Seinfeld Warm Up

Movie Time!

In Season 9 Episode 7 “The Slicer” of the hit 90s TV show *Seinfeld*, George discovers that, years prior, he had a heated argument with his new boss, Mr. Kruger. This argument ended in George throwing Mr. Kruger’s boombox into the ocean. How did George make this discovery?

https://www.youtube.com/watch?v=pSB3HdmLcY4

Midterm

- Wednesday, March 4 in class
  - SDAC: Please schedule with SDAC for Wednesday
  - Mostly in-class with a (required) take-home portion
- Practice Midterm and Solutions on Collab
- Review Session on Panopto

Today's Keywords

- Dynamic Programming
- Longest Common Subsequence
- Seam Carving

CLRS Readings

- Chapter 15
  - Section 15.1, Log/Rod cutting, optimal substructure property
    - Note: r_i in book is called Cut() or C[] in our slides. We use their example.
  - Section 15.3, More on elements of DP, including optimal substructure property
  - Section 15.2, matrix-chain multiplication
  - Section 15.4, longest common subsequence (later example)

Log Cutting

Given a log of length n
A list (of length n) of prices P ($P[i]$ is the price of a cut of size i)
Find the best way to cut the log

Price:	1	5	8	9	10	17	17	20	24	30
Length:	1	2	3	4	5	6	7	8	9	10

Select a list of lengths $\ell_1, \ldots, \ell_k$ such that:
$\sum \ell_i = n$
to maximize $\sum P[\ell_i]$

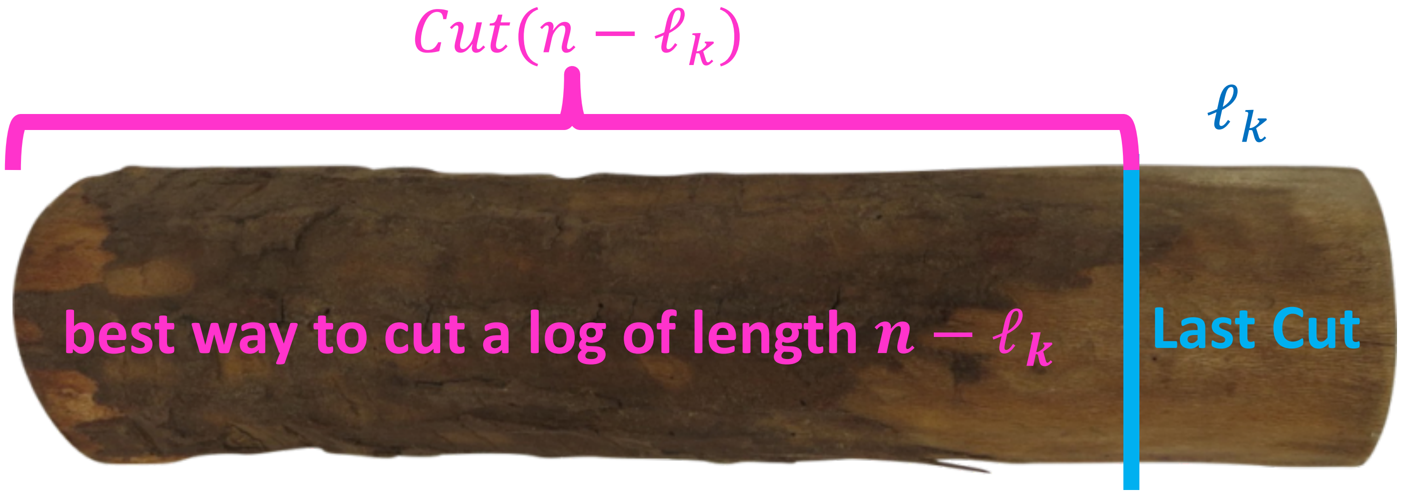
Brute Force: $O(2^n)$

1. Identify Recursive Structure

$P[i]$ = value of a cut of length i

$Cut(n)$ = value of best way to cut a log of length n

$$Cut(n) = \max \begin{cases} Cut(n-1) + P[1] \\ Cut(n-2) + P[2] \\ \dots \\ Cut(0) + P[n] \end{cases}$$



Remember the choice made

```
Initialize Memory C, Choices
Cut(n):
    C[0] = 0
    for i=1 to n:
        best = 0
        for j = 1 to i:
            if best < C[i-j] + P[j]:
                best = C[i-j] + P[j]
                Choices[i]=j
        C[i] = best
    return C[n]
```

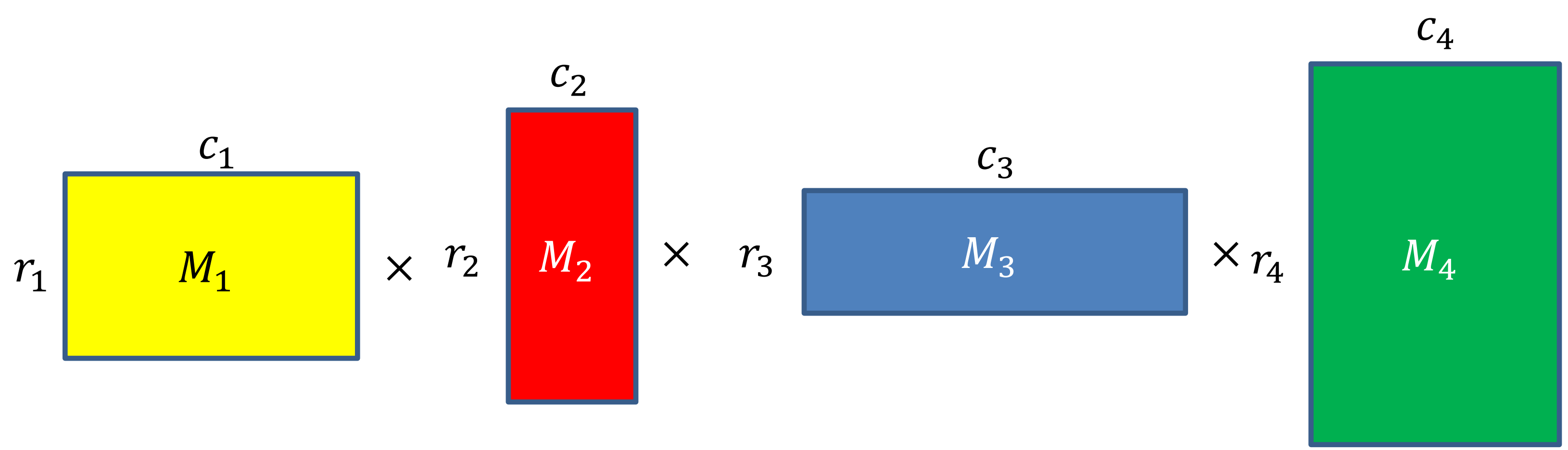
Gives the size of the last cut

Backtracking Pseudocode

```
i = n
while i > 0:
        print Choices[i]
        i = i – Choices[i]
```

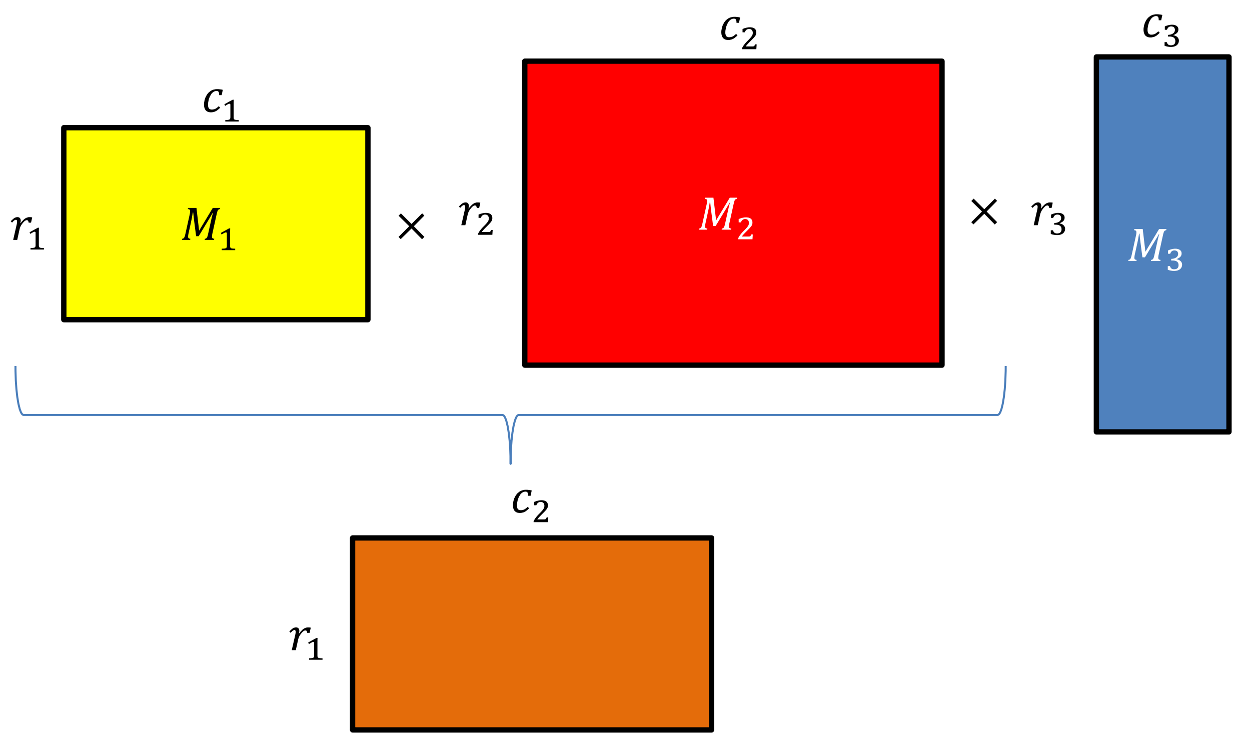
Matrix Chaining

- Given a sequence of Matrices $(M_1, \ldots, M_n)$, what is the most efficient way to multiply them?



Order Matters!

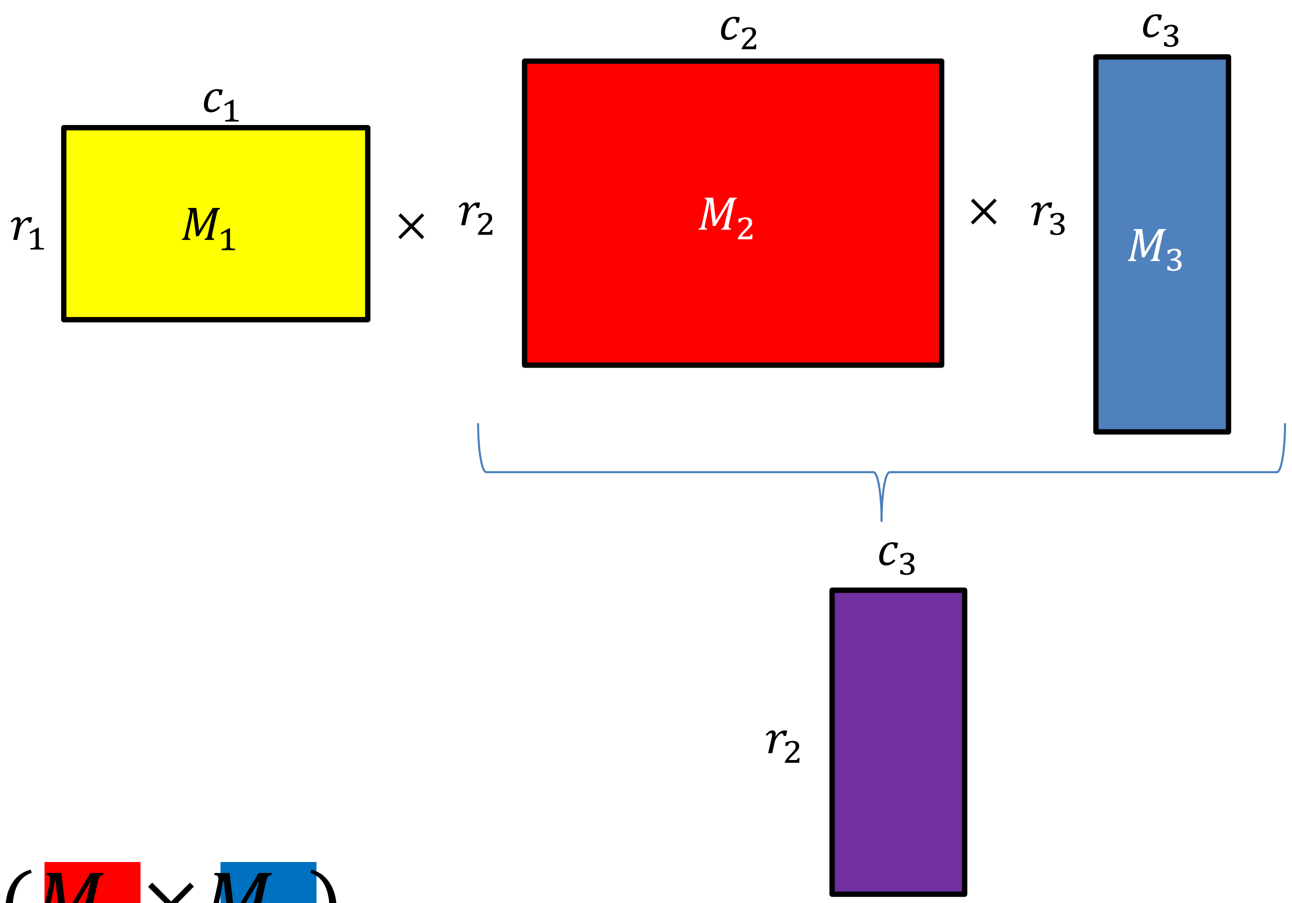
$c_1 = r_2$
$c_2 = r_3$



- $(M_1 \times M_2) \times M_3$
  - uses $(c_1 \cdot r_1 \cdot c_2) + c_2 \cdot r_1 \cdot c_3$ operations

Order Matters!

$c_1 = r_2$
$c_2 = r_3$



- $M_1 \times (M_2 \times M_3)$
  - uses $c_1 \cdot r_1 \cdot c_3 + (c_2 \cdot r_2 \cdot c_3)$ operations

2. Save Subsolutions in Memory

- In general:

$Best(i,j) =$ cheapest way to multiply together M_i through M_j

$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

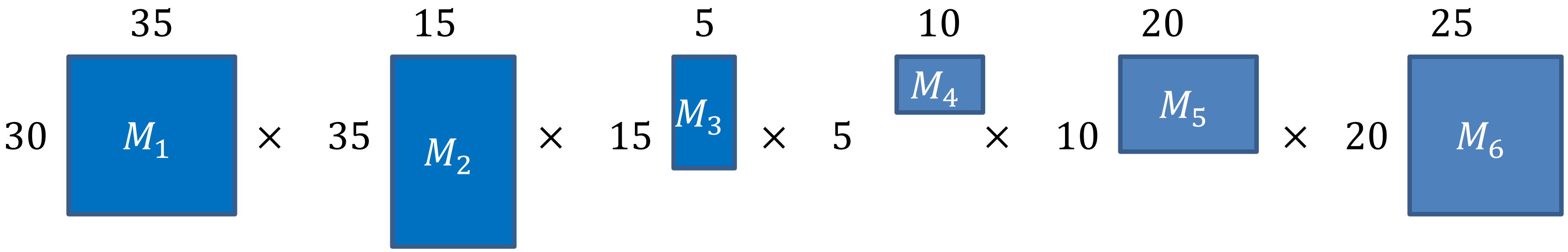
$$Best(i,i) = 0$$

Read from M[n] if present

Save to M[n]

$$Best(1,n) = \min \begin{cases} Best(2,n) + r_1 r_2 c_n \\ Best(1,2) + Best(3,n) + r_1 r_3 c_n \\ Best(1,3) + Best(4,n) + r_1 r_4 c_n \\ Best(1,4) + Best(5,n) + r_1 r_5 c_n \\ \quad \ldots \\ Best(1,n-1) + r_1 r_n c_n \end{cases}$$

3. Select a good order for solving subproblems



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$ 1	2	3	4	5	6	$i =$
0	15750	7875				1
	0	2625				2
		0	750			3
			0	1000		4
				0	5000	5
					0	6

To find $Best(i,j)$: Need all preceding terms of row i and column j

Conclusion: solve in order of diagonal

Generic Top-Down Dynamic Programming Soln

```
mem = {}
def myDPalgo(problem):
    if mem[problem] not blank:
        return mem[problem]
    if baseCase(problem):
        solution = solve(problem)
        mem[problem] = solution
        return solution
    for subproblem of problem:
        subsolutions.append(myDPalgo(subproblem))
    solution = OptimalSubstructure(subsolutions)
    mem[problem] = solution
    return solution
```

Seam Carving

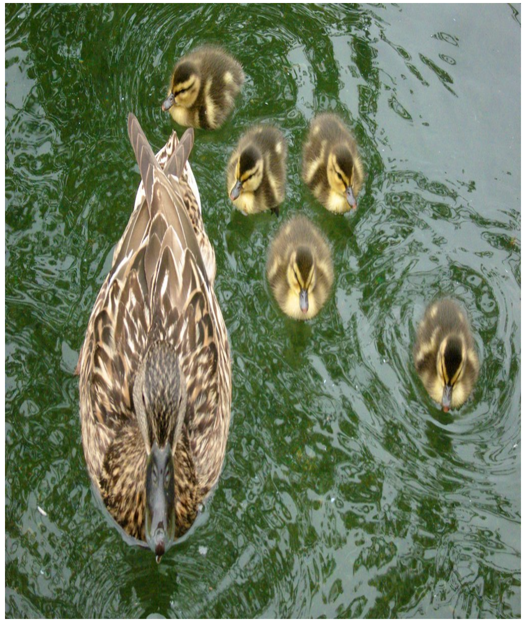
- Method for image resizing that doesn't scale/crop the image

Seam Carving

- Method for image resizing that doesn't scale/crop the image

Cropped

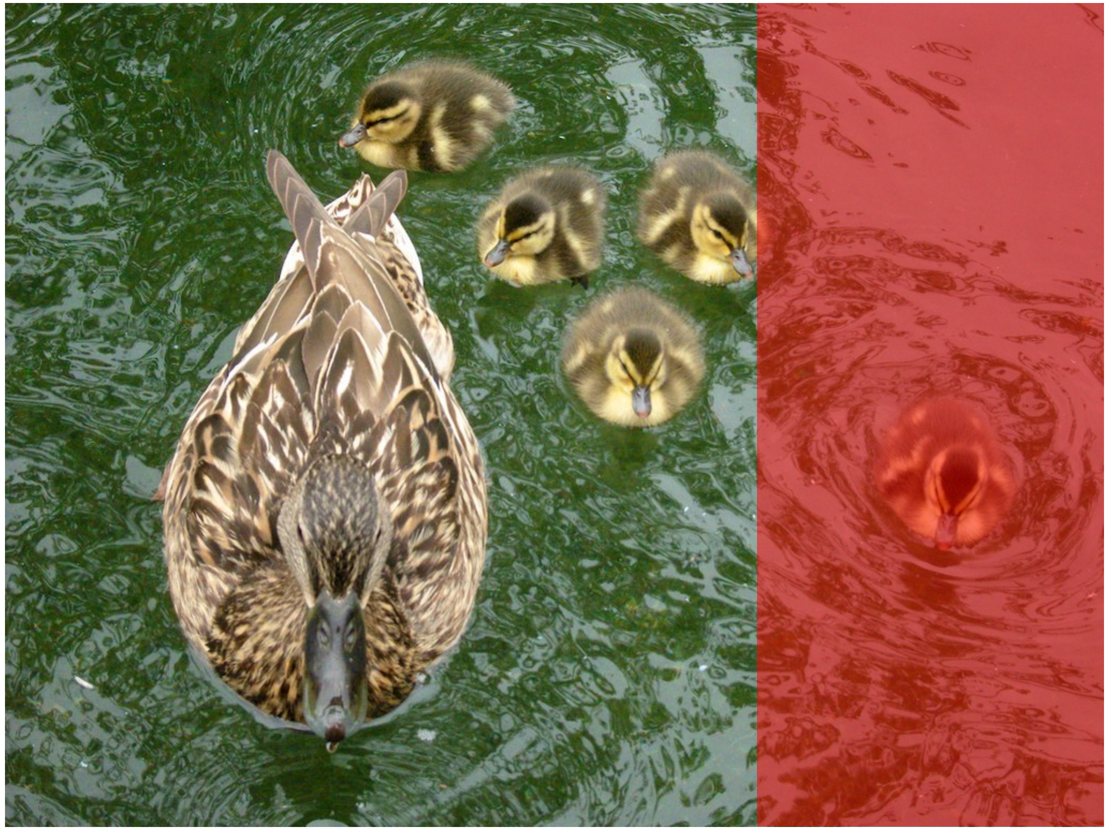
Scaled



Carved

Cropping

- Removes a "block" of pixels



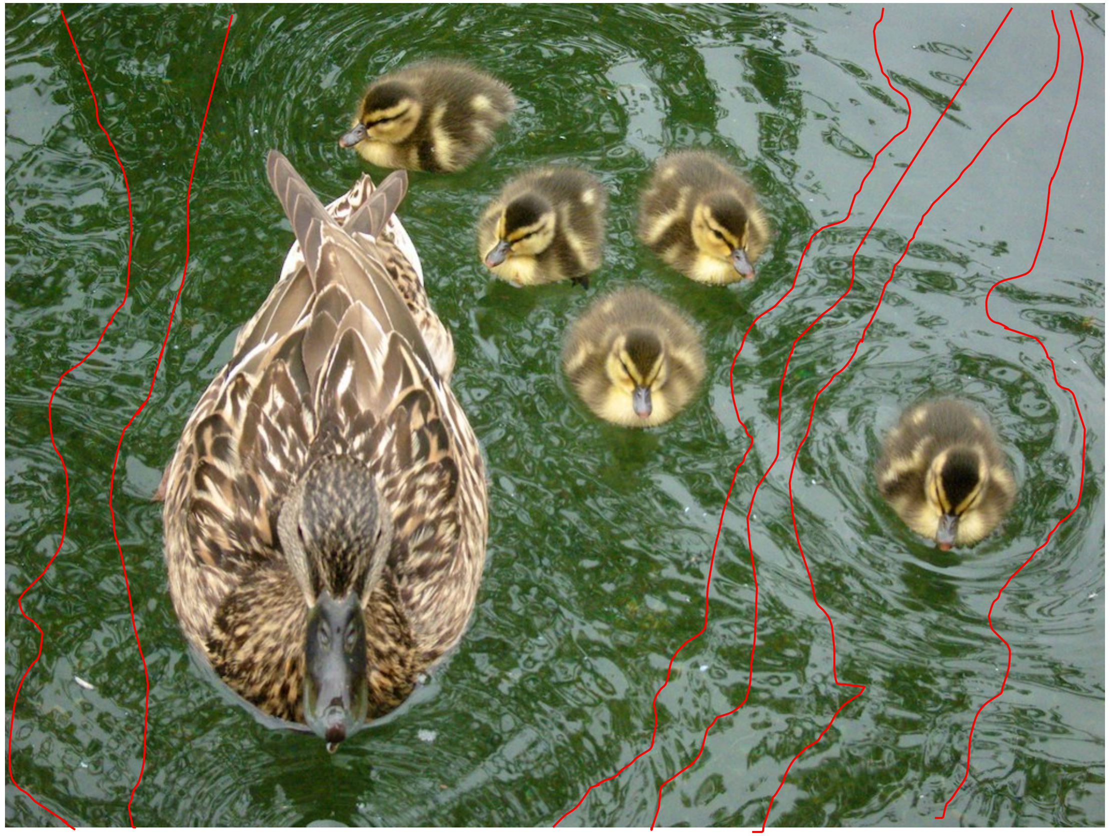
Cropped

Scaling

- Removes “stripes” of pixels

Seam Carving

- Removes "least energy seam" of pixels
- http://rsizr.com/



Carved

Energy of Pixels

Define the "interestingness" or energy of a pixel

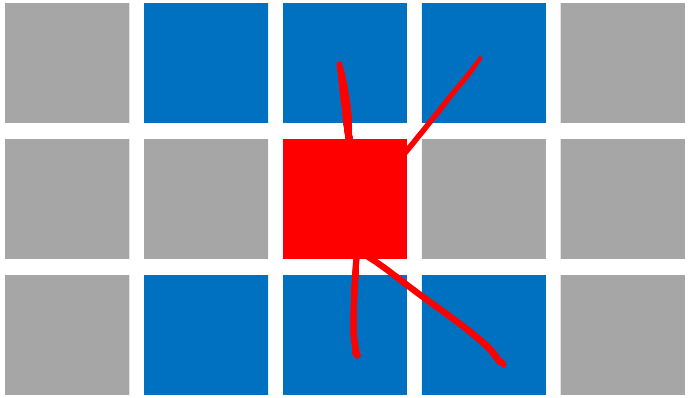
- $e(p) =$ energy of pixel p
- Many choices
  – Ex: change of gradient (how much the color of this pixel differs from its neighbors)
    - Euclidian distance from it's direct neighbors
    - Gradient of some number of surrounding pixels
    - Difference in intensity (but not color)

  – Particular choice doesn't matter, we use it as a "black box"

Seams

Seam: path of pixels from the top to the bottom of the image

- One pixel per row
- Direct neighbors: vertically or horizontally



Energy of Seam: Sum of the energies of each pixel

- $\sum_{i=1}^{n} e(p_i)$ - the sum of each pixel on the seam (one per row)

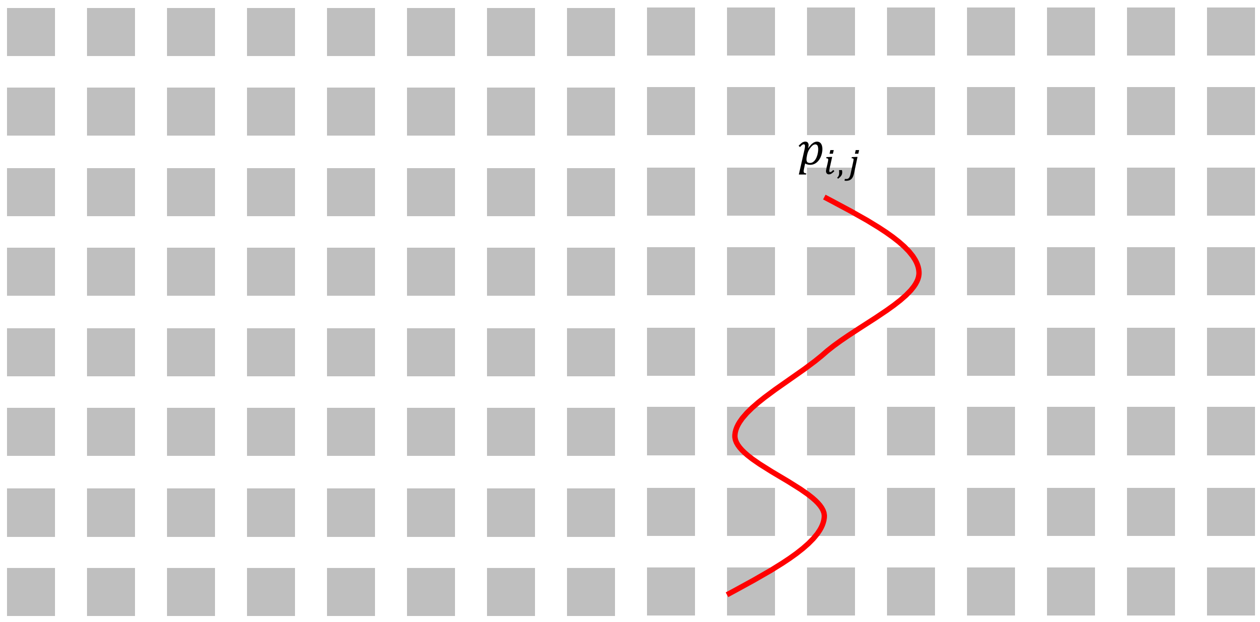
Example: Seattle Skyline

Dynamic Programming

- Requires Optimal Substructure
  - – Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the “last thing” done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

Identify Recursive Structure

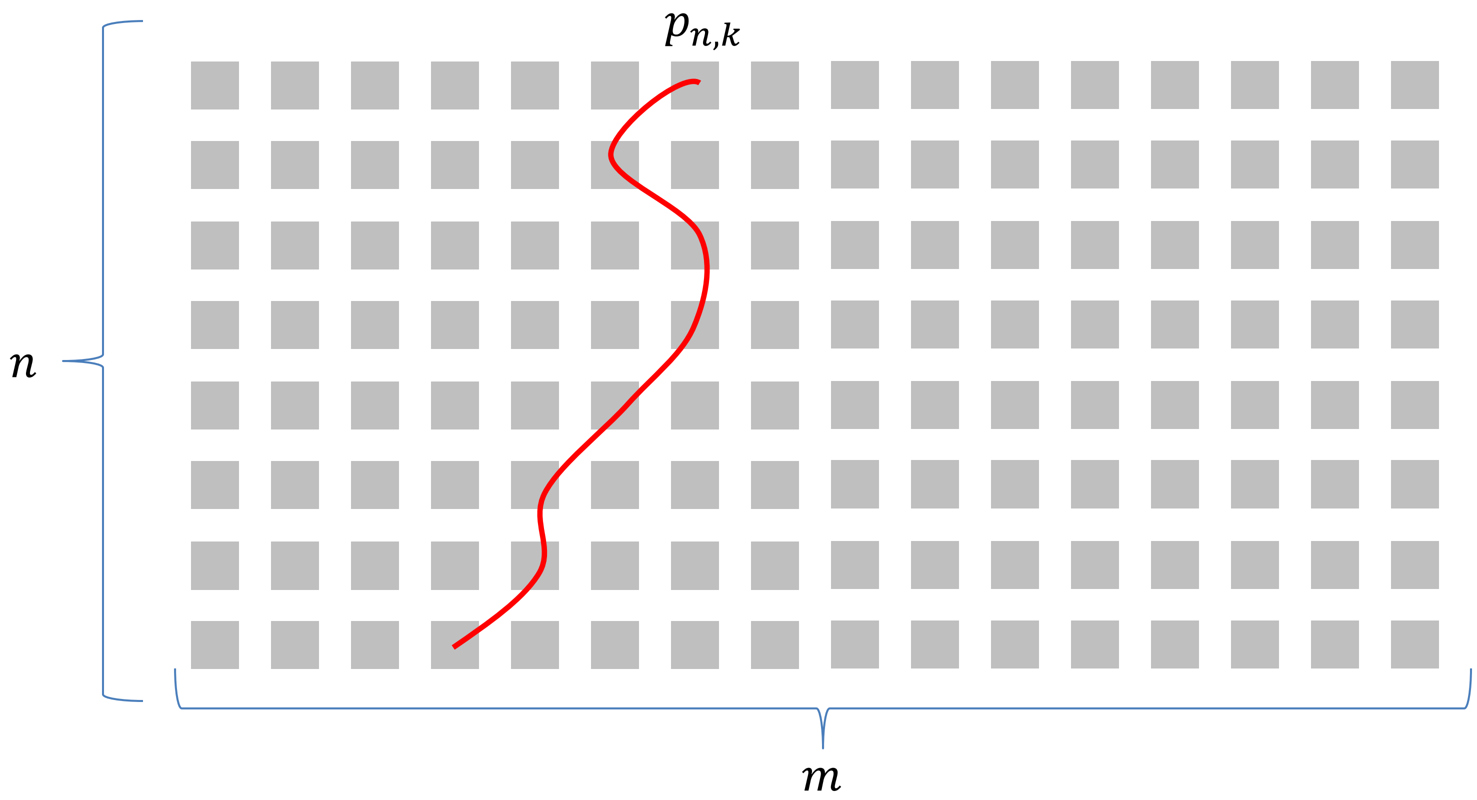
Let $S(i, j)$ = least energy seam from the bottom of the image up to pixel $p_{i,j}$



Finding the Least Energy Seam

Want the least energy seam going from bottom to top, so find and delete:

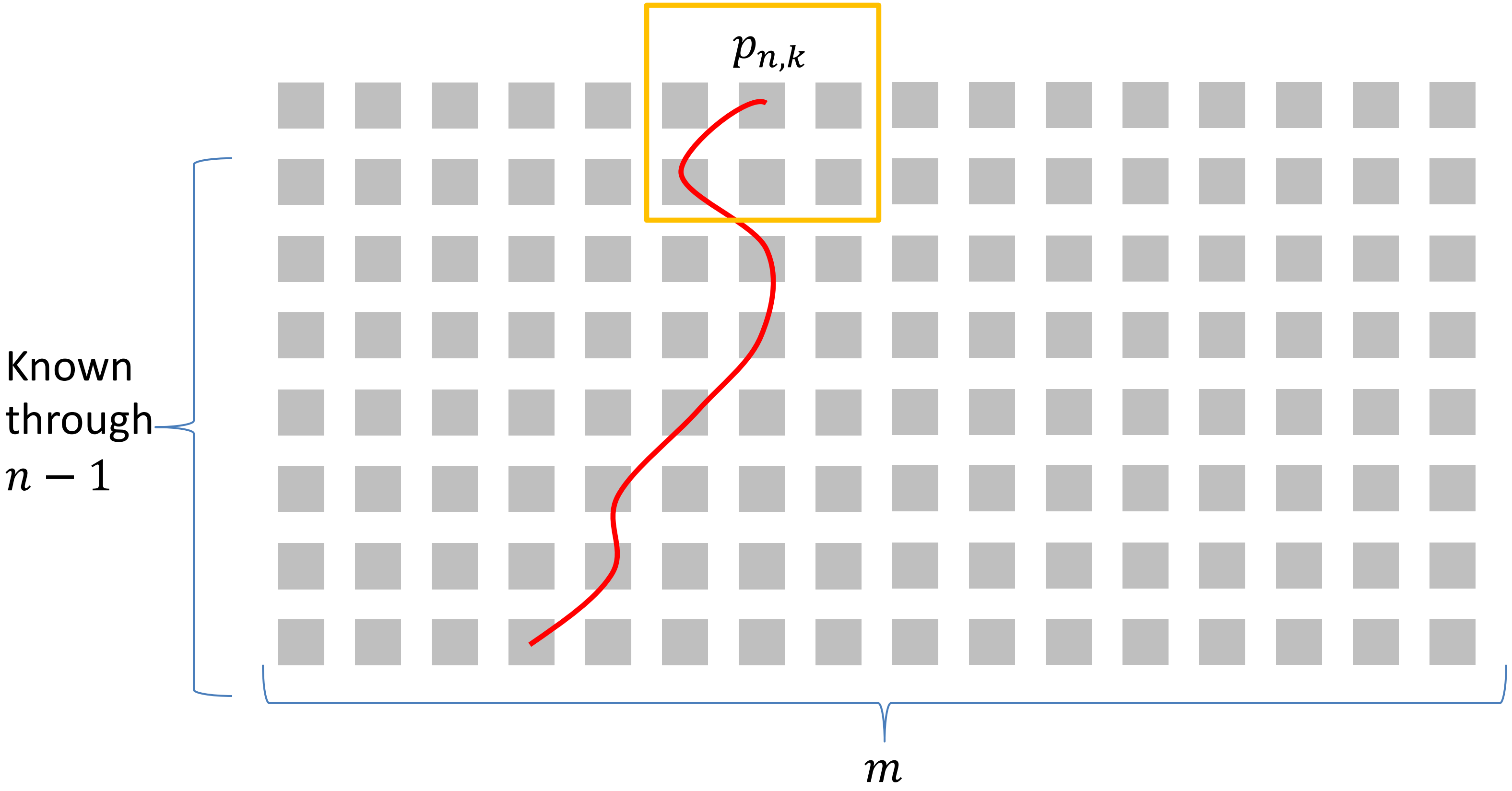
$$\min_{k=1}^{m}(S(n,k))$$



Computing $S(n, k)$

Assume we know the least energy seams for all of row $n-1$

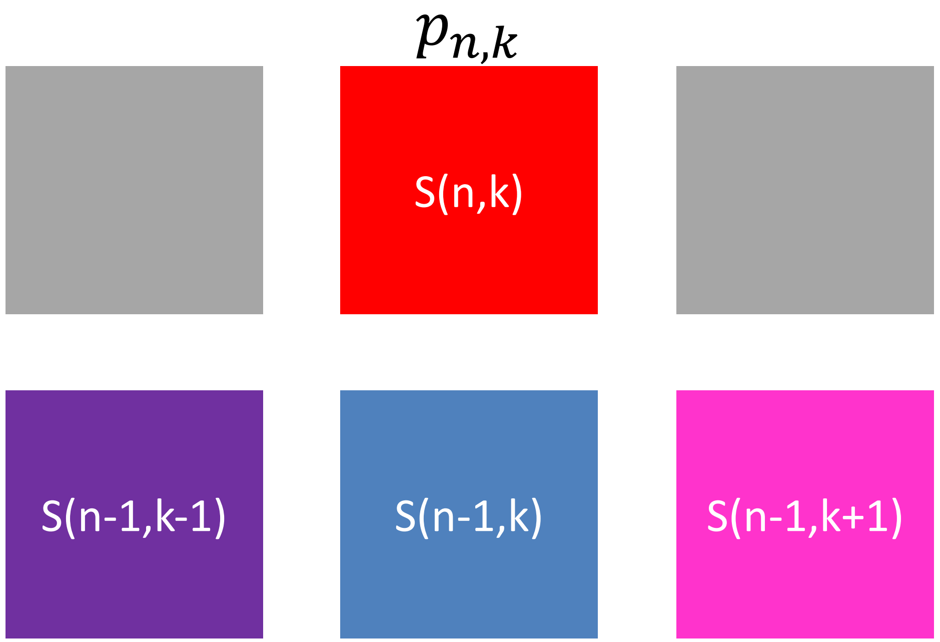
(i.e. we know $S(n-1, \ell)$ for all ℓ)



Computing $S(n,k)$

Assume we know the least energy seams for all of row $n-1$

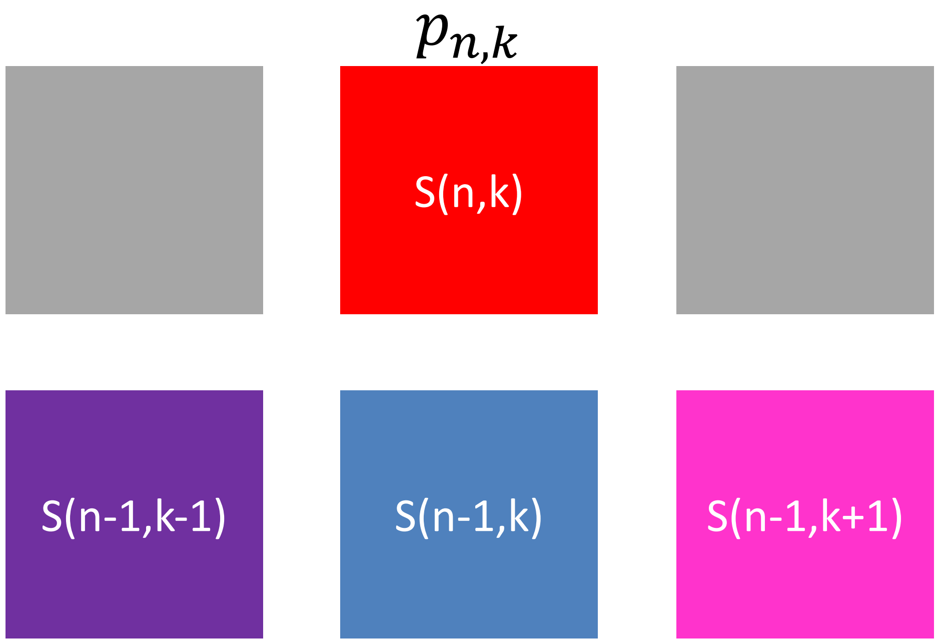
(i.e. we know $S(n-1,\ell)$ for all ℓ)



Computing $S(n,k)$

Assume we know the least energy seams for all of row $n-1$ (i.e. we know $S(n-1,\ell)$ for all ℓ)

$$S(n,k) = min \begin{cases} S(n-1,k-1) + e(p_{n,k}) \\ S(n-1,k) + e(p_{n,k}) \\ S(n-1,k+1) + e(p_{n,k}) \end{cases}$$



Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the "last thing" done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest

Computing $S(n,k)$

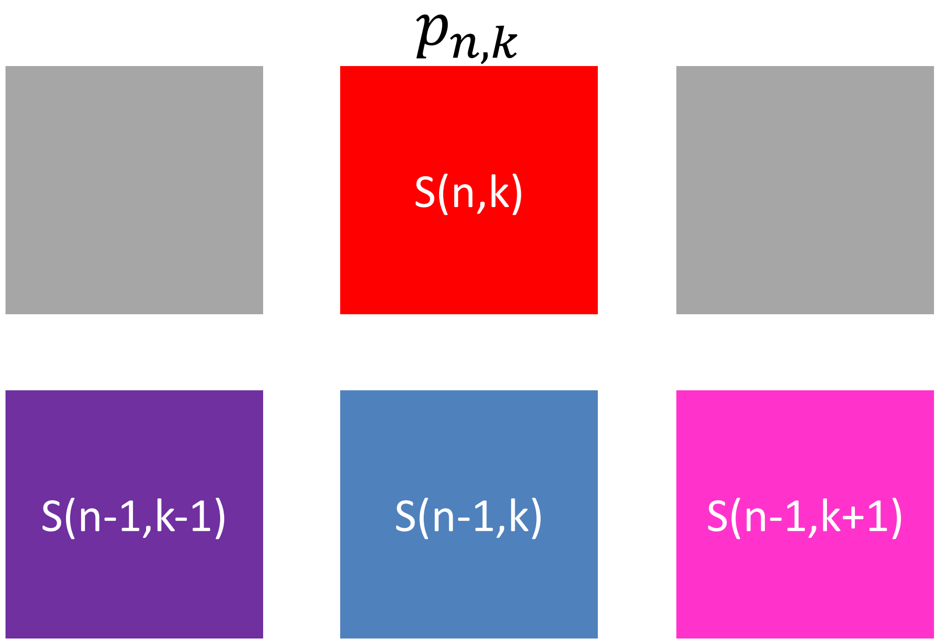
Assume we know the least energy seams for all of row $n-1$

(i.e. we know $S(n-1,\ell)$ for all ℓ)

$$S(n,k) = min\begin{cases} S(n-1,k-1) + e(p_{n,k}) \\ S(n-1,k) + e(p_{n,k}) \\ S(n-1,k+1) + e(p_{n,k}) \end{cases}$$

Store in memory!

Read from memory!



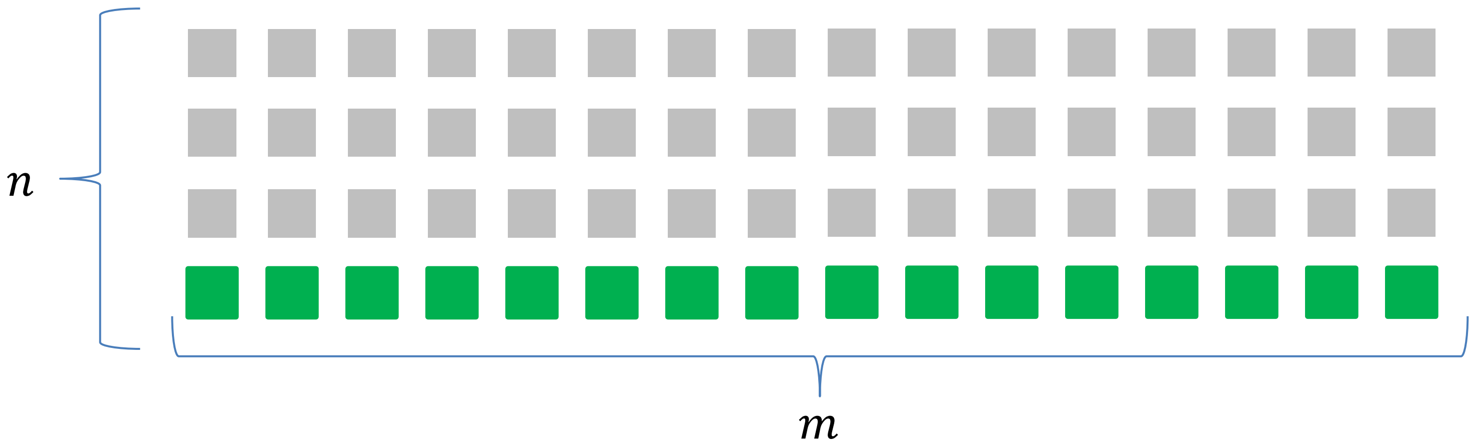
Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the "last thing" done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest

Seam Carving Solution

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel in row 1



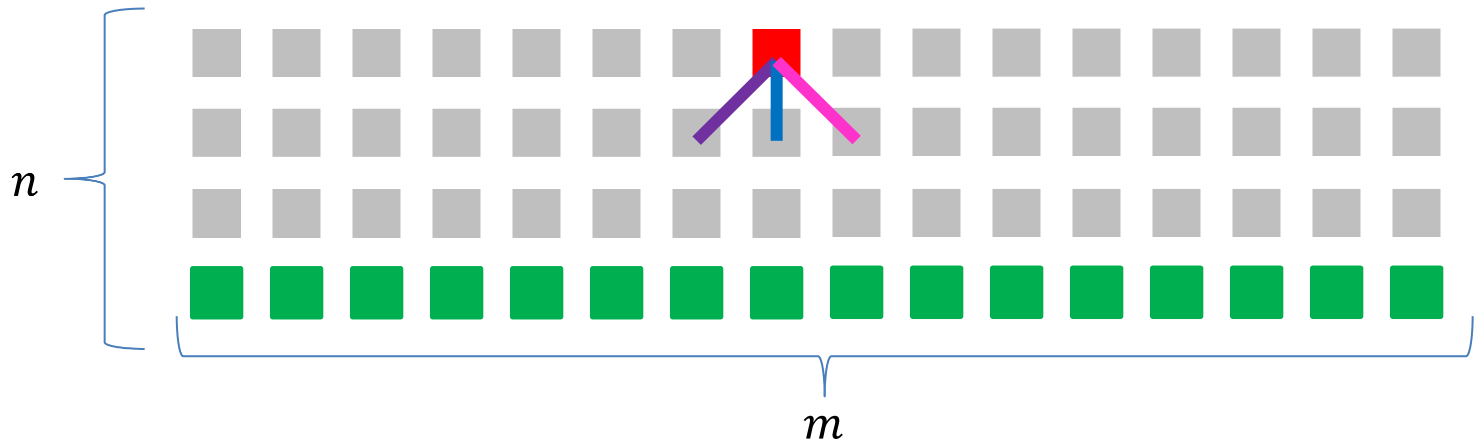
Energy of the seam initialized to the energy of that pixel

Seam Carving Solution

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel $p_{1,k}$

For $i > 2$ find $S(i,k) = \min \begin{cases} S(n-1,k-1) + e(p_{n,k}) \\ S(n-1,k) + e(p_{n,k}) \\ S(n-1,k+1) + e(p_{n,k}) \end{cases}$



Energy of the seam initialized to the energy of that pixel

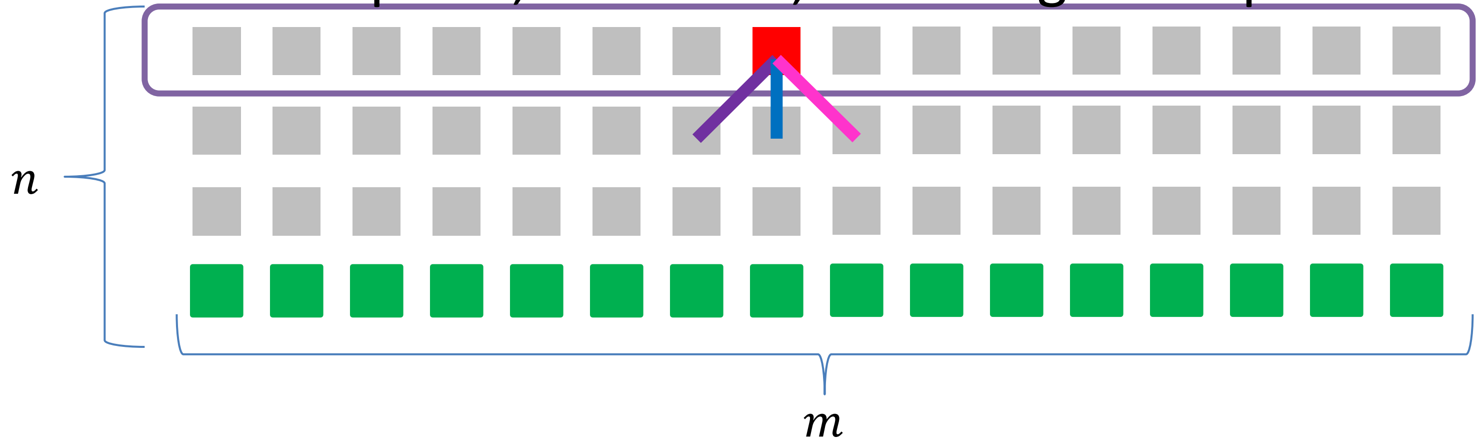
Seam Carving Solution

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel $p_{1,k}$

For $i > 2$ find $S(i,k) = \min \begin{cases} S(n-1,k-1) + e(p_{n,k}) \\ S(n-1,k) + e(p_{n,k}) \\ S(n-1,k+1) + e(p_{n,k}) \end{cases}$

Pick smallest from top row, backtrack, removing those pixels



Energy of the seam initialized to the energy of that pixel

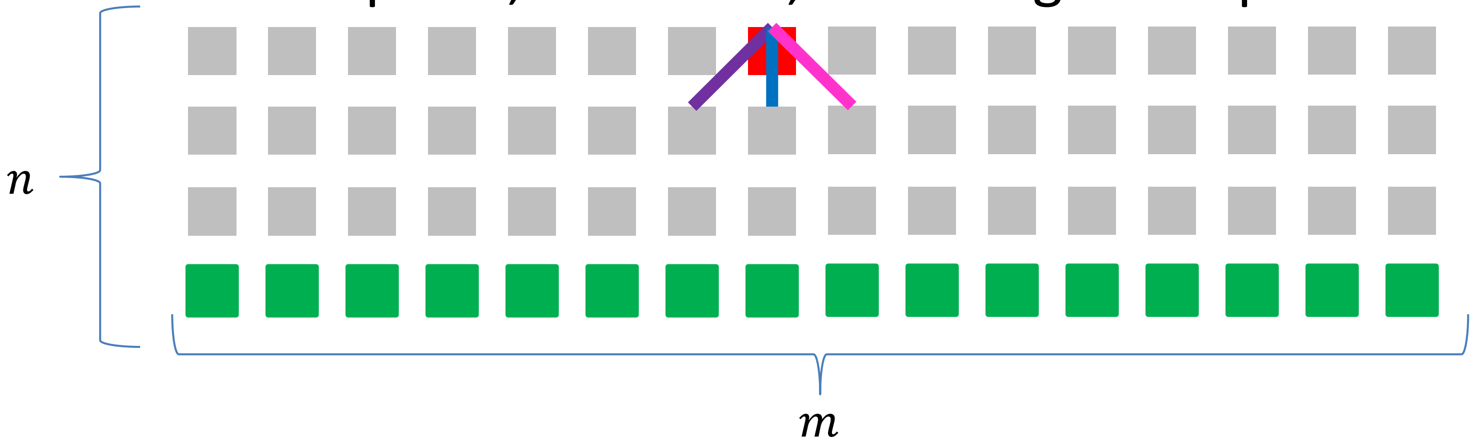
Run Time?

Start from bottom of image (row 1), solve up to top

Initialize $S(1,k) = e(p_{1,k})$ for each pixel $p_{1,k}$ $\quad\Theta(m)$

For $i \geq 2$ find $S(i,k) = \min \begin{cases} S(n-1,k-1) + e(p_{i,k}) \\ S(n-1,k) + e(p_{i,k}) \\ S(n-1,k+1) + e(p_{i,k}) \end{cases}$ $\quad\Theta(n \cdot m)$

Pick smallest from top row, backtrack, removing those pixels $\quad\Theta(n+m)$



Energy of the seam initialized to the energy of that pixel

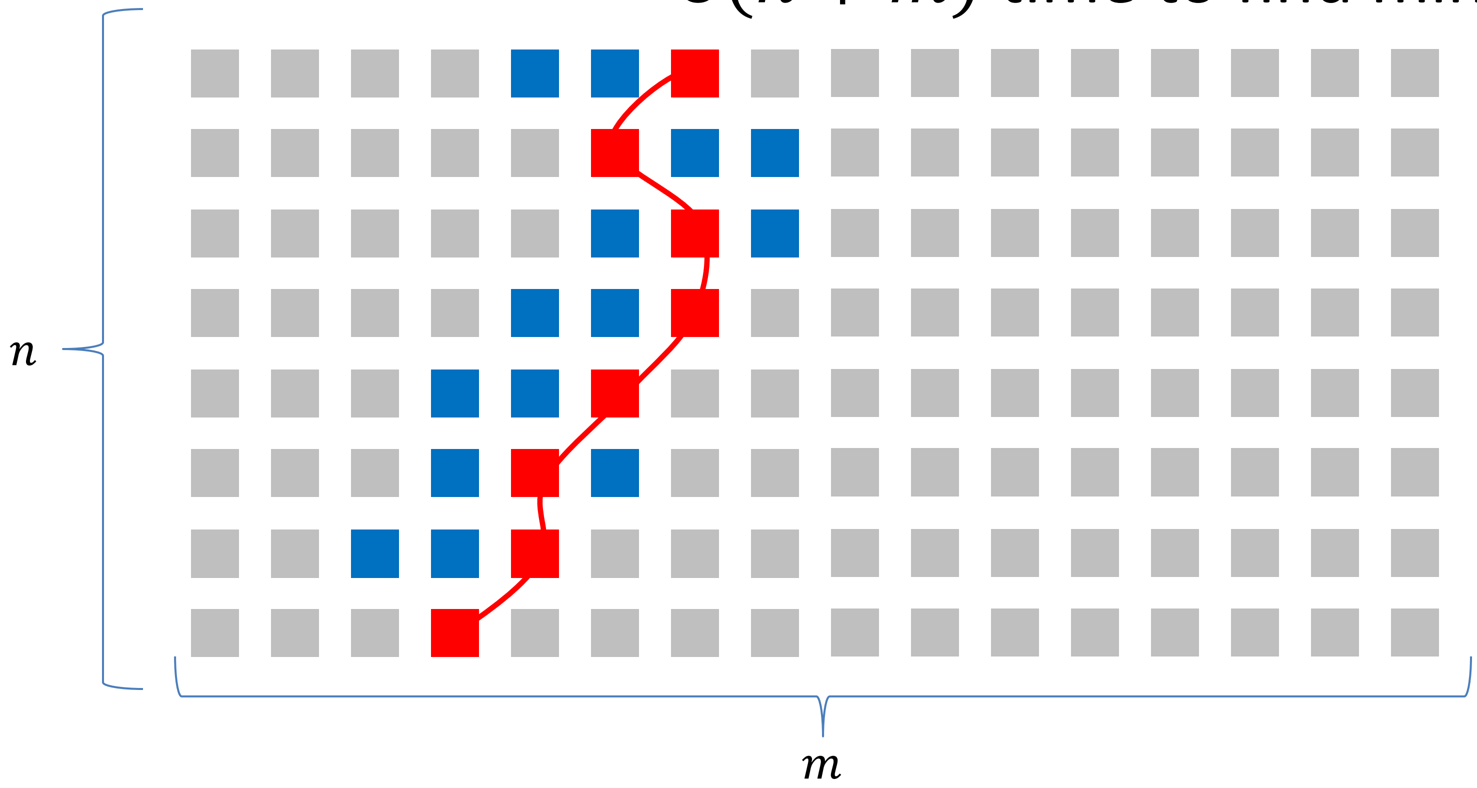
Repeated Seam Removal

Only need to update pixels dependent on the removed seam

$2n$ pixels change

$\Theta(2n)$ time to update pixels

$\Theta(n+m)$ time to find min+backtrack



Longest Common Subsequence

Given two sequences X and Y, find the length of their longest common subsequence

Example:

$X = ATCTGAT$
$Y = TGCATA$
$LCS = TCTA$

Brute force: Compare every subsequence of X with Y
$\Omega(2^n)$

Dynamic Programming

- Requires Optimal Substructure
  – Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the "last thing" done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest

1. Identify Recursive Structure

Let $LCS(i,j)$ = length of the LCS for the first i characters of X, first j character of Y

Find $LCS(i,j)$:

Case 1: $X[i] = Y[j]$

$$X = ATCTGCG\boxed{T}$$
$$Y = TGCATAT$$

$$LCS(i,j) = LCS(i-1,j-1) + 1$$

Case 2: $X[i] \neq Y[j]$

$$X = ATCTGCG\boxed{A}$$
$$Y = TGCATAT$$

$$LCS(i,j) = LCS(i,j-1)$$

$$X = ATCTGCGT$$
$$Y = TGCATAC$$

$$LCS(i,j) = LCS(i-1,j)$$

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1,j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i,j-1), LCS(i-1,j)) & \text{otherwise} \end{cases}$$

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the “last thing” done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest

1. Identify Recursive Structure

Let $LCS(i, j)$ = length of the LCS for the first i characters of X, first j character of Y

Find $LCS(i, j)$:

Case 1: $X[i] = Y[j]$

$$X = ATCTGCGT$$
$$Y = TGCATAT$$

$$LCS(i, j) = LCS(i-1, j-1) + 1$$

Case 2: $X[i] \neq Y[j]$

$$X = ATCTGCGA$$
$$Y = TGCATAT$$

$$LCS(i, j) = LCS(i, j-1)$$

$$X = ATCTGCGT$$
$$Y = TGCATAC$$

$$LCS(i, j) = LCS(i-1, j)$$

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

Read from M[i,j] if present

Save to M[i,j]

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify the recursive structure of the problem
     - What is the "last thing" done?
  2. Save the solution to each subproblem in memory
  3. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest

3. Solve in a Good Order

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$		0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
T	1	**0**							
G	2	**0**							
C	3	**0**							
A	4	**0**							
T	5	**0**							
A	6	**0**							

To fill in cell (i, j) we need cells $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$

Fill from Top->Bottom, Left->Right (with any preference)

3. Solve in a Good Order

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$		0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
T	1	**0**							
G	2	**0**							
C	3	**0**							
A	4	**0**							
T	5	**0**							
A	6	**0**							

To fill in cell (i, j) we need cells $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$

Fill from Top->Bottom, Left->Right (with any preference)

3. Solve in a Good Order

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$		0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
T	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G	2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
C	3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
A	4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
T	5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A	6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

To fill in cell (i, j) we need cells $(i-1, j-1)$, $(i-1, j), (i, j-1)$
Fill from Top->Bottom, Left->Right (with any preference)

Run Time?

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$	0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
T 1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G 2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
C 3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
A 4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
T 5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A 6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

Run Time: $\Theta(n \cdot m)$ (for $|X| = n$, $|Y| = m$)

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$	$X =$		A	T	C	$\boxed{T}$	$\boxed{G}$	$\boxed{A}$	$\boxed{T}$
		0	1	2	3	4	5	6	7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$\boxed{T}$	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
$\boxed{G}$	2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
C	3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
$\boxed{A}$	4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
$\boxed{T}$	5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A	6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

Start from bottom right,
if symbols matched, print that symbol then go diagonally
else go to largest adjacent

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$	0	A 1	$\boxed{T}$ 2	$\boxed{C}$ 3	T 4	G 5	$\boxed{A}$ 6	$\boxed{T}$ 7
0	0	0	0	0	0	0	0	0
$\boxed{T}$ 1	0	0	1	1	1	1	1	1
G 2	0	0	1	1	1	2	2	2
$\boxed{C}$ 3	0	0	1	2	2	2	2	2
$\boxed{A}$ 4	0	1	1	2	2	2	3	3
$\boxed{T}$ 5	0	1	2	2	3	3	3	4
A 6	0	1	2	2	3	3	4	4

Start from bottom right,

if symbols matched, print that symbol then go diagonally

else go to largest adjacent

Reconstructing the LCS

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

	$X =$	0	A 1	$\boxed{T}$ 2	$\boxed{C}$ 3	$\boxed{T}$ 4	G 5	$\boxed{A}$ 6	T 7
$Y =$	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
$\boxed{T}$	1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G	2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
$\boxed{C}$	3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
A	4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
$\boxed{T}$	5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
$\boxed{A}$	6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

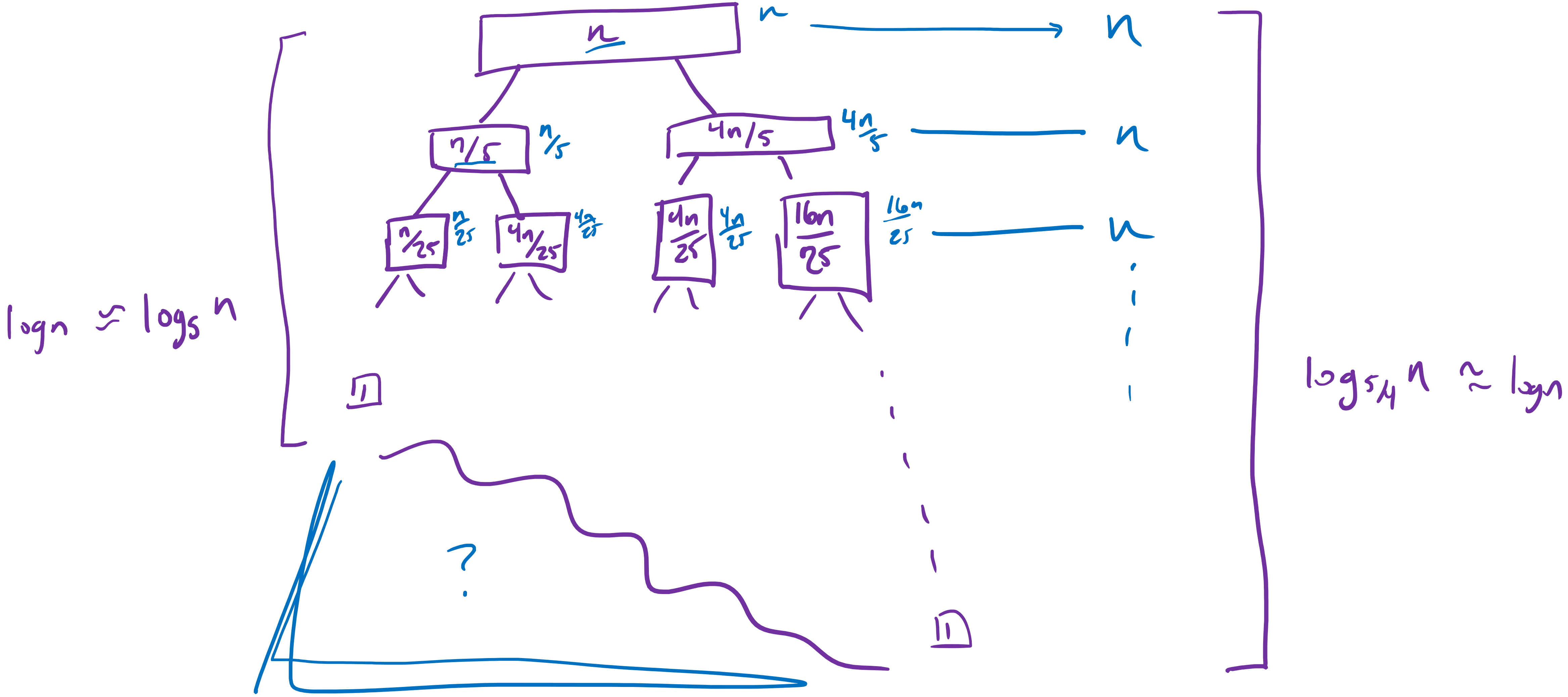
Start from bottom right,

if symbols matched, print that symbol then go diagonally

else go to largest adjacent

Midterm Review

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + O(n)$$



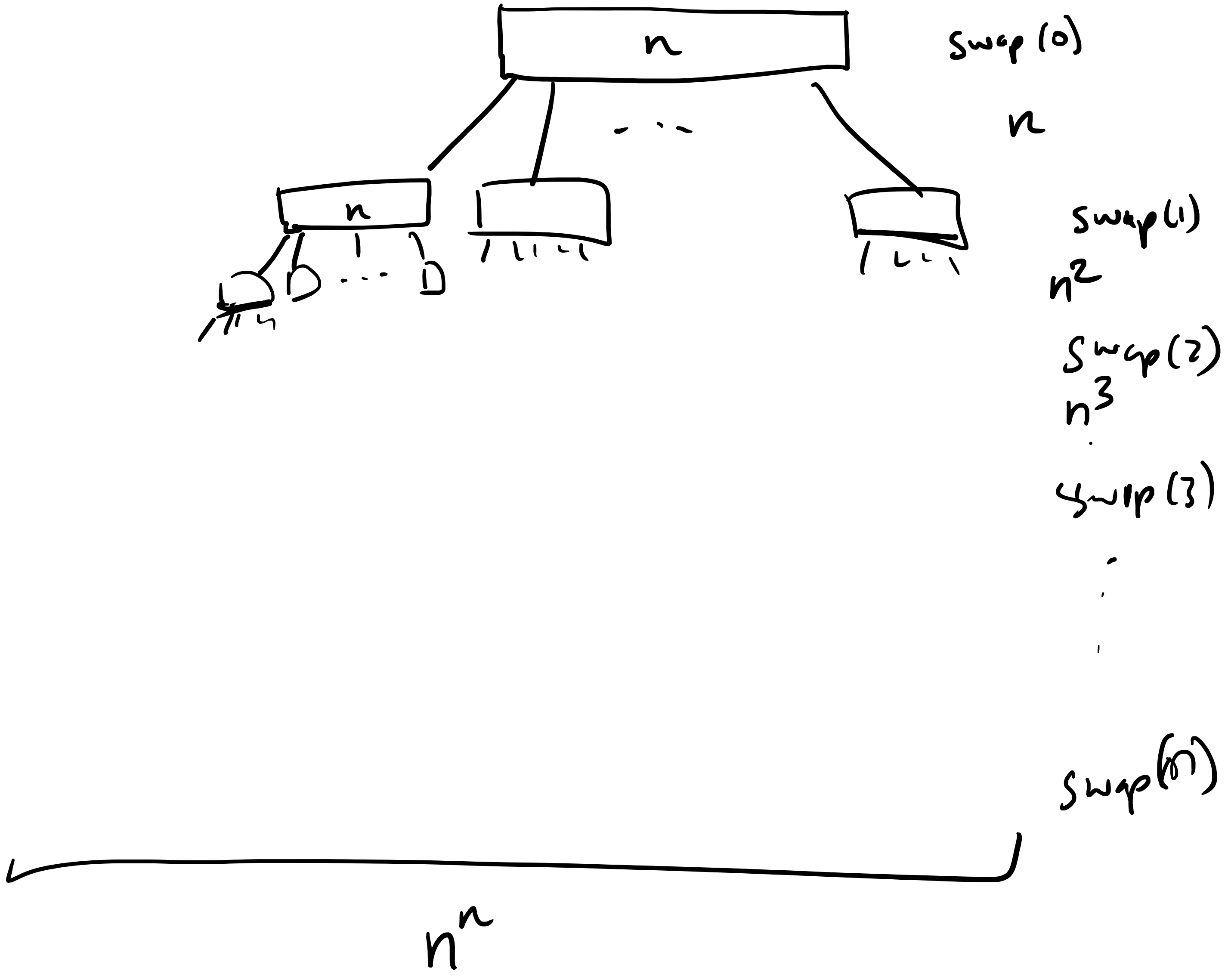
$O(n \log n)$

$\log_2 4 = 2$

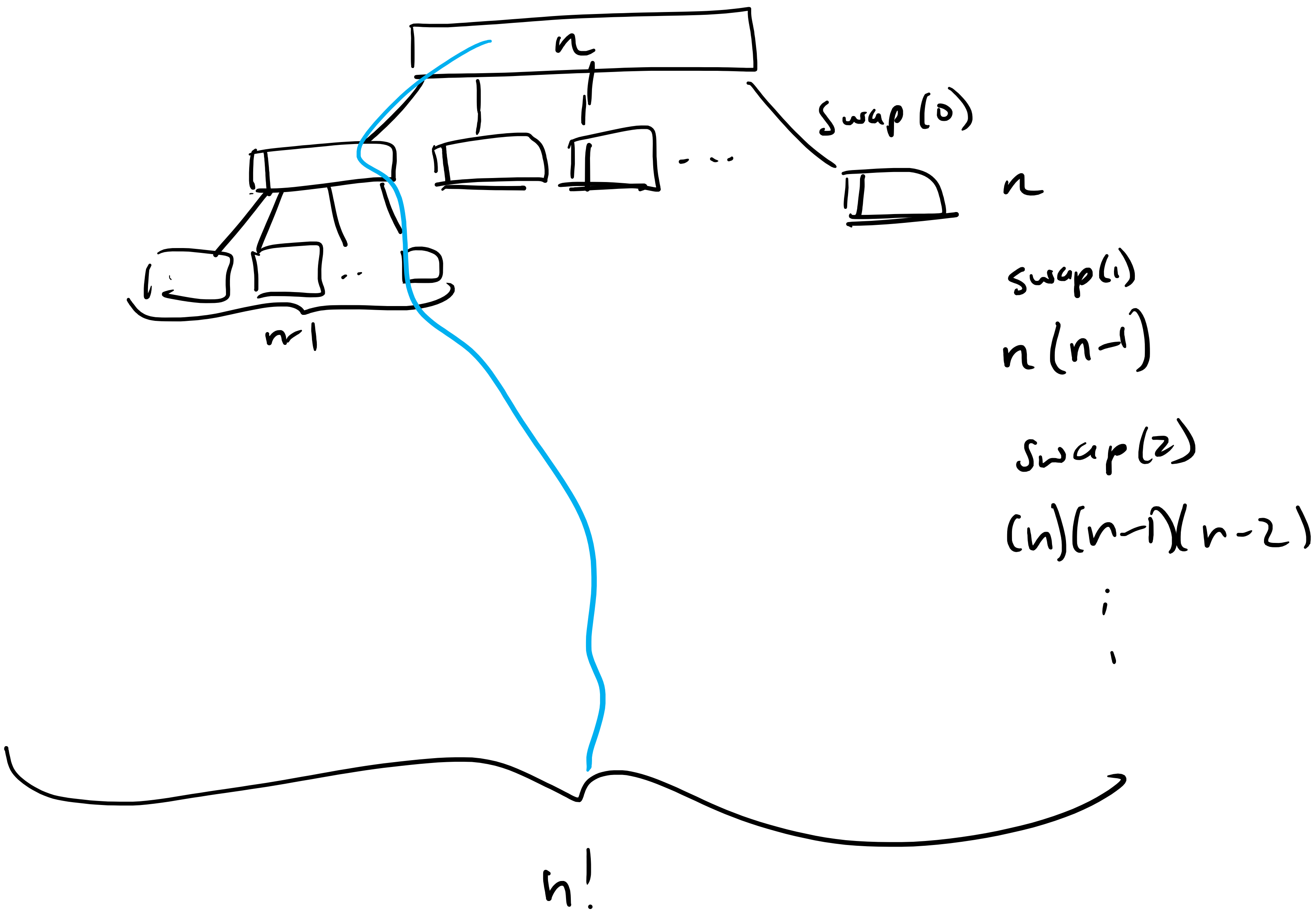
$\log_2 1 = 0$

$\log_2 2 = 1$

$\log_4 2 = \frac{1}{2}$



n
swap(0)
n
n
swap(1)
n^2
swap(2)
n^3
swap(3)
swap(n)
n^n


n
swap(0)
n
n-1
swap(1)
n(n-1)
swap(2)
(n)(n-1)(n-2)
n!