Outline

• **Review: sequence alignment in sub-quadratic time for unrestricted Scoring Schemes.**
• Discuss dynamic programming special properties: convexity/concavity.
• SMAWK algorithm for computing the row/column minima/maxima of a Totally Monotone n x m matrix in O(n).
• Four Russians algorithm for sub-quadratic sequence alignment under discrete scoring schemes.
Alignment Graph

For $i, j > 0$:

$$F(i, j) = \min \begin{cases} 
F(i - 1, j) + 1, \\
F(i, j - 1) + 1, \\
F(i - 1, j - 1) + w(i, j)
\end{cases}$$
Computing the Optimal Global Alignment Value

Classical Dynamic Programming: $O(n^2)$
The O(n^2) time, Classical Dynamic Programming Algorithm

The Alignment Graph

Can the quadratic complexity of the optimal alignment value computation be reduced without relaxing the problem?
Is it Possible to Align Sequences in Subquadratic Time?

• Dynamic Programming takes $O(n^2)$ for global alignment
• Can we do better? $O(h \frac{n^2}{\log n})$, $h \leq 1$.

Techniques:
(1) **Compress** the sequences.
(2) Utilize the **Total Monotonicity** of DIST.
$O(n^2)$ vertices

$O(h \frac{n}{\log n})$ rows of $n$ vertices + $O(h \frac{n}{\log n})$ columns of $n$ vertices
Standard, single-cell DP

\[ O = \max(I_x + \text{edge}[I_x, O]) \]

\[ x = 1 \]

New, extended-cell DP

\[ O_4 = \max(I_x + \text{DIST}[x, 3]) \]

\[ x = 0 \]
Computing the score for Output Border Vertex $O_4$

$$O_4 = \max_{x=0}^6 (I_x + \text{DIST}[x,4])$$
### Input I \ DIST Matrix

<table>
<thead>
<tr>
<th>I1 = 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2 = 2</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>I3 = 3</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-3</td>
</tr>
<tr>
<td>I4 = 2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>I5 = 1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>I6 = 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Output vector \( O \)

\[
OUT[x,j] = I_x + DIST[x,j]
\]

- How to compute the column maxima of \( OUT \) in \( O(t) \) time?
  
  (Utilize the Total Monotonicity Property of \( OUT \)).

- How to obtain the \( DIST \) for \( G \) in \( O(t) \) time?
  
  (Take advantage of the incremental nature of LZ78 parsing).

### The Main Challenges

Output vector \( O \)

\[
1 3 3 4 2 3
\]
Accessing a Prefix Block in Constant time.
### The Main Challenges

How to obtain the DIST for G in $O(t)$ time? 
(Take advantage of the incremental nature of LZ78 parsing).

How to compute the column maxima of OUT in $O(t)$ time? 
(Utilize the Total Monotonicity Property of OUT).

#### Input $I$ \ DIST Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$ = 1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>$I_2$ = 2</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>□</td>
</tr>
<tr>
<td>$I_3$ = 3</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-3</td>
</tr>
<tr>
<td>$I_4$ = 2</td>
<td>□</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>$I_5$ = 1</td>
<td>□</td>
<td>□</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$I_6$ = 3</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

**OUT**[$x$, $j$] = $I_x$ + DIST[$x$, $j$]

#### Output vector $O$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>
The **Total Monotonicity** Property [Aggarwal et al 1987].

For any $a < b$ and $c < d$

$$\text{OUT}[b,c] \geq \text{OUT}[a,c] \implies \text{OUT}[b,d] \geq \text{OUT}[a,d]$$

**Computing the score for Output Border Vertex $O_4$**

$$O_4 = \max_x (I_4 + \text{DIST}[x,3])$$

**The Main Challenges**

How to compute the column maxima of OUT in $O(t)$ time? (Utilize the Total Monotonicity Property of OUT).
How does Total Monotonicity affect Column Maxima behavior?

For all $a < b$ and $c < d$, $\text{OUT}[a,c] \leq \text{OUT}[b,c] \Rightarrow \text{OUT}[a,d] \leq \text{OUT}[b,d]$

**OUT Matrix**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>-12</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>-13</td>
<td>-13</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>-14</td>
<td>-14</td>
<td>-14</td>
<td>1</td>
</tr>
</tbody>
</table>

Column maxima row indices are monotonically non-decreasing.

**SMAWK Matrix Searching** [Aggarwal et-al 87].

The $t$ column maxima of a Totally Monotone array can be computed in $O(t)$ time, by querying only $O(t)$ elements.
Outline

• Review: sequence alignment in sub-quadratic time for unrestricted Scoring Schemes.
• **Discuss dynamic programming special properties: convexity/concavity.**
• **SMAWK algorithm for computing the row/column minima/maxima of a Totally Monotone nxm matrix in O(n).**
• Four Russians algorithm for sub-quadratic sequence alignment under discrete scoring schemes
Elements of Dynamic Programming

- For dynamic programming to be applicable, an optimization problem must have:
  
  1. **Optimal substructure**
     - An optimal solution to the problem contains within it optimal solution to subproblems
  
  2. **Overlapping subproblems**
     - The space of subproblems must be small; i.e., the same subproblems are encountered over and over
Features of a Dynamic Programming Algorithm:

(1) A table (where we store optimal costs of subproblems).

(2) The entry dependency of the table (given by the recurrence relation).

When we are interested in the design of efficient algorithms for dynamic Programming, a third feature emerges:

(3) The order to fill in the table (the algorithm).

Sometimes one can use some properties of the problem at hand to change the order in which (3) is computed to obtain better algorithms.
Convexity/Concavity Properties of the DP Table $A$

(1) Monge (convex) \([\text{Monge 1781}]\)


Monge (concave)


(2) Total Monotonicity (convex)

$A[a, c] \geq A[b, c] \Rightarrow A[a,d] \geq A[b, d]$ for all $a < b$ and $c < d$

Total Monotonicity (concave)

$A[a, c] \leq A[b, c] \Rightarrow A[a,d] \leq A[b, d]$ for all $a < b$ and $c < d$
The Total Monotonicity Property

$$A[a, c] \leq A[b, c] \Rightarrow A[a, d] \leq A[b, d]$$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-∞</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

-12 0 0 2 0 0
-13 -14 -1 -1 0 0
-14 -14 -14 1 2 3

(1) matrix A is **Monotone** if, for any column d > c,
Column Maximum [d] \(\geq\) Colum Maximum [c]

(2) matrix A is **Totally Monotone** if every 2x2 submatrix is Monotone.
How does Total Monotonicity affect Column Maxima behavior?

For all $a < b$ and $c < d$, $\text{OUT}[a,c] \leq \text{OUT}[b,c] \Rightarrow \text{OUT}[a,d] \leq \text{OUT}[b,d]$

Column maxima row indices are monotonically non-decreasing.

This statement holds for every $2 \times 2$ submatrix.
How does Total Monotonicity affect Column Maxima behavior?

For all \( a < b \) and \( c < d \), \( \text{OUT}[a, c] \leq \text{OUT}[b, c] \Rightarrow \text{OUT}[a, d] \leq \text{OUT}[b, d] \)

Column maxima row indices are monotonically non-decreasing.

This statement holds for every 2x2 submatrix.

\( G[k, k] > G[k+1, k] \) and \( k < n \)
SMAWK Matrix Searching [Aggarwal et-al 87].

The \( n \) column maxima of a Totally Monotone array can be computed in \( O(n) \) time, by querying only \( O(n) \) elements.

The hearth of the algorithm is the subroutine REDUCE. It takes as input an \( n \times m \) totally monotone matrix \( A \) with \( m < n \) and returns an \( m \times m \) matrix \( G \) which is a submatrix of \( A \) such that \( G \) contains the rows of \( A \) which carry the column maxima of \( A \).
**Procedure REDUCE(A, n):**

**Input:** an $n \times m$ totally monotone matrix $A$ with $m \leq n$.

**Output:** an $m \times m$ matrix $G$ which is a submatrix of $A$ such that $G$ contains only the rows of $A$ which carry the column maxima of $A$.

\[
G \leftarrow A
\]

\[
k \leftarrow 1
\]

while the number of rows of $G$ is greater than $m$ do

begin

    case:

    a: $G[k,k] > G[k+1,k]$ and $k < n$: $k \leftarrow k+1$

    b: $G[k,k] > G[k+1,k]$ and $k = m$: delete row $k+1$

    c: $G[k,k] \leq G[k+1,k]$: delete row $k$; $k \leftarrow \max(1, k-1)$

endcase

end

return($G$);

Invariant: $G[k, 1.. k-1]$ is dead by total monotonicity.
SMAWK Matrix Searching[Aggarwal et-al 87] .
The $n$ column maxima of a Totally Monotone array can be computed in $O(n)$ time, by querying only $O(n)$ elements.

Procedure MaxColCompute(A):

$G \leftarrow \text{REDUCE}(A)$

if $(n = 1)$ then output the maximum and return

$P \leftarrow \{c2, c4, \ldots, c[n]\}$ of $G$

$\text{MaxColCompute}(P)$

from the positions (now known) of the maxima in the even columns of $G$,
find the maxima of its odd columns.
MaxColCompute(OUT = c1, c2, c3, c4, c5, c6)

\[
\begin{array}{ccccccc}
1 & 0 & -1 & -2 & -\omega & -\omega \\
1 & 1 & 0 & 1 & -1 & -\omega \\
1 & 3 & 3 & 4 & 2 & 0 \\
-12 & 0 & 0 & 2 & 0 & 0 \\
-13 & -13 & -1 & 1 & 0 & 0 \\
-14 & -14 & -14 & 1 & 2 & 3 \\
c1 & c2 & c3 & c4 & c5 & c6
\end{array}
\]
MaxColCompute(A= c1,c2,c3,c4,c5,c6)
Reduce: A is 6x6, the number of rows is not greater than m... return

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>-1</th>
<th>-2</th>
<th>-ω</th>
<th>-ω</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-ω</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>-12</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-13</td>
<td>-13</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-14</td>
<td>-14</td>
<td>-14</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\begin{align*}
\text{c1} & \quad \text{c2} & \quad \text{c3} & \quad \text{c4} & \quad \text{c5} & \quad \text{c6}
\end{align*}
\( P \leftarrow \) the even columns of \( A \)

\[
\begin{array}{ccc}
0 & -2 & -\omega \\
1 & 1 & -\omega \\
3 & 4 & 0 \\
0 & 2 & 0 \\
-13 & 1 & 0 \\
-14 & 1 & 3 \\
c2 & c4 & c6 \\
\end{array}
\]
MaxColCompute(A= c2,c4,c6)
Reduce: A is 6x3, the number of rows is greater than m.
MaxColCompute (OUT[c4])
Reduce (OUT[c4], n = 1)

MaxColCompute (n = 3)

3 4 0
0 2 0
-14 1 3

c2 c4 c6
MaxColCompute\((n=6)\)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-ω</td>
<td>-ω</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-ω</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-12</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-13</td>
<td>-13</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-14</td>
<td>-14</td>
<td>-14</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

\(c_1\) \(c_2\) \(c_3\) \(c_4\) \(c_5\) \(c_6\)
Time Analysis: Procedure Reduce takes time $O(n)$

**input:** an $n \times m$ totally monoton matrix $A$ with $m \leq n$.

**output:** an $m \times m$ matrix $G$ which is a submatrix of $A$ such that $G$ contains only the rows of $A$ which carry the column maxima of $A$.

Procedure REDUCE(A, n):

input: an $n \times m$ totally monoton matrix $A$ with $m \leq n$.

output: an $m \times m$ matrix $G$ which is a submatrix of $A$ such that $G$ contains only the rows of $A$ which carry the column maxima of $A$.

G $\leftarrow$ A
k $\leftarrow$ 1
while the number of rows of G is greater than $m$ do
begin
    case:
    a: $G[k,k] > G[k+1,k]$ and $k < n$: $k \leftarrow k+1$
    b: $G[k,k] > G[k+1,k]$ and $k = m$: delete row $k+1$
    c: $G[k,k] \leq G[k+1,k]$: delete row $k$; $k \leftarrow \max(1, k-1)$
    endcase
end
return(G);
Time Analysis.

Let $T(n,m)$ be the time taken by MaxColCompute for an $n \times m$ matrix. The call to Reduce takes time $O(n)$.

Notice that $P$ is an $m \times m/2$ totally monotone matrix, so the recursive call takes time $T(m,m/2)$.

Once the positions of the maxima in the even rows of $G$ have been found, the maximum in each odd column is restricted to the interval of maximum positions of the neighboring even columns. Thus, finding all maxima in the odd columns can be done in $O(m)$ time.

For some constants $c_1$ and $c_2$, the time complexity satisfies

$$T(n,m) \leq c_1n + c_2m + T(m, m/2)$$

which gives the solution

$$T(n,m) \leq 2(c_1+ c_2)m + c_1n = O(n) , \text{ since } m\leq n.$$