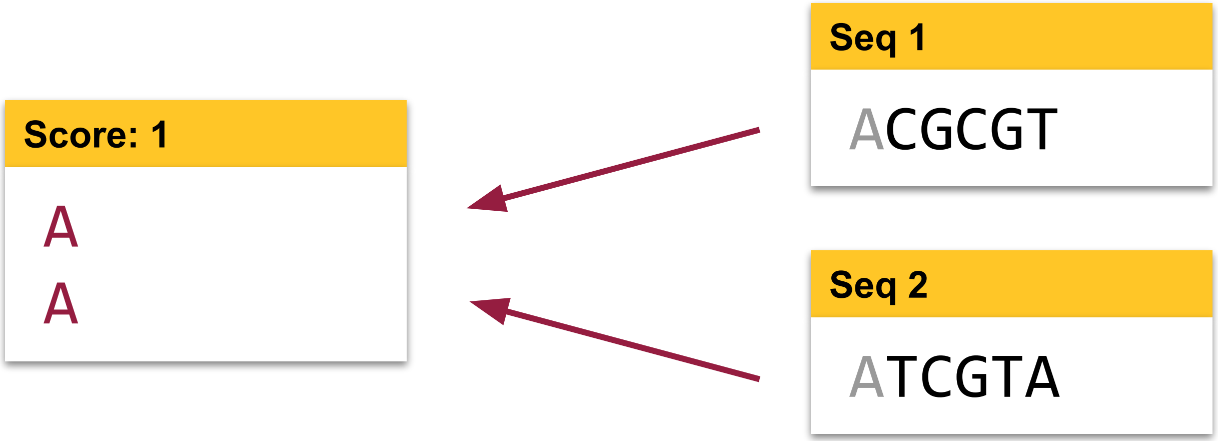
ACGCGT

Seq 2

ATCGTA

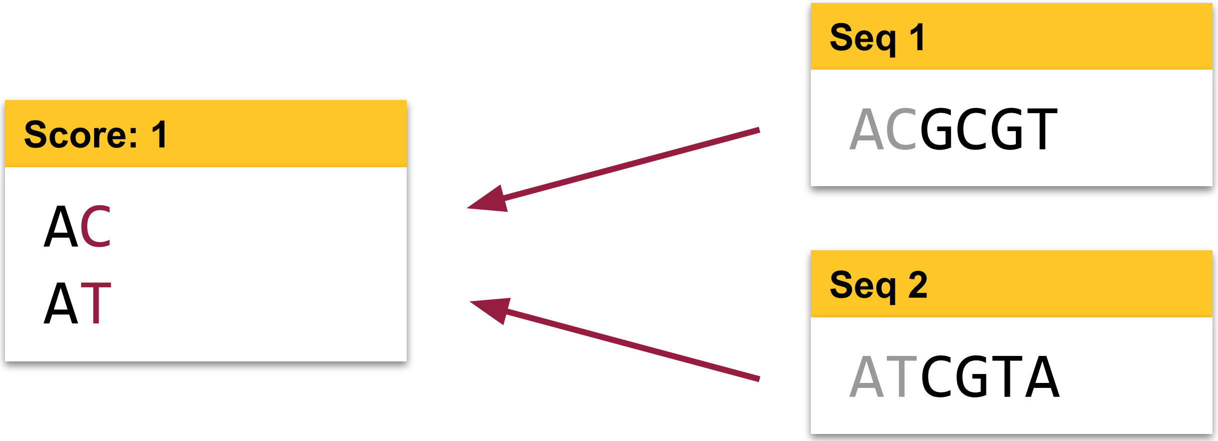
Round 1: Move one base from both sequences

- And they are a **match** (A-A).
- Alignment score + 1.



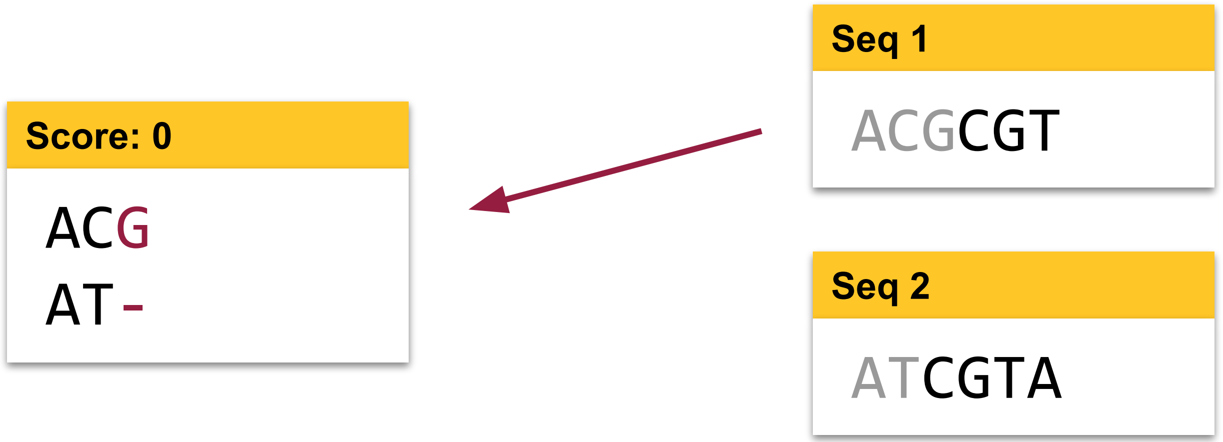
Round 2: Move one base from both sequences

- But they are a **mismatch** (C-T).
- Alignment score + 0.



Round 3: Move a base from Seq 1, but not from Seq 2

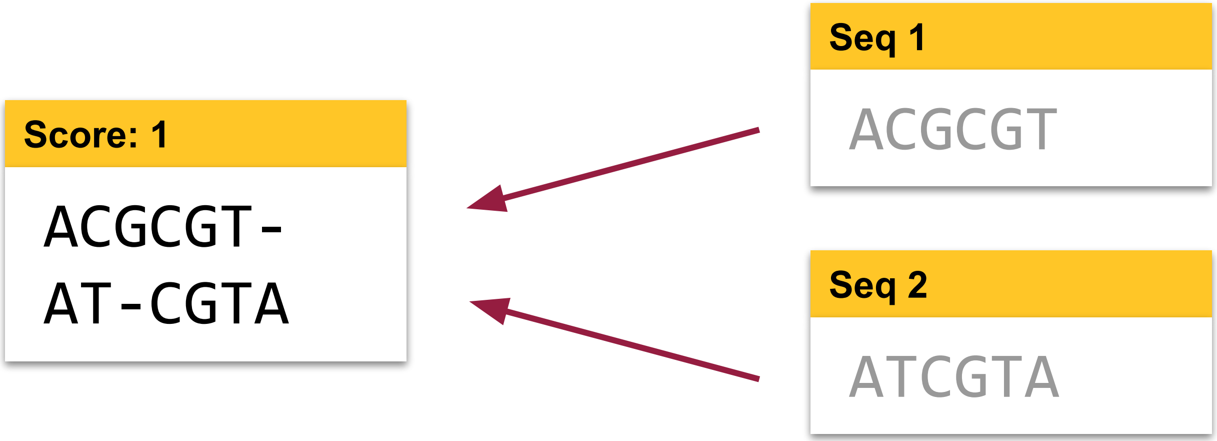
- Now there is a **gap**.
- Alignment score - 1.



- If we consider this as a process of Seq 1 evolving into Seq 2, then this is a **deletion**.
- Oppositely, it will be an **insertion** if we move from Seq 2 but not from Seq 1.

Let’s keep going until all bases are moved into the alignment

- Round 4, 5, 6: Match.
- Round 7: Insertion.



Present the entire alignment protocol

The procedures and corresponding scores are:

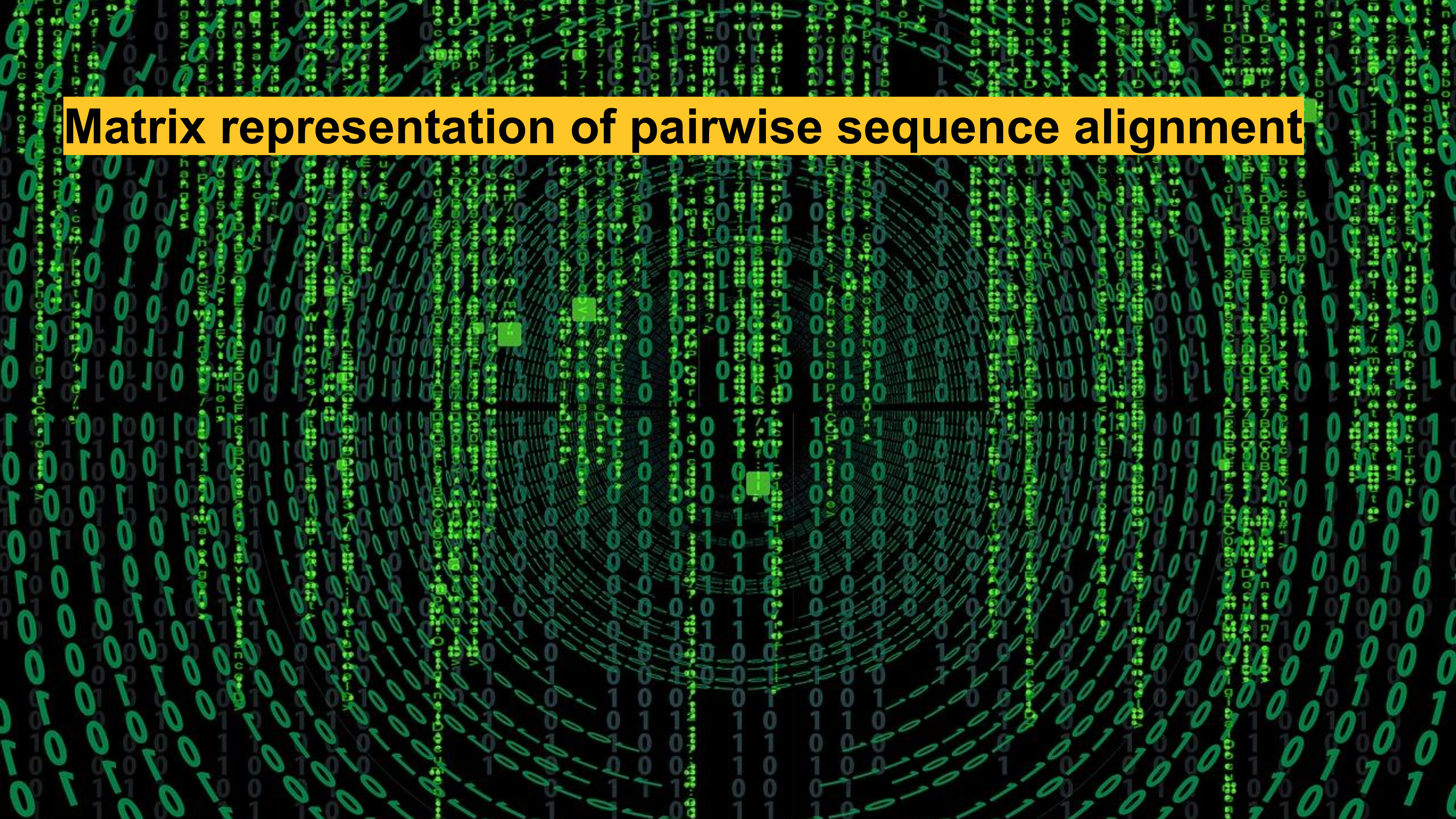
- match - mismatch - deletion - match - match - match - insertion
- 1 + 0 - 1 + 1 + 1 + 1 - 1 = 2

If we code moving as “Y”, not moving as “N”, we have:

Seq 1	Y	Y	Y	Y	Y	Y	N
Seq 2	Y	Y	N	Y	Y	Y	Y

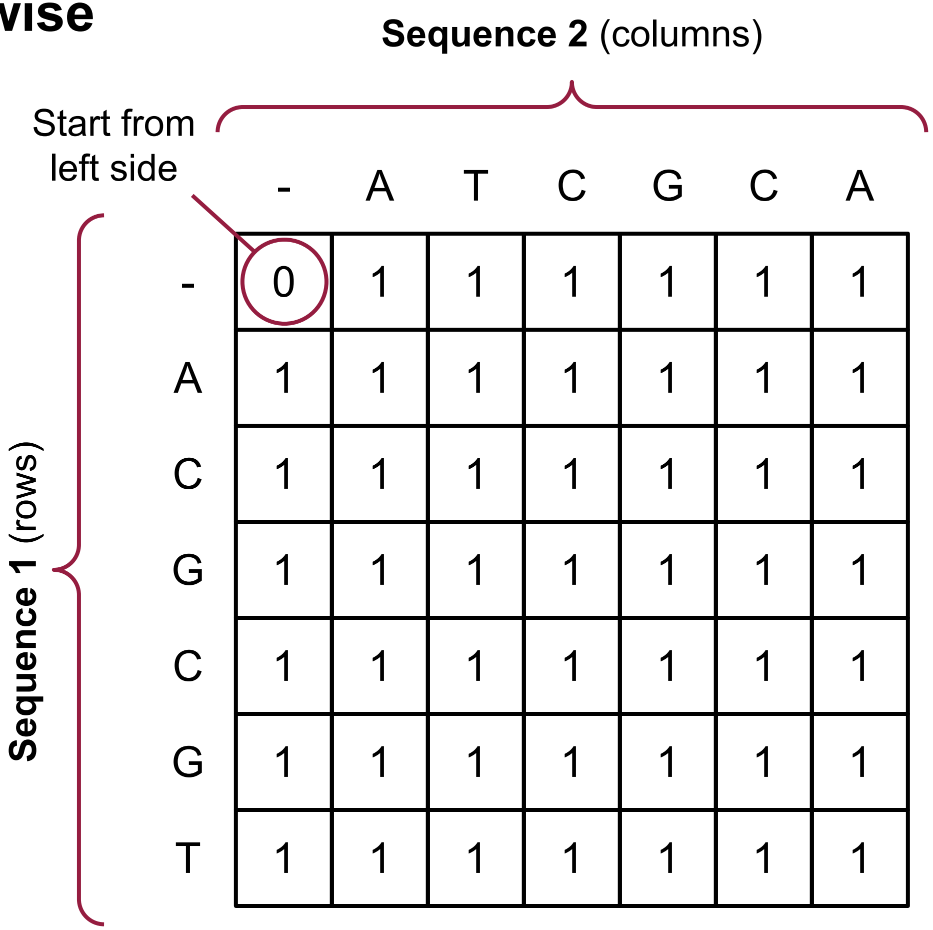
In each time, we can only explore one possible alignment.

Matrix representation of pairwise sequence alignment



Matrix representation of pairwise sequence alignment

- Step-by-step alignment starts from top-left and ends at bottom-right.
- Each step is represented by moving from a cell to its adjacent cell.

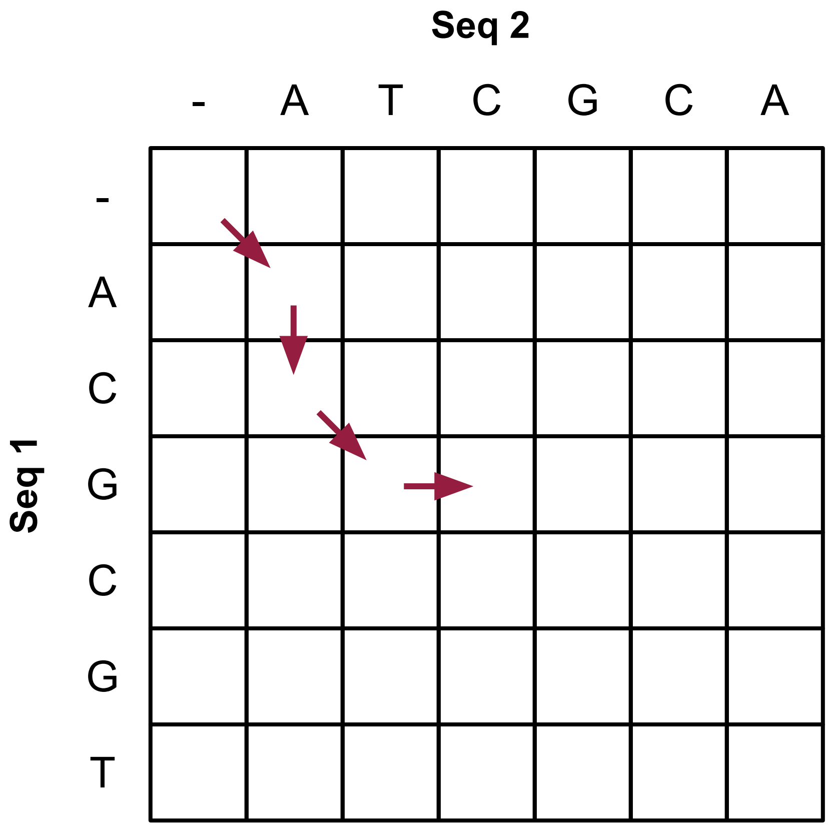


	-	A	T	C	G	C	A
-	0	1	1	1	1	1	1
A	1	1	1	1	1	1	1
C	1	1	1	1	1	1	1
G	1	1	1	1	1	1	1
C	1	1	1	1	1	1	1
G	1	1	1	1	1	1	1
T	1	1	1	1	1	1	1

Each step is represented by moving from a cell to its adjacent cell

Three possible moving directions:

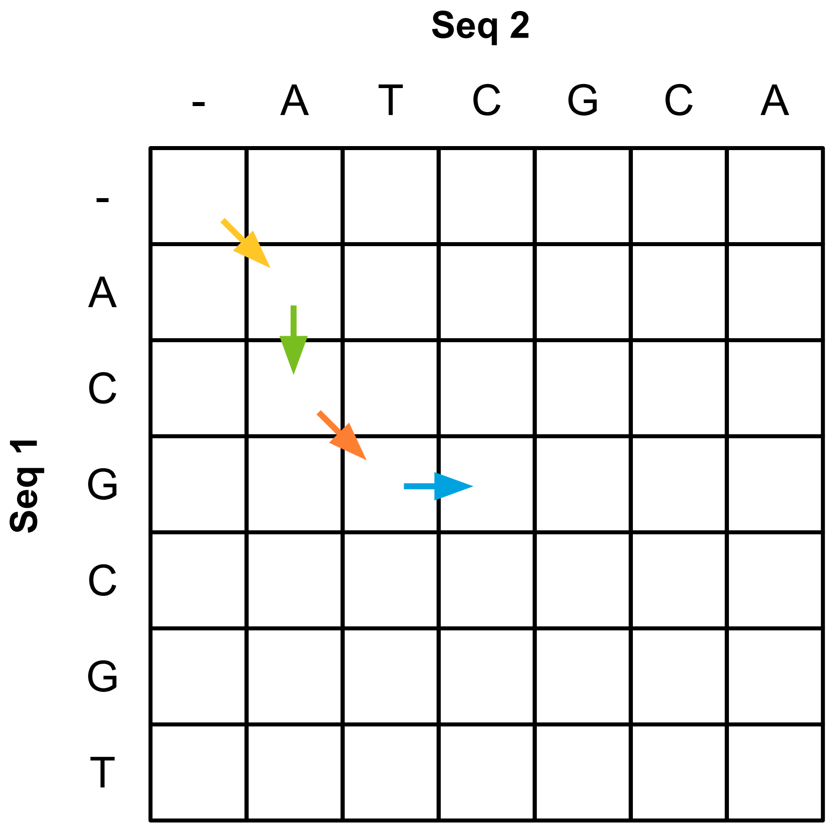
- ↘ (diagonal): Take a base from both seqs, i.e., a **(mis)match**.
- ↓ (down): Take from Seq 1 but not Seq 2, i.e., a **deletion**.
- → (right): Take from Seq 2 but not Seq 1, i.e., an **insertion**.



Each step is represented by moving from a cell to its adjacent cell

For examples:

- ↘ Is a match (`A` vs `A`)
- ↓ Is a deletion (`C` vs `-`)
- ↘ Is a mismatch (`G` vs `T`)
- → Is an insertion (`-` vs `C`)

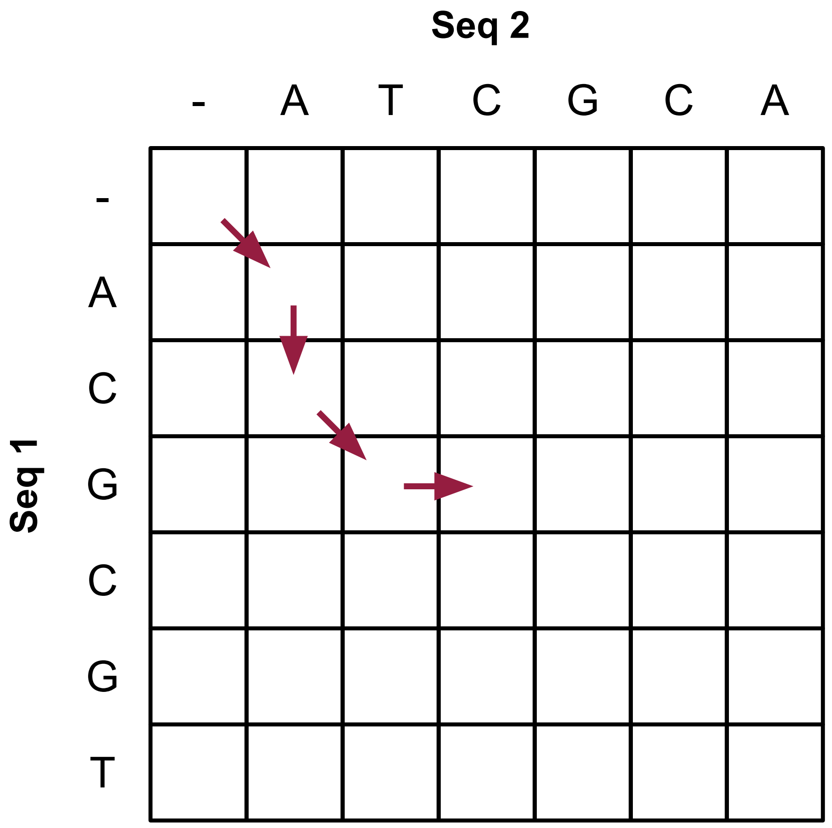


Alignment is represented by the moving path

- These four arrows represent the following alignment:

```
A C G -
A - T C
```

- Our goal is to find a path from top left to bottom right, which suffice...



Accumulative alignment scores are noted in the cells

- Start from the top left cell, with value 0 (because nothing has happened yet).

Scoring

Match	1
Mismatch	0
Gap	-1

	-	A	T	C	G	C	A
-	0						
A							
C							
G							
C							
G							
T							

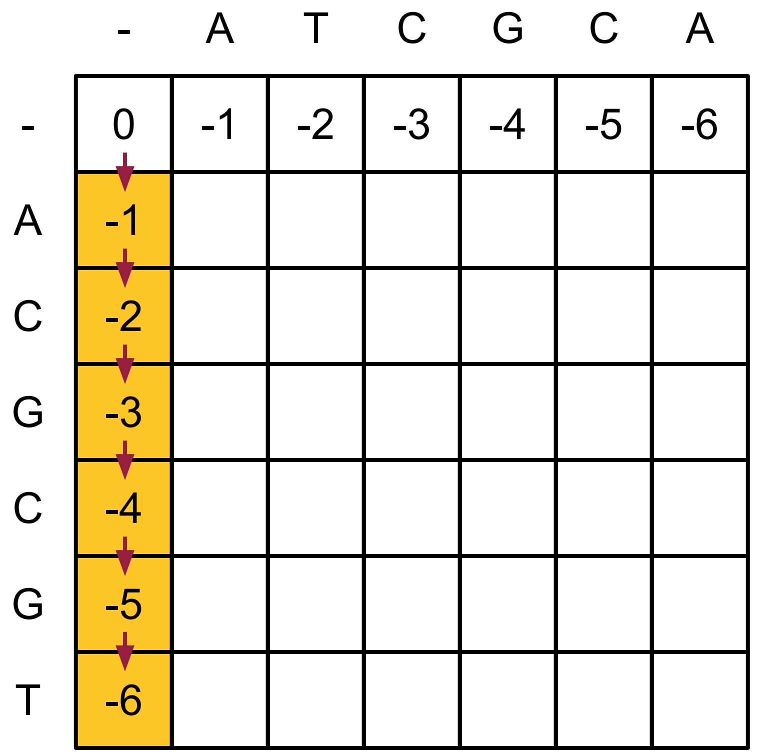
Fill the first row with accumulative gap costs

- Each right arrow represents an **insertion** (with score -1).

$$\begin{array}{c|ccccccccccccc}
 & - & & A & & T & & C & & G & & C & & A \\
\hline
- & 0 & \rightarrow & -1 & \rightarrow & -2 & \rightarrow & -3 & \rightarrow & -4 & \rightarrow & -5 & \rightarrow & -6 \\
A & & & & & & & & & & & & & \\
C & & & & & & & & & & & & & \\
G & & & & & & & & & & & & & \\
C & & & & & & & & & & & & & \\
G & & & & & & & & & & & & & \\
T & & & & & & & & & & & & &
\end{array}$$

Fill the first column in the same way

- Each down arrow represents a **deletion** (with score -1).



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

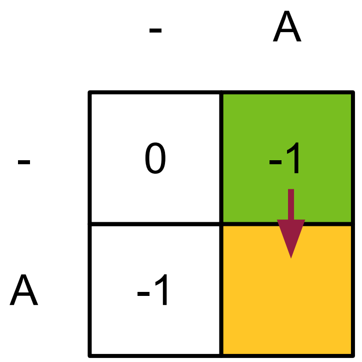
Now work on the matrix body

- Always from top / left to bottom / right.

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

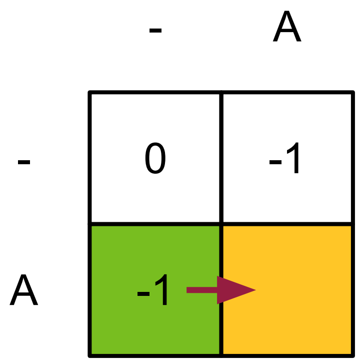
There are three ways to move to this cell

From top



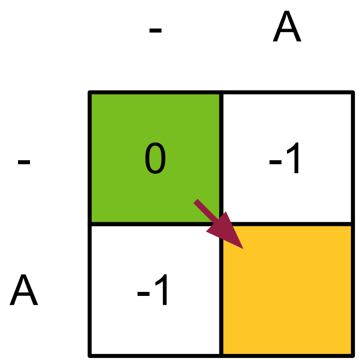
Deletion (-1)

From left



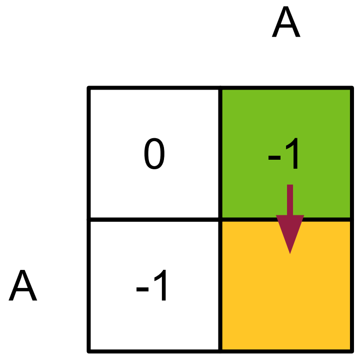
Insertion (-1)

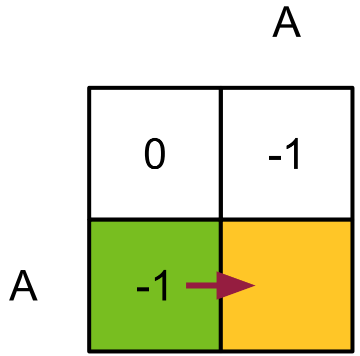
From top left (diagonal)

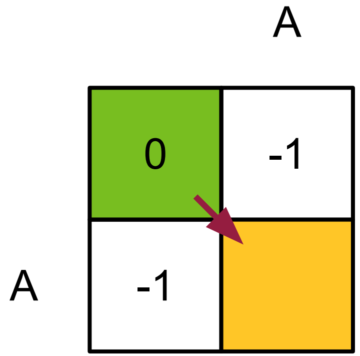


Match (+1)

Add the current score to the source cell







This step	Deletion (-1)	Insertion (-1)	Match (+1)
Accumulative	-1 - 1 = -2	-1 - 1 = -2	0 + 1 = 1

- Which accumulative score is the largest?

Winner

Keep the maximum score from the three directions

- Therefore we note the cell with the largest score (1) (from diagonal direction).

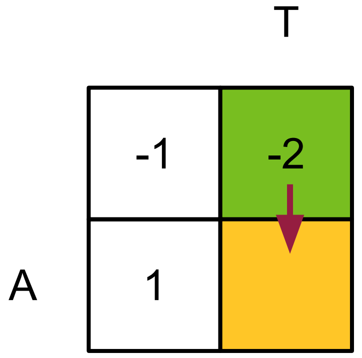
	-	A	T	C	G	C	A
-	0	-1	-2	-3	-4	-5	-6
A	-1	1					
C	-2						
G	-3						
C	-4						
G	-5						
T	-6						

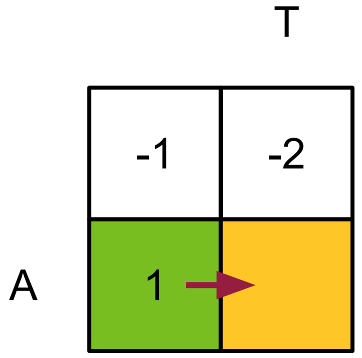
Let’s try another one

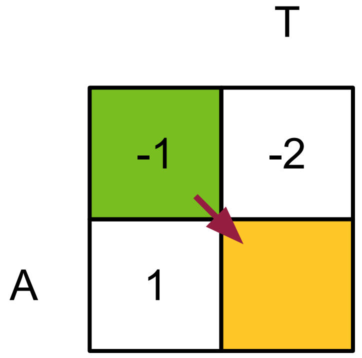
- What will be the score of this cell (and from which direction)?

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

Determine maximum score and direction in the same way







This step	Deletion (-1)	Insertion (-1)	**Mismatch** (+0)
Accumulative	-2 - 1 = -3	1 - 1 = 0	-1 + 0 = -1

Winner

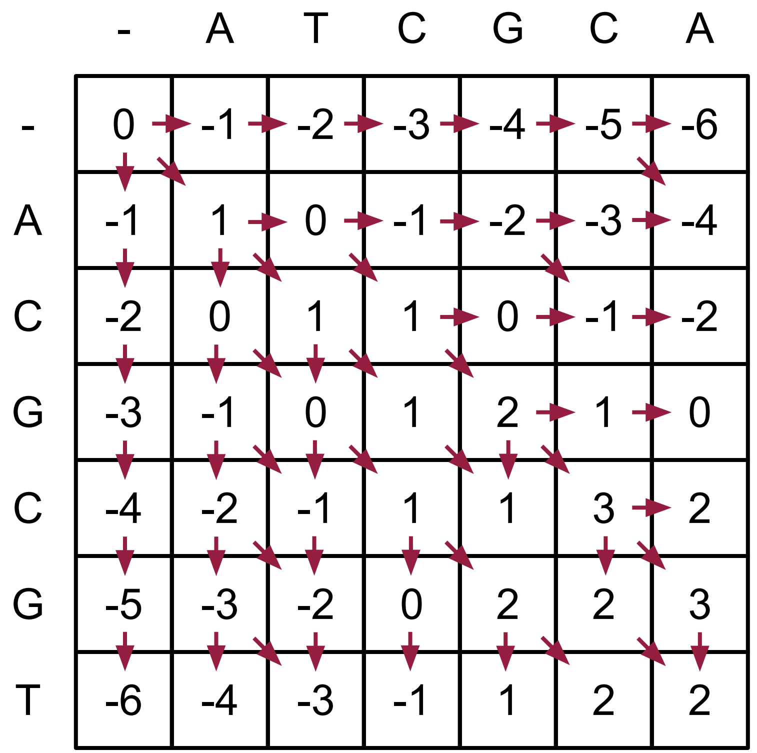
Therefore, this cell should be

- We will keep doing this until we fill the entire matrix.

$$
\begin{array}{c|c|c|c|c|c|c|c|}
 & - & A & T & C & G & C & A \\
\hline
- & 0 & -1 & -2 & -3 & -4 & -5 & -6 \\
\hline
A & -1 & 1 & \rightarrow \boxed{0} & & & & \\
\hline
C & -2 & & & & & & \\
\hline
G & -3 & & & & & & \\
\hline
C & -4 & & & & & & \\
\hline
G & -5 & & & & & & \\
\hline
T & -6 & & & & & & \\
\hline
\end{array}
$$

Fill the entire matrix with scores and directions

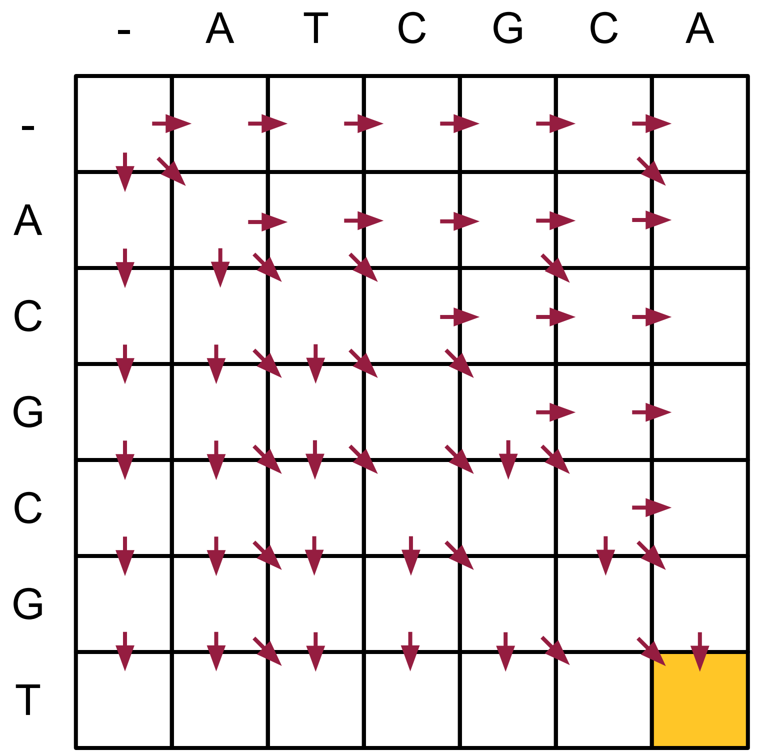
- We will keep doing this until we fill the entire matrix.
    - Note: For each cell, there could be one or two or three directions that achieve the same maximum score.



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

Now, we will determine which path gives the maximum overall score

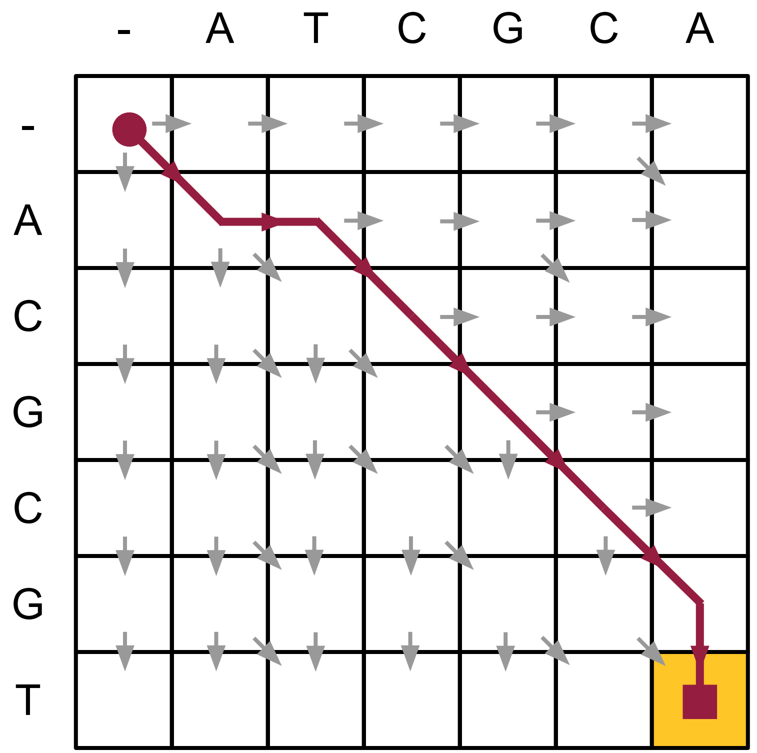
- Starting from the bottom right cell.



Trace back through arrows until reaching top left

- This is the path that represents the best alignment!
- In this case, the best alignment is:

```
A - C G C G T
|   | | |
A T C G C A -
```

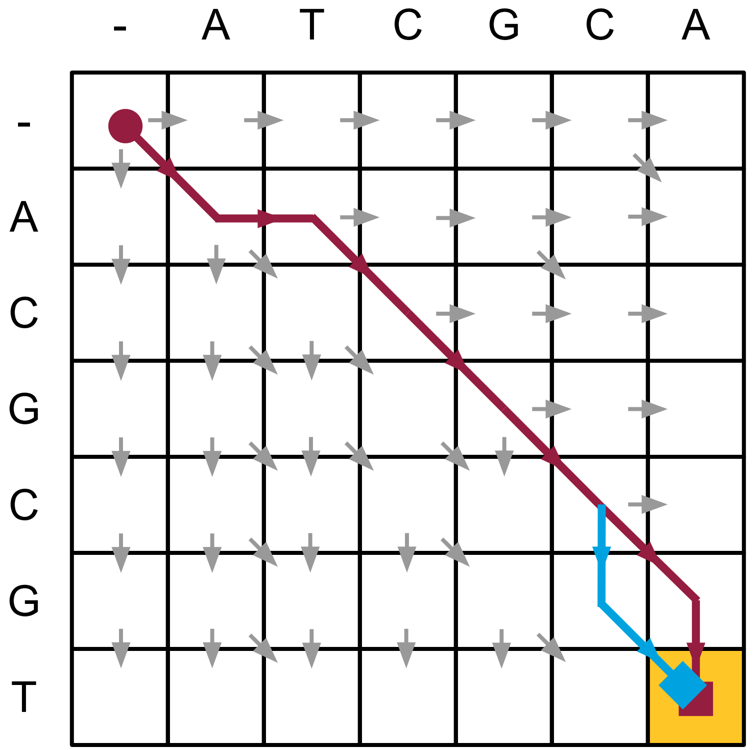


There could be more than one best alignment (path)

- They have the same alignment score.

```
A - C G C G T
|   | | |
A T C G C A -
```

```
A - C G C G T
|   | | |
A T C G C - A
```



It is practice time

It is practice time

- Two sequences

Seq 1

ATCG

Seq 2

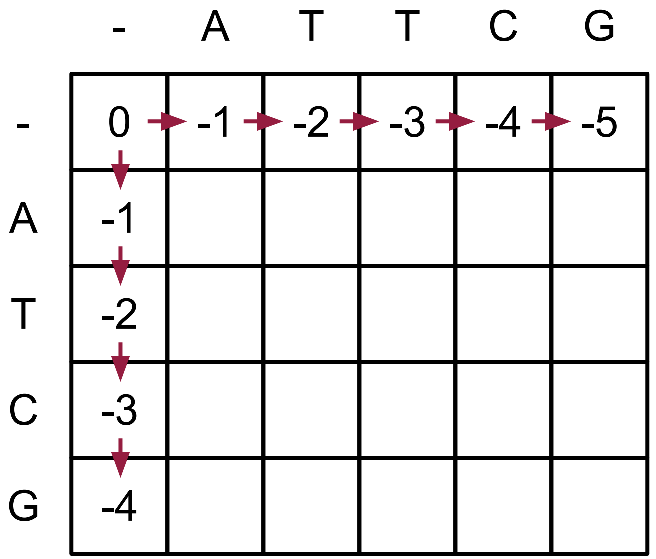
ATTCG

- Scoring system

Scoring

Match	1
Mismatch	0
Gap	-1

- Alignment matrix



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

But I can’t draw arrows in Excel!

- We need to find an ~~Excel~~ computer-friendly way to handle the matrix.
- How about we use two matrices, one for scores and the other for directions?

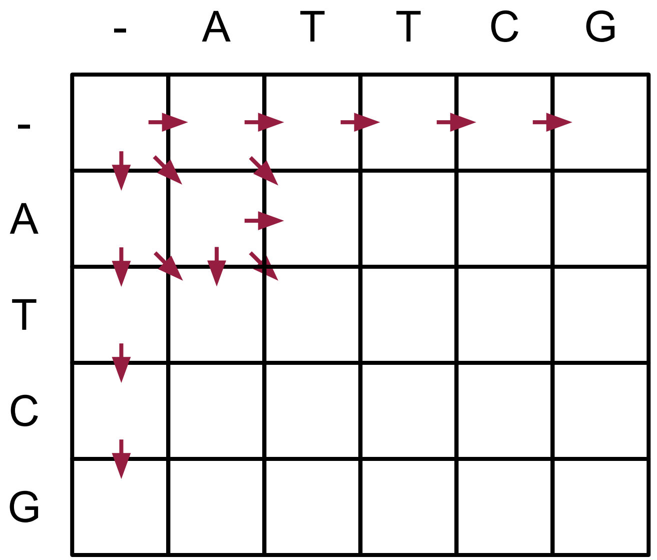
	-	A	T	T	C	G
-	0	-1	-2	-3	-4	-5
A	-1					
T	-2					
C	-3					
G	-4					

	-	A	T	T	C	G
-		→	→	→	→	→
A	↓					
T	↓					
C	↓					
G	↓					

Convert arrows into direction codes

In each cell, use letter code(s) to represent the **source** cell(s):

- **L** (left to right), **U** (upper to lower), **D** (diagonal) (upper-left to lower-right)



	-	A	T	T	C	G
-		L	L	L	L	L
A	U	D	LD			
T	U	UD	D			
C	U					
G	U					

We can effectively work on the two matrices

Score matrix

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

Traceback matrix

	-	A	T	T	C	G
-		L	L	L	L	L
A	U					
T	U					
C	U					
G	U					

- Now, let’s get rolling!

Here is the outcome

Score matrix

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

Traceback matrix

	-	A	T	T	C	G
-		L	L	L	L	L
A	U	D	L	L	L	L
T	U	U	D	LD	L	L
C	U	U	U	D	D	L
G	U	U	U	UD	D	D

So the best alignment is (are) --

- Did you get these results?

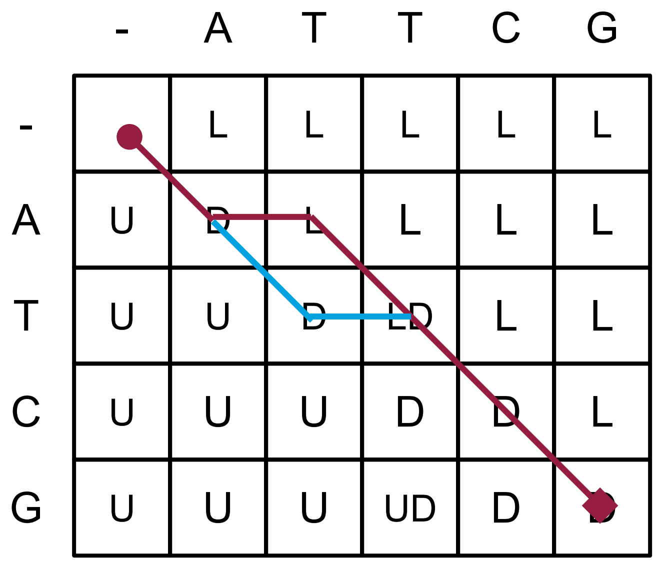
Best 1

```
A T - C G
A T C C G
```

Best 2

```
A - T C G
A T C C G
```

Traceback matrix



	-	A	T	T	C	G
-		L	L	L	L	L
A	U	D	L	L	L	L
T	U	U	D	LD	L	L
C	U	U	U	D	D	L
G	U	U	U	UD	D	D

What you just did is called the --

Needleman-Wunsch algorithm

- The **inventors**: Saul B. Needleman and Christian D. Wunsch (1970)
- It is an **algorithm**, not just an equation. It aims to resolve a problem using computer

What is the N-W algorithm for?

Optimal global alignment

- **Optimal**: Find the alignment(s) that have the highest score.
- **Global alignment**: Align the entire sequences (not just parts of them).

↓

- This is called an **optimization problem** (finding the best solution)

An algorithm resolves a problem using computer code

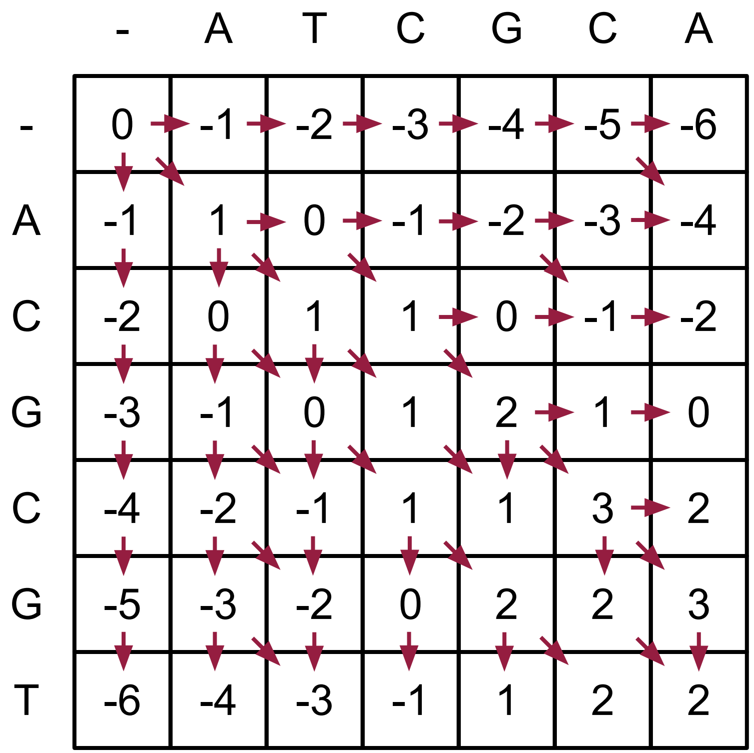
- It is a sequence of computer-implementable instructions (i.e., a program).
- It is well-defined, unambiguous and specific.
- It should be **efficient** (i.e., problem can be resolved in a reasonable amount of time using a reasonable amount of memory space)
- As compared to a mathematical equation.

Why the N-W algorithm has to be so complicated?

- Or if it's really complicated if you ask a computer?

Y	Y	Y	Y	Y	Y	N
Y	Y	N	Y	Y	Y	Y

VS



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

The naive method is prohibitively slooooooow

- Theoretically, we can generate ALL possible alignments, and determine the highest scored one, but...
- For each alignment site, there are three options: YY, YN, NY.
- So for an alignment of length n, there are roughly 3^n combinations, which is **exponential**.

Y	Y	Y	Y	Y	Y	N
Y	Y	N	Y	Y	Y	Y

- **What if we have 1000 sites?**

PS: the actual total number of possible alignments of two sequences of n and m bases is $(m+n)!/m!*n!$

Whereas the N-W algorithm only needs...

- There are a total of $n \times m$ cells to calculate.
- Which is **quadratic** (meaning "square")

m

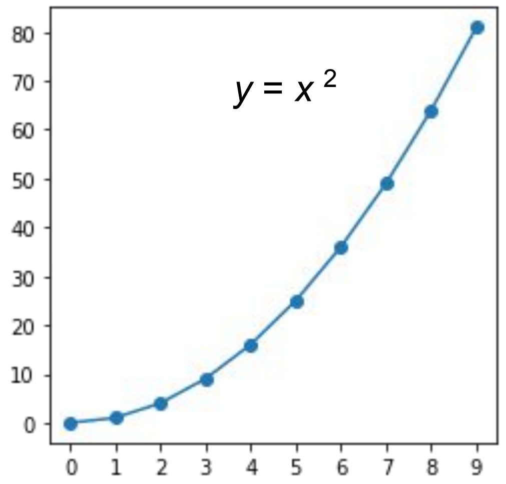
	-	A	T	C	G	C	A
-	0	-1	-2	-3	-4	-5	-6
A	-1						
C	-2						
G	-3						
C	-4						
G	-5						
T	-6						

n

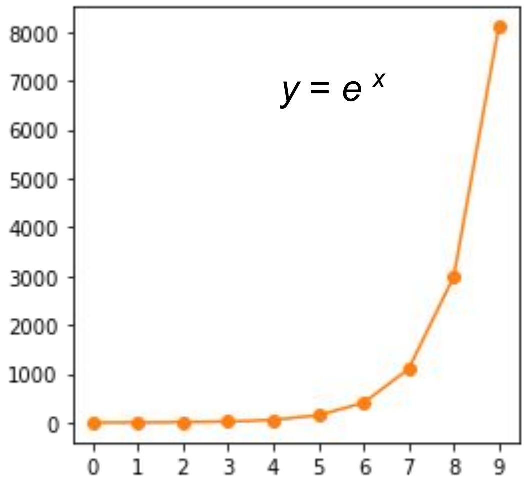
A quadratic function grows slowly compared to an exponential one

- Therefore, a quadratic algorithm is usually much cheaper (i.e., more efficient) than an exponential algorithm, when the input size is beyond 1-2 digits.

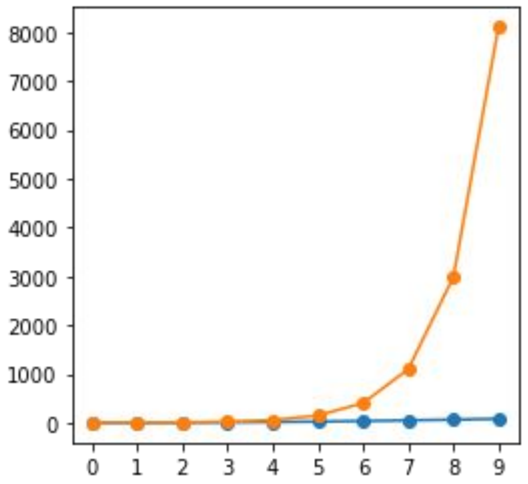
Quadratic



Exponential



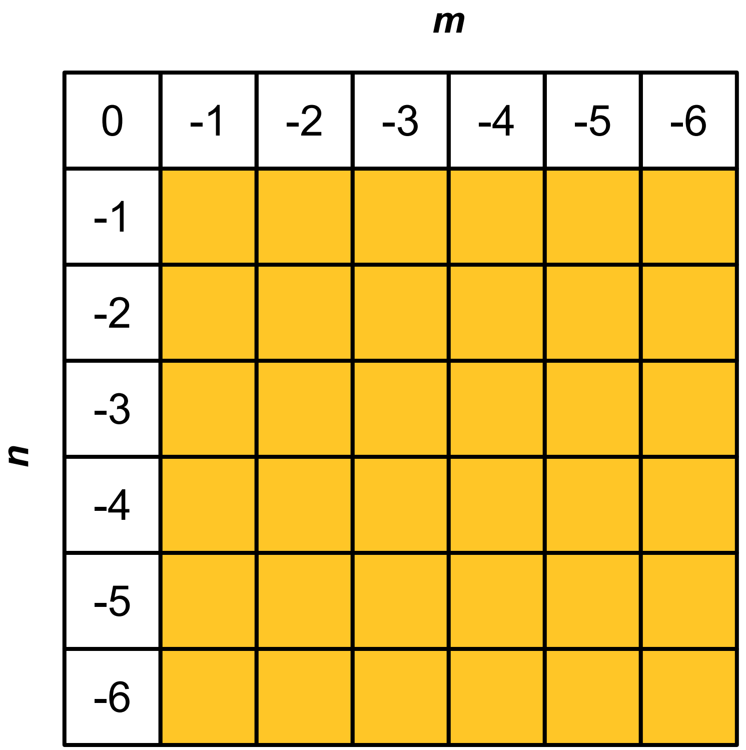
Both



Computational complexity: the big O notion

The N-W algorithm has:

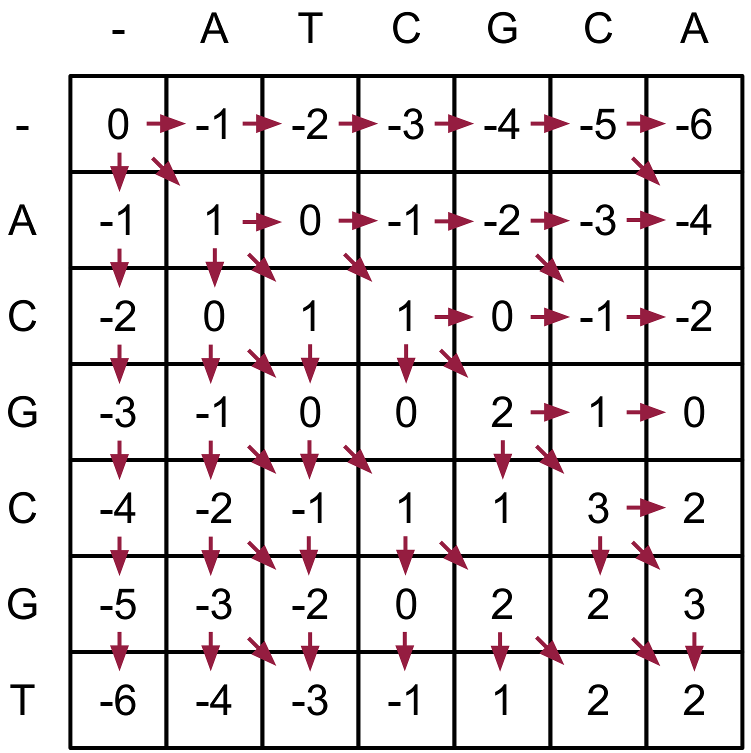
- **Time complexity**: $O(nm)$
  - Computer runtime to complete the calculation.
- **Space complexity**: $O(nm)$
  - Computer memory needed during the calculation.



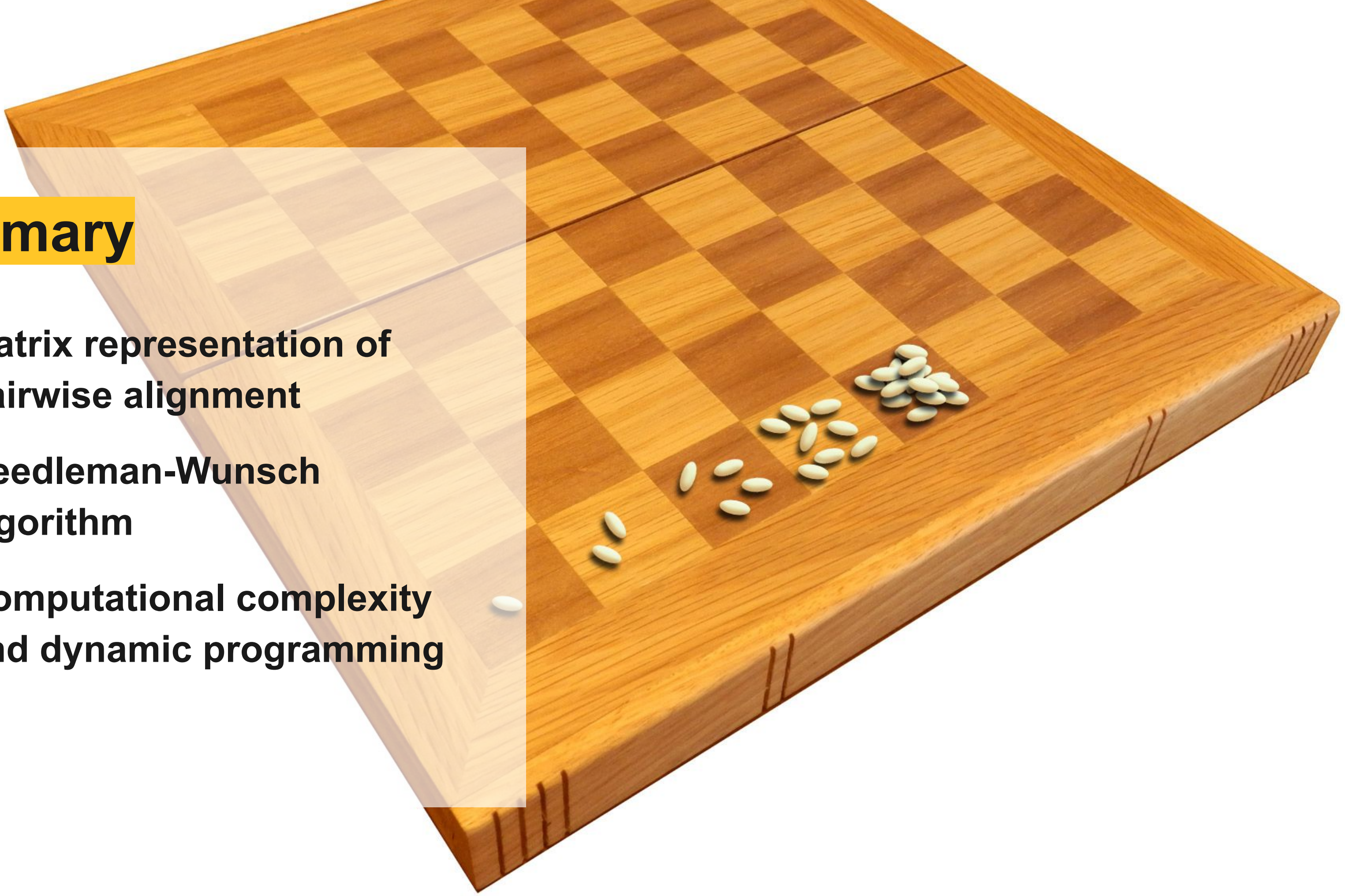
0	-1	-2	-3	-4	-5	-6
-1						
-2						
-3						
-4						
-5						
-6						

Why is the N-W algorithm so good?

- Because it breaks down the whole problem into multiple related, simpler sub-problems.
  - Here sub-alignments (cells).
- Therefore, we only need to find the optimal solution for each sub-problem.
- Then we synchronize the sub-solutions to get the global optimal solution.
- This strategy is called **dynamic programming**.



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



Summary

- Matrix representation of pairwise alignment
- Needleman-Wunsch algorithm
- Computational complexity and dynamic programming

```
.widget-area-sidebar [illegible]
.widget-area-sidebar [illegible]
.widget-area-sidebar [illegible]
.widget-area-sidebar [illegible]
.widget-area-sidebar [illegible]
.widget-area-sidebar [illegible]
	font-size: 13px;
}


/* =Menu
--------------------------------------------

#access {
	display: inline-block;
	height: 69px;
	float: right;
	margin: 11px 28px 0px 0px;
	max-width: 800px;
}

#access ul {
	font-size: 13px;
	list-style: none;
	margin: 0 0 0 -0.8125em;
	padding-left: 0;
	z-index: 99999;
	text-align: right;
}

#access li {
	display: inline-block;
	text-align: left;
```

Next class

Python implementation of the Needleman-Wunsch global alignment algorithm