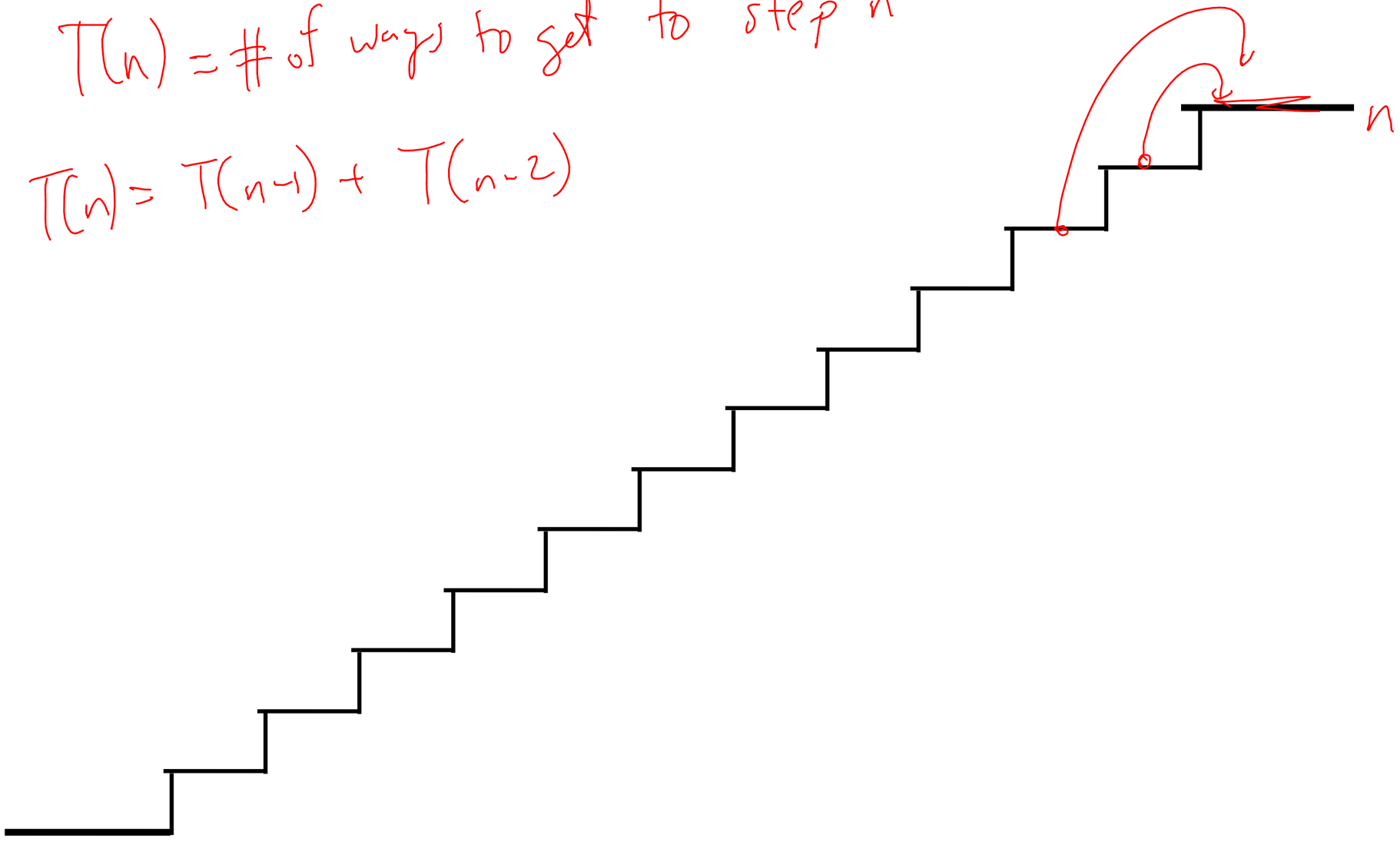
L8

shelat
16f-4800
oct 4 2016

Dynamic Programming

$T(n)$ = # of ways to get to step n

$T(n) = T(n-1) + T(n-2)$

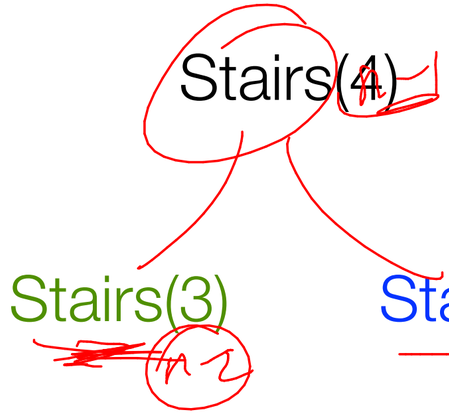


```
Stairs(n)
    if n<=1 return 1

    return Stairs(n-1) + Stairs(n-2)
```

Stairs(n) if n<=1 return 1

ret Stairs(n-1) + Stairs(n-2



Stairs(2) Stairs(1)

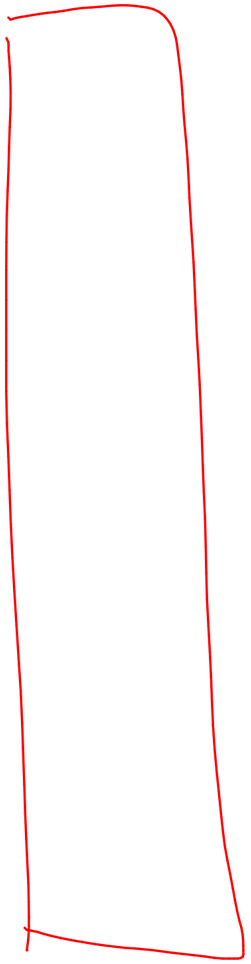
Stairs(2) Stairs(1) Stairs(1) Stairs(0) Stairs(1) Stairs(0)

```
initialize memory M

Stairs(n)
```

```
Stairs(n)
    if n<=1 then return 1
    if n is in M, return M[n]
    answer = Stairs(i-1)+ Stairs(i-2)
    M[n] = answer
    return answer
```

Stairs(5)



Stairs(n)

```
stair[0]=1
stair[1]=1
```

for i = 2 to n

    stairs[i] = stairs[i-1] + stairs[i-2];

return stairs[n]

Stairs(n)

```
stair[0]=1
stair[1]=1

for i=2 to n
   stair[i] = stair[i-1]+stair[i-2]
return stair[i]
```

pequod.cs.virginia.edu:3001/

2.html

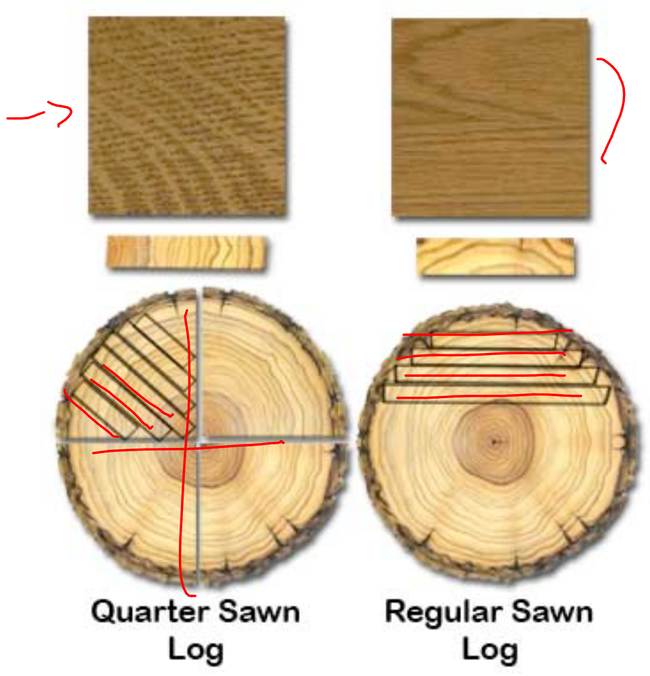
Dynamic Programming

two ideas

recursive structure

memoizing

wood cutting

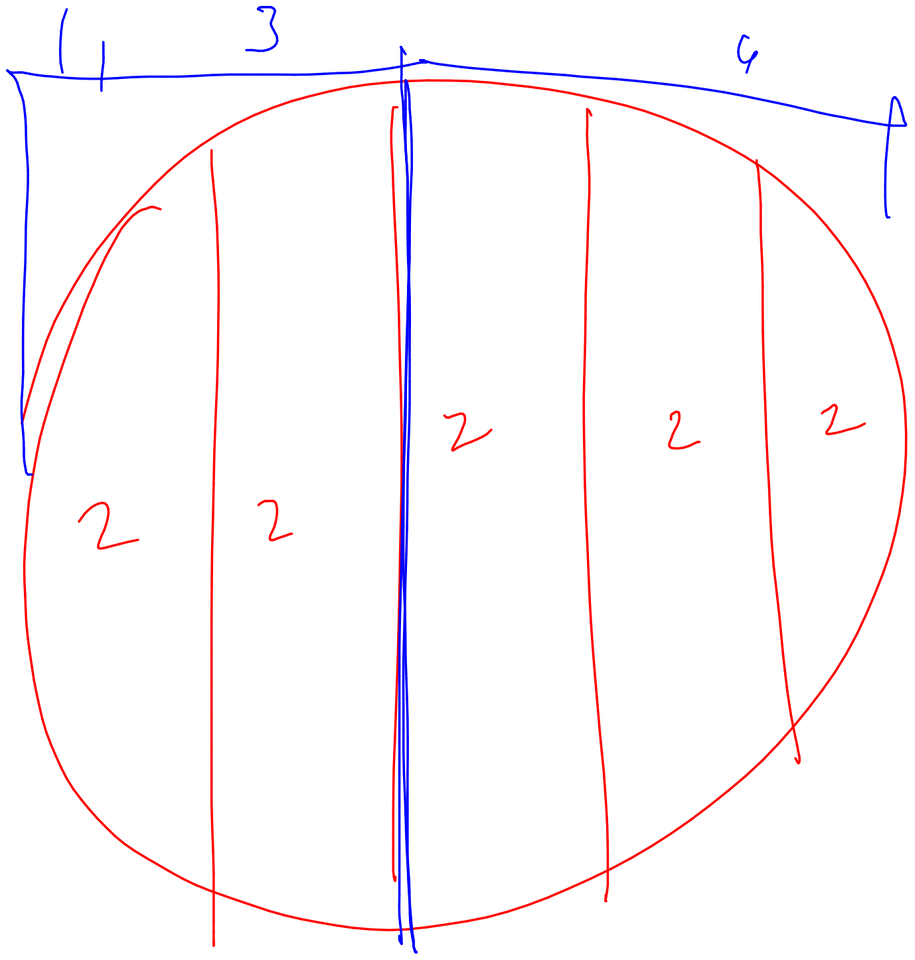


http://www.amishhandcraftedheirlooms.com/quarter-sawn-oak.htm

http://snlm.files.wordpress.com/2008/08/bill-wakefield-and-carl-fie.gif

Spot price for lumber

1"	2"	3"	4"	5"	6"	7"	8"
2	5	8	19	24	36	80	100



Log cutter dilemma

input to the problem: $n, (p_1, \ldots, p_n)$ → spot price for each thickness

width of your tree

goal: MAX revenue, i.e.

determine max revenue you can earn from cutting the n"-thick log into slabs of sizes $\{i_1, i_2 \ldots i_k\}$ s.t.

$$\sum_{j=1}^{k} i_j \leq n$$

$$\max \sum_{j=1}^{k} p_{i_j}$$

Greedy fails

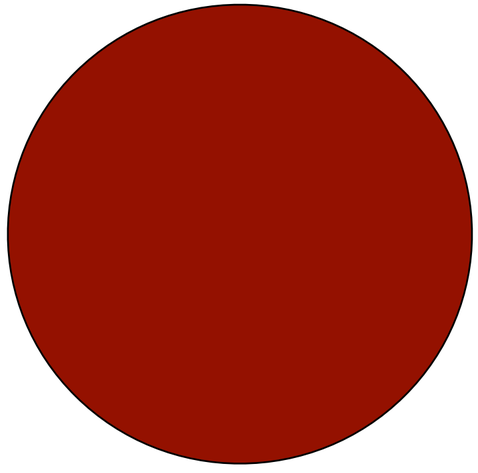
1"	2"	3"	4"	5"
1$	6$	7$	8$	10$
1	3	2 1/7	2	2

greedy: 10$

2", 3" → 13$

2, 2, 1 → 13$

5" log



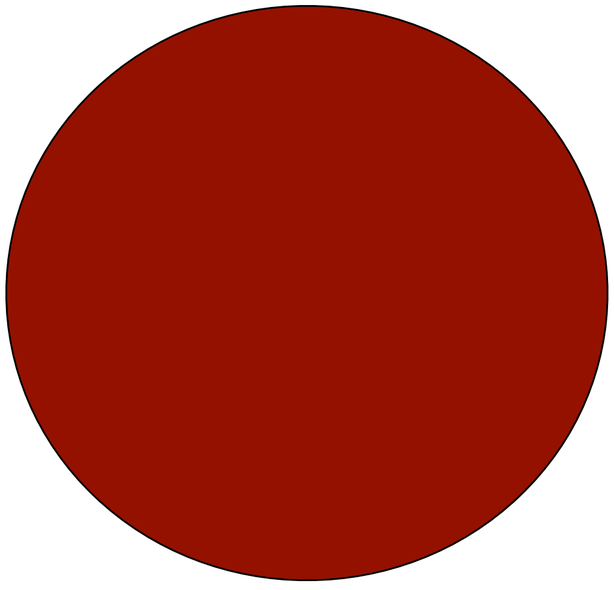
Greedy “Avg” fails

1”	2”	3”	4”	5”	6”
1$	18$	24$	36$	50$	50$
1	9	8	9	10	8..

5", 1" = 51 $

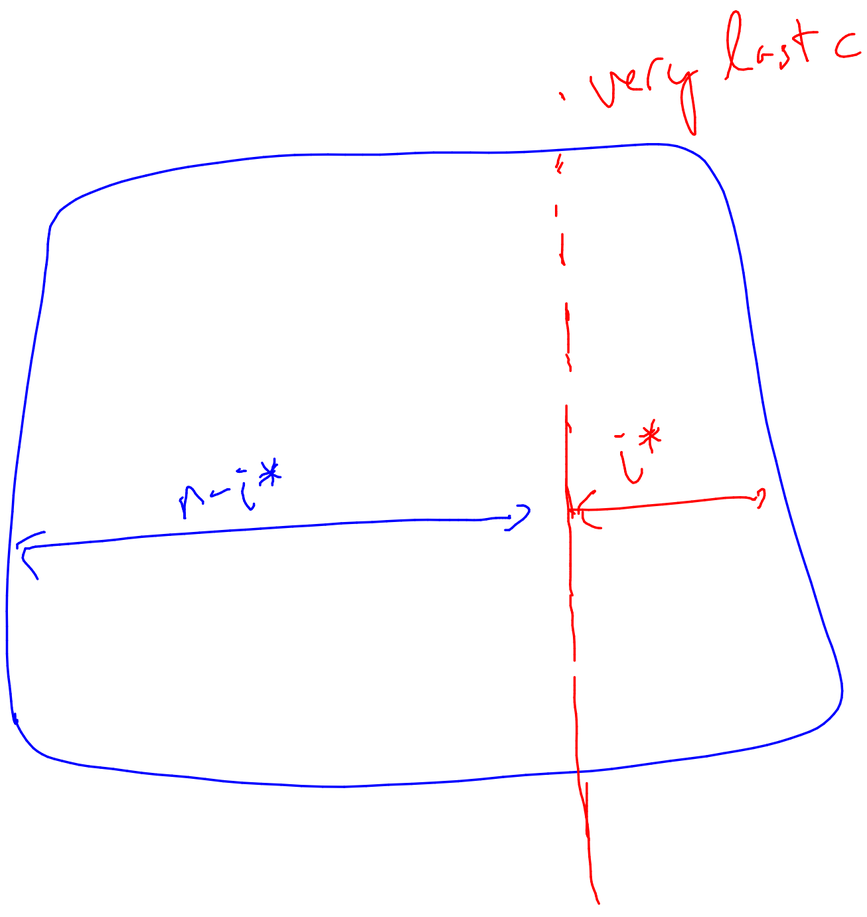
4", 2" = 54 $

6” log



Observation

$Best_n$ = max revenue from slicing an n'' thick log.



$Best_n = P_{i^*} + Best_{n-i^*}$

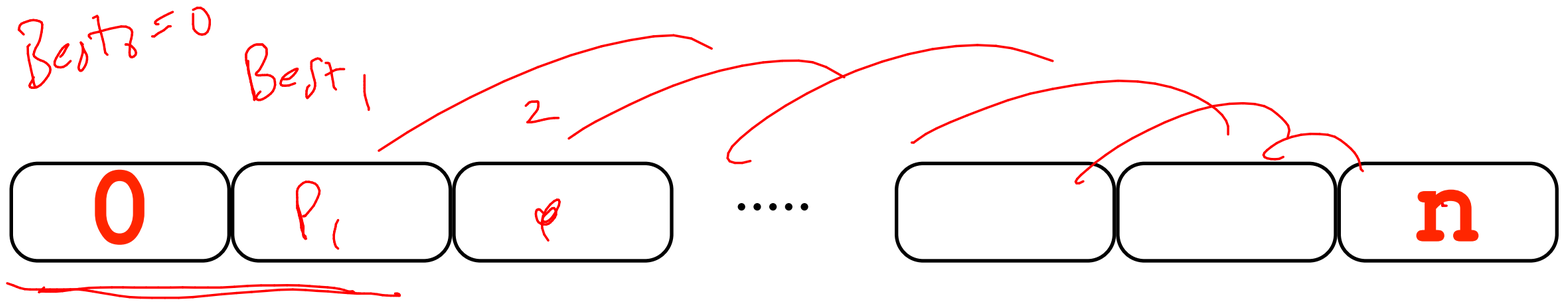
how many choices are there for i^* ?

$1 \ldots n$

Solution equation

$$Best_n = \max \begin{cases} P_1 + Best_{n-1} \\ P_2 + Best_{n-2} \\ P_3 + Best_{n-3} \\ \cdots \\ P_n + Best_0 \end{cases}$$

Approach



$Best_1 = \max \begin{cases} P_1 + B_0 \end{cases}$

$Best_2 = \max \begin{cases} P_1 + Best_1 \\ P_2 + Best_0 \end{cases}$

$Best_3 = \max \begin{cases} P_1 + B[2] \\ P_2 + B[1] \\ P_3 + B[0] \end{cases}$

BestLogs($n, (p_1, \ldots, p_n)$)

```
    Initialize B[0…n]=0
```

BestLogs($n, (p_1, \ldots, p_n)$)

```
Initialize B[0…n]=0

for i=1 to n
  Best[i] = max_{k=1…i} {p_k + Best[i − k]}

return Best[n]
```

BestLogs($n, (p_1, \ldots, p_n)$)

```
Initialize B[0…n]=0

for i=1 to n
  Best[i] = max_{k=1…i} {p_k + Best[i − k]}

return Best[n]
```

The actual cuts?

Remember your optimal choices

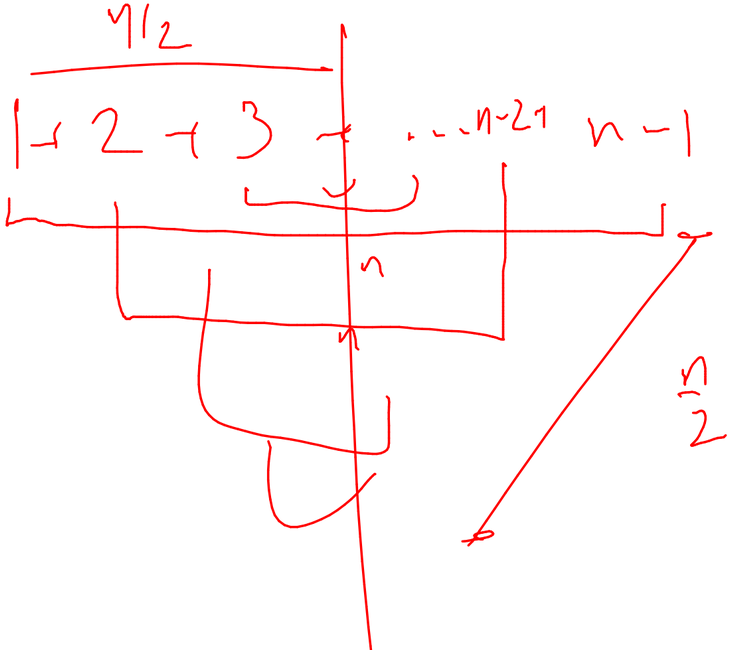
BestLogs($n, (p_1, \ldots, p_n)$)

```
Initialize B[0…n]=0
for i=1 to n
  Best[i] = max_{k=1…i} {p_k + Best[i − k]}
  choice[i] = k*
return Best[n], choice[]
```

Initialize best[0…n] = 0 ; c[0…] = -1 — Θ(n)

run Θ(n) iterations

choice from this line which resulted in the max



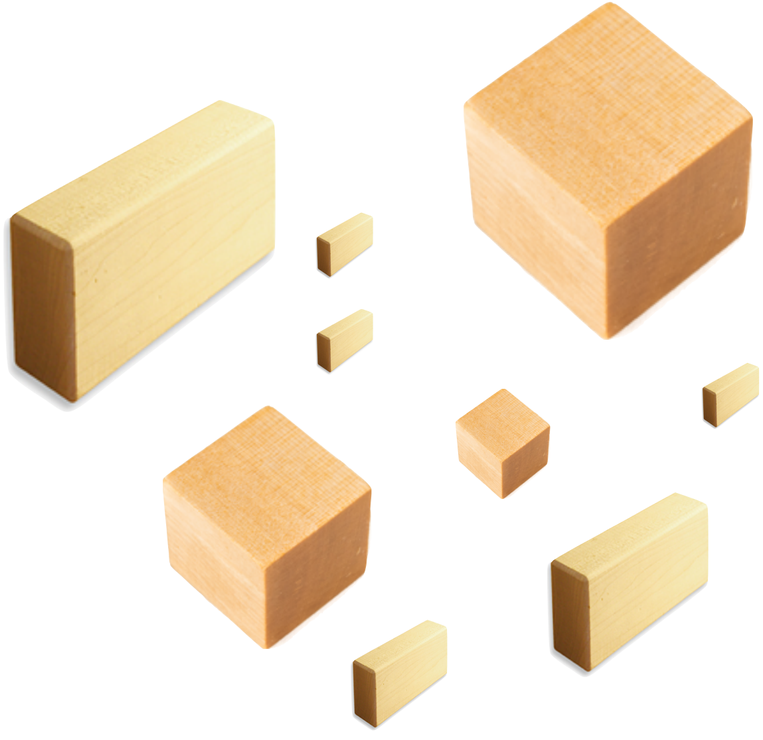
$\frac{n}{2} \cdot n \approx \frac{n^2}{2}$

$\frac{n(n-1)}{2}$ =

$\Theta(n^2) + \Theta(1) + \Theta(n) =$

$\Theta(n^2)$

Knapsack



Sack has Capacity W

W

w_1 v_1

w_2 v_2

w_3 v_3

Each item
has a weight w_i
and a value v_i

Goal is to select
a set of items that
"**fit**" into the Knapsack
and have the **greatest**
value.

Define a quantity that captures the optimal solution:

S({1,…,n}, C) : max value obtainable from items {1…n} that fit in sack of size C

Consider the very first item. Is it part of the max solution?

w_1 v_1

$$S(\{1 \dots n\}, C) = \max \begin{cases} S(\{2 \dots n\}, C - w_1) + v_1 & \text{if item 1 is part of the best solution} \\ S(\{2 \dots n\}, C) \end{cases}$$

Recursive structure

Either the best solution doesn't include item i

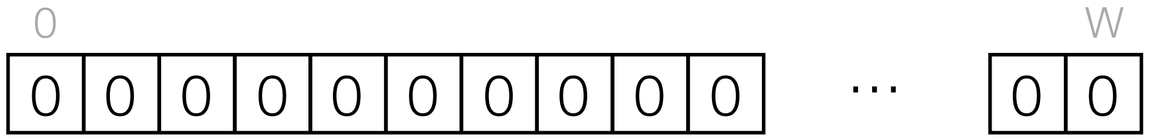
$$S(\{i\ldots n\}, C) = \max \begin{cases} S(\{i+1\ldots n\}, C) \\ v_i + S(\{i+1\ldots n\}, C - w_i) \end{cases}$$

Or, it includes item i and the best solution for the remaining space, $C-w_i$

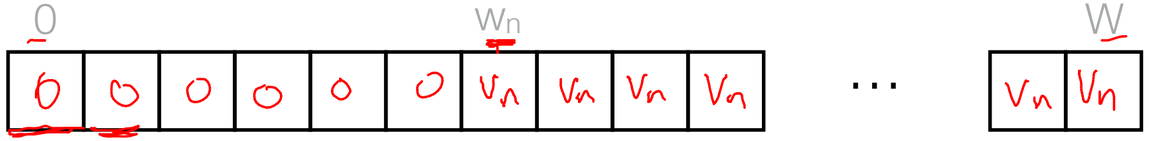
Pick an order

Start from the last item

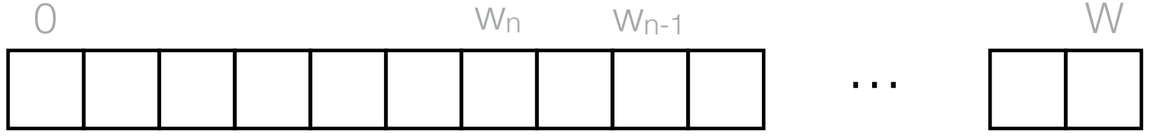
S({ }, 0…W)



S({ n }, 0…W)



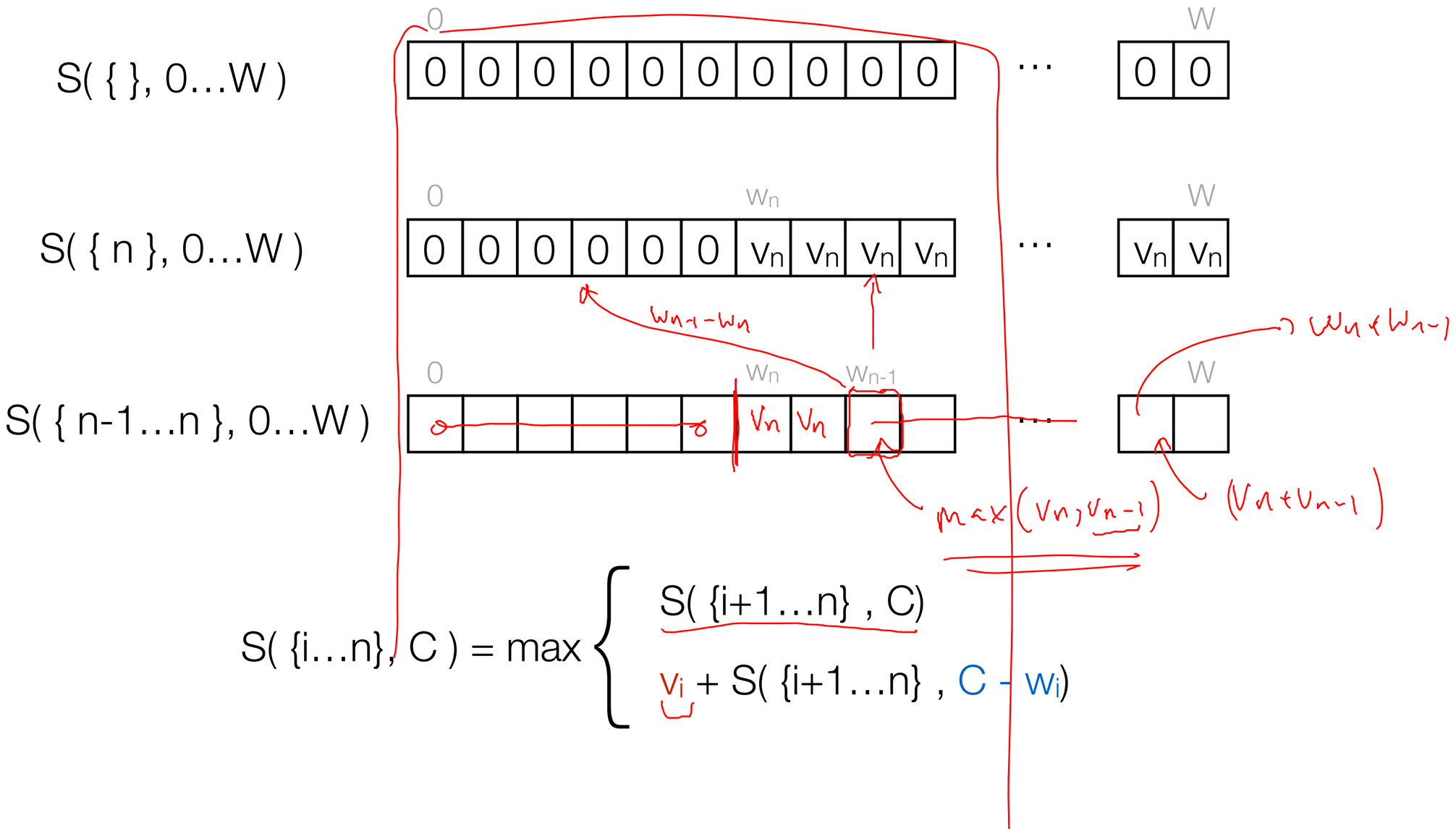
S({n}, 0)



$$S(\{i \ldots n\}, C) = \max \begin{cases} S(\{i+1 \ldots n\}, C) \\ v_i + S(\{i+1 \ldots n\}, C - w_i) \end{cases}$$

Pick an order

Start from the last item



$$S(\{i \ldots n\}, C) = \max \begin{cases} S(\{i+1 \ldots n\}, C) \\ v_i + S(\{i+1 \ldots n\}, C - w_i) \end{cases}$$

Knapsack($\{w_i, v_i\}_n$, W)

Initialize S({n-1}, 0…W) = 0

for i from n to 1

for j from 0 to W

S(i, j) = max $\begin{cases} S(i+1, j) \\ v_i + S(i+1, j - w_i) \end{cases}$

as long as $j > w_i$
because otherwise,
this term is negative

Return S(1,W)

Knapsack($\{w_i, v_i\}_n$, W)

Initialize S({n-1}, 0…W) = 0

for i from n to 1

    for j from 0 to W

        S(i, j) = S(i+1, j)

        if j > wi and S(i+1, j-wi)+ vi > S(i, j)
            S(i,j) = S(i+1, j-wi)+ vi

Return S(1,W)

How can we determine WHICH items are selected?

Knapsack($\{w_i, v_i\}_n$, W)

```
Initialize S( {n-1}, 0…W) = 0

for i from n to 1

        for j from 0 to W

            S( i, j ) = S( i+1, j)
            pick( i, j ) = false
            if j > wi and S( i+1, j-wi )+ vi > S( i, j)
                S(i,j) = S( i+1, j-wi )+ vi
                pick( i, j ) = true
//Backtrack to find solution
cap = W, sol = {}
for i from 1 to n
        if picked(i,cap) = true { sol = sol + {i};  cap = cap - wi; }
```

PROBLEM: REDUCE IMAGE WIDTH

scaling: distortion

deleting column: distortion

delete the most invisible seam

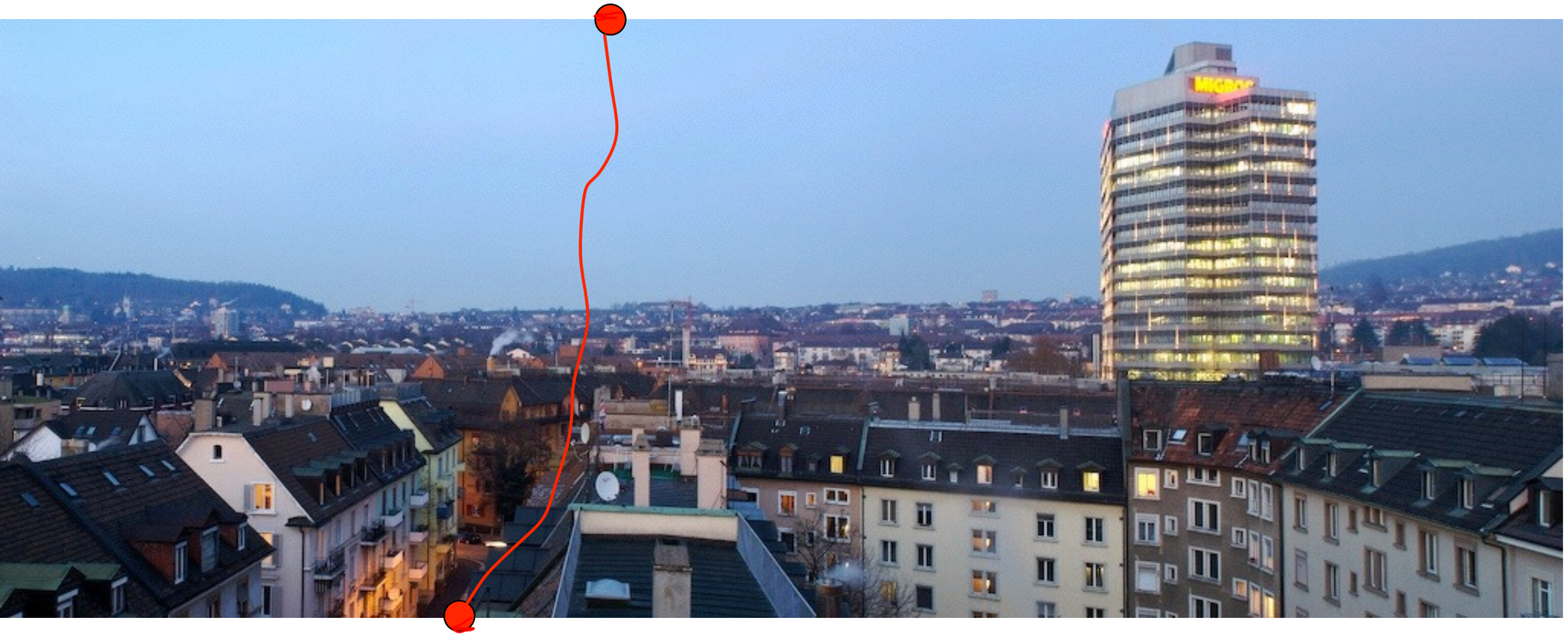
http://www.youtube.com/watch?v=qadw0BRKeMk

http://www.youtube.com/watch?v=qadw0BRKeMk

DEMO?

http://rsizr.com/

WHICH SEAM TO DELETE?

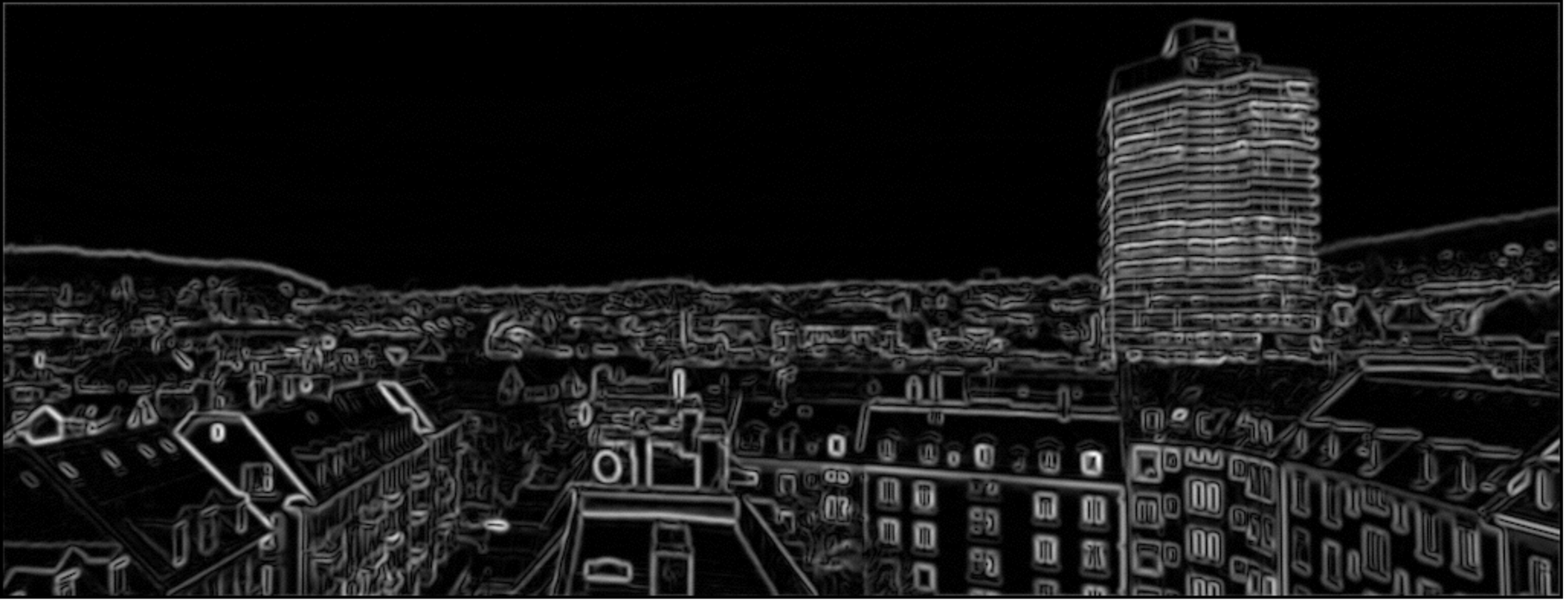


ENERGY OF AN IMAGE

$$e(\mathbf{I}) = \left| \frac{\partial}{\partial x}\mathbf{I} \right| + \left| \frac{\partial}{\partial y}\mathbf{I} \right|$$

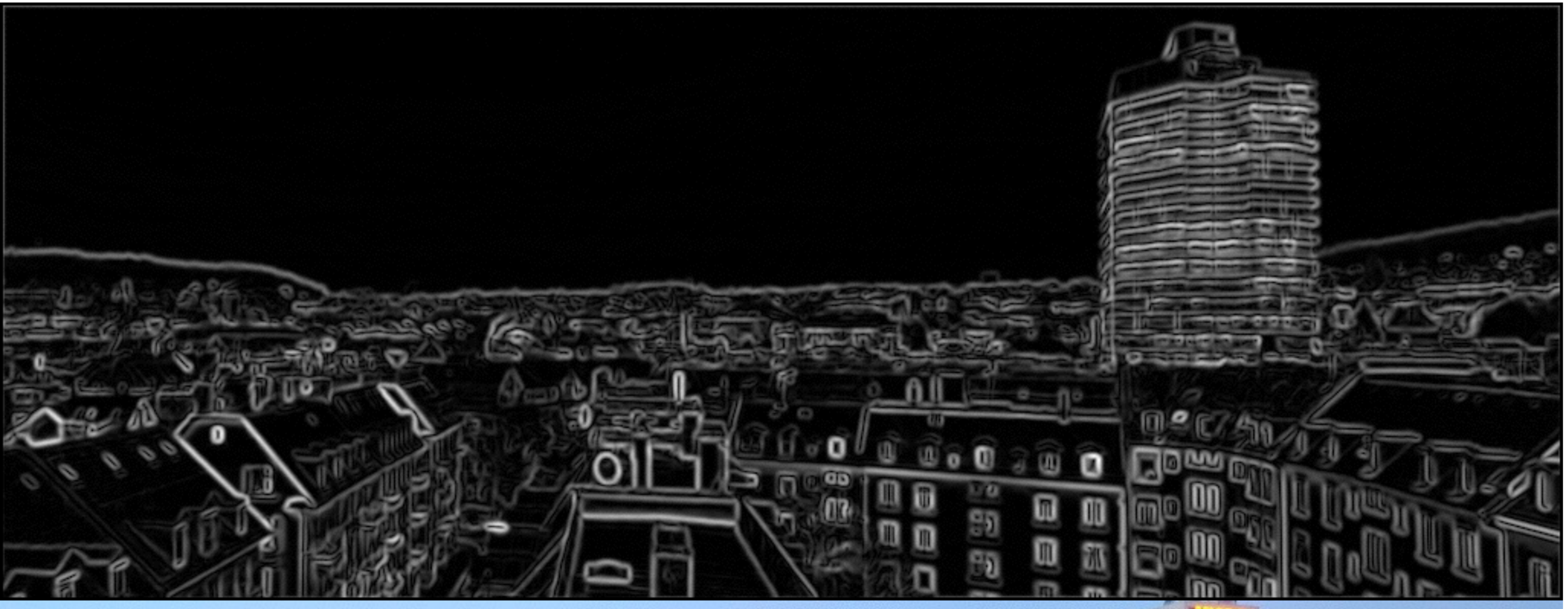
“magnitude of gradient at a pixel”

$$\frac{\partial}{\partial x} I_{x,y} = I_{x-1,y} - I_{x+1,y}$$



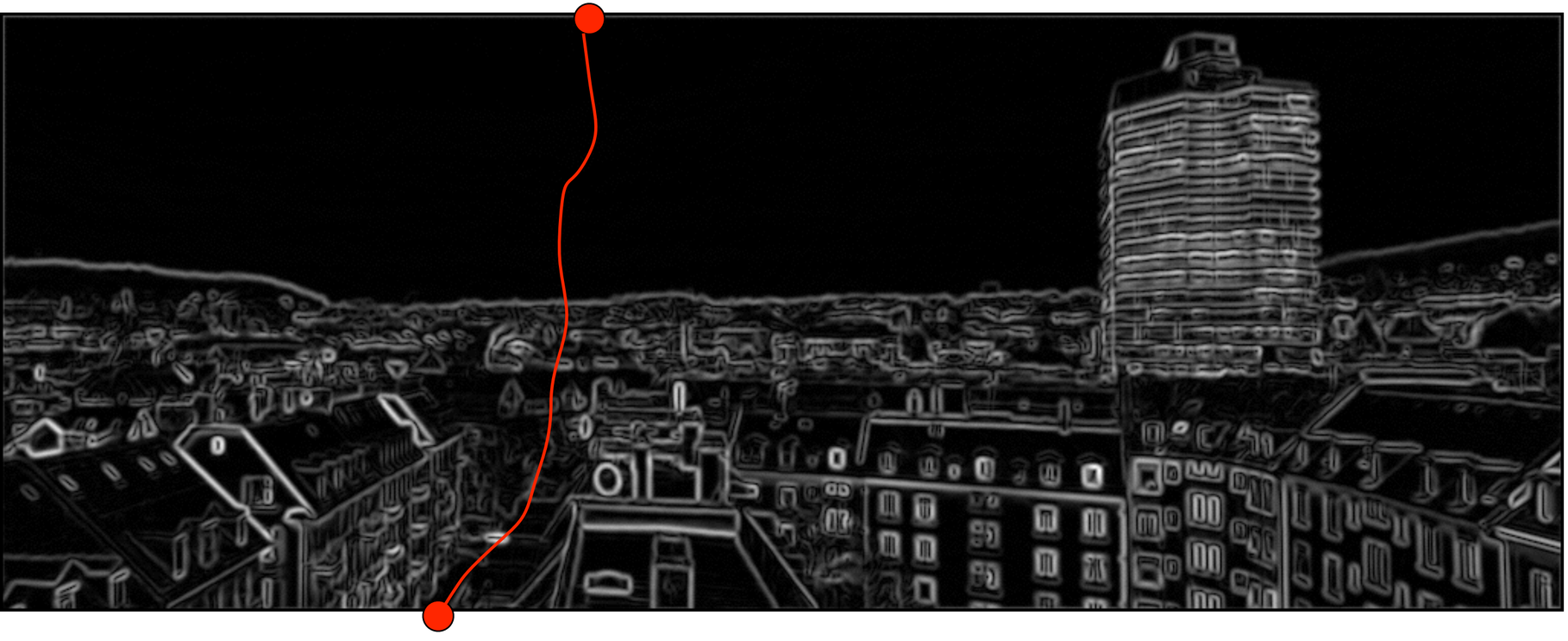
energy of sample image

thanks to Jason Lawrence for gradient software

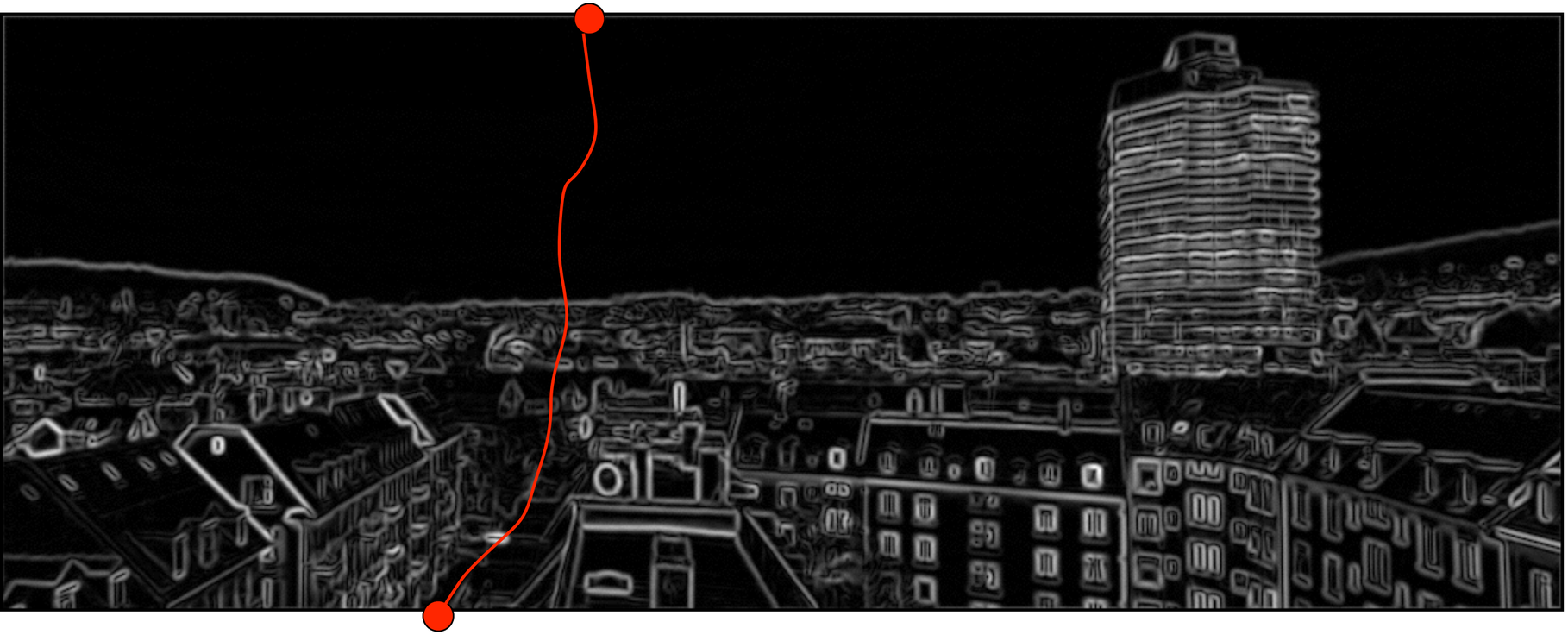



MIGROS

BEST SEAM HAS LOWEST ENERGY



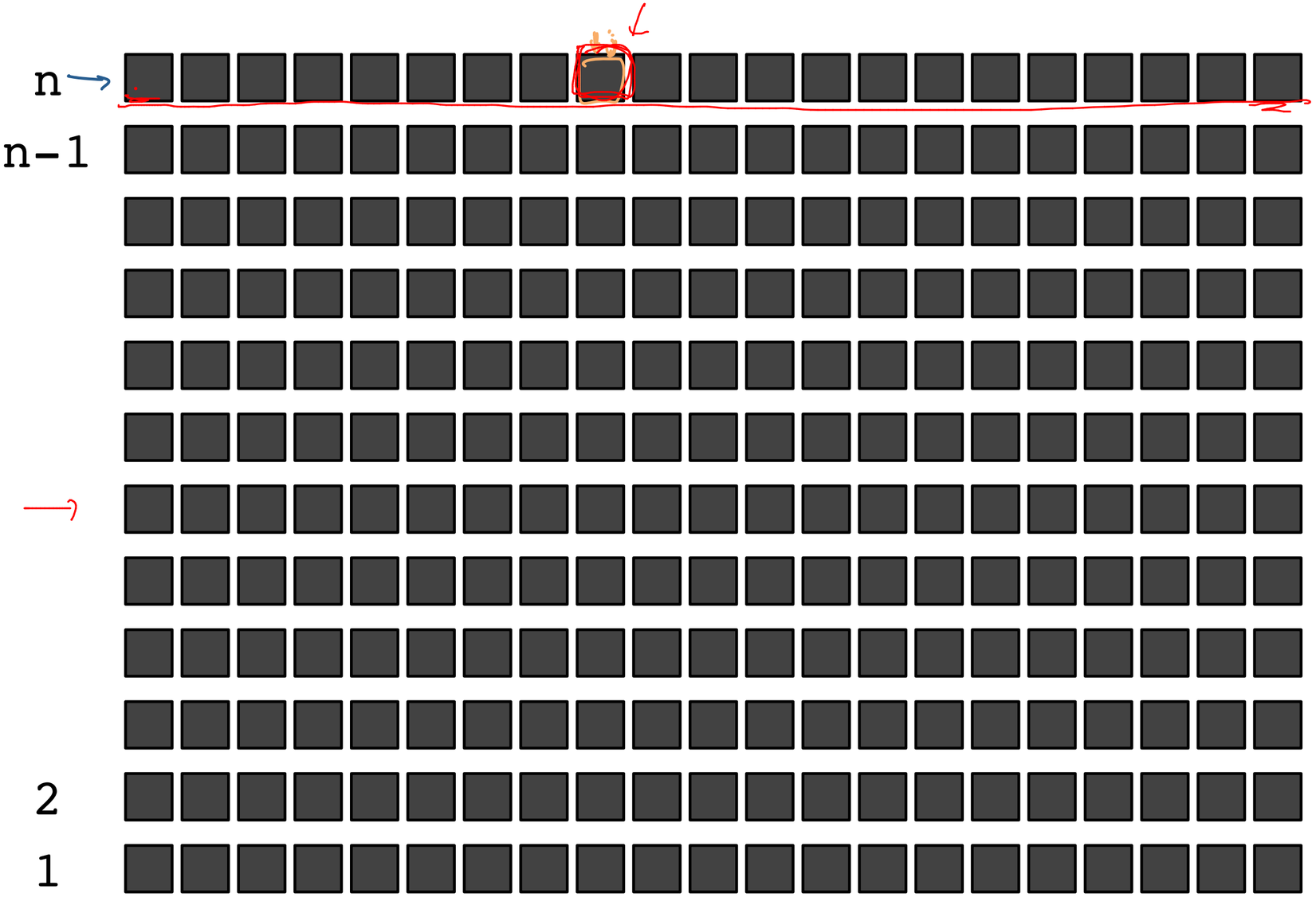
FINDING LOWEST ENERGY SEAM?



Define a variable:

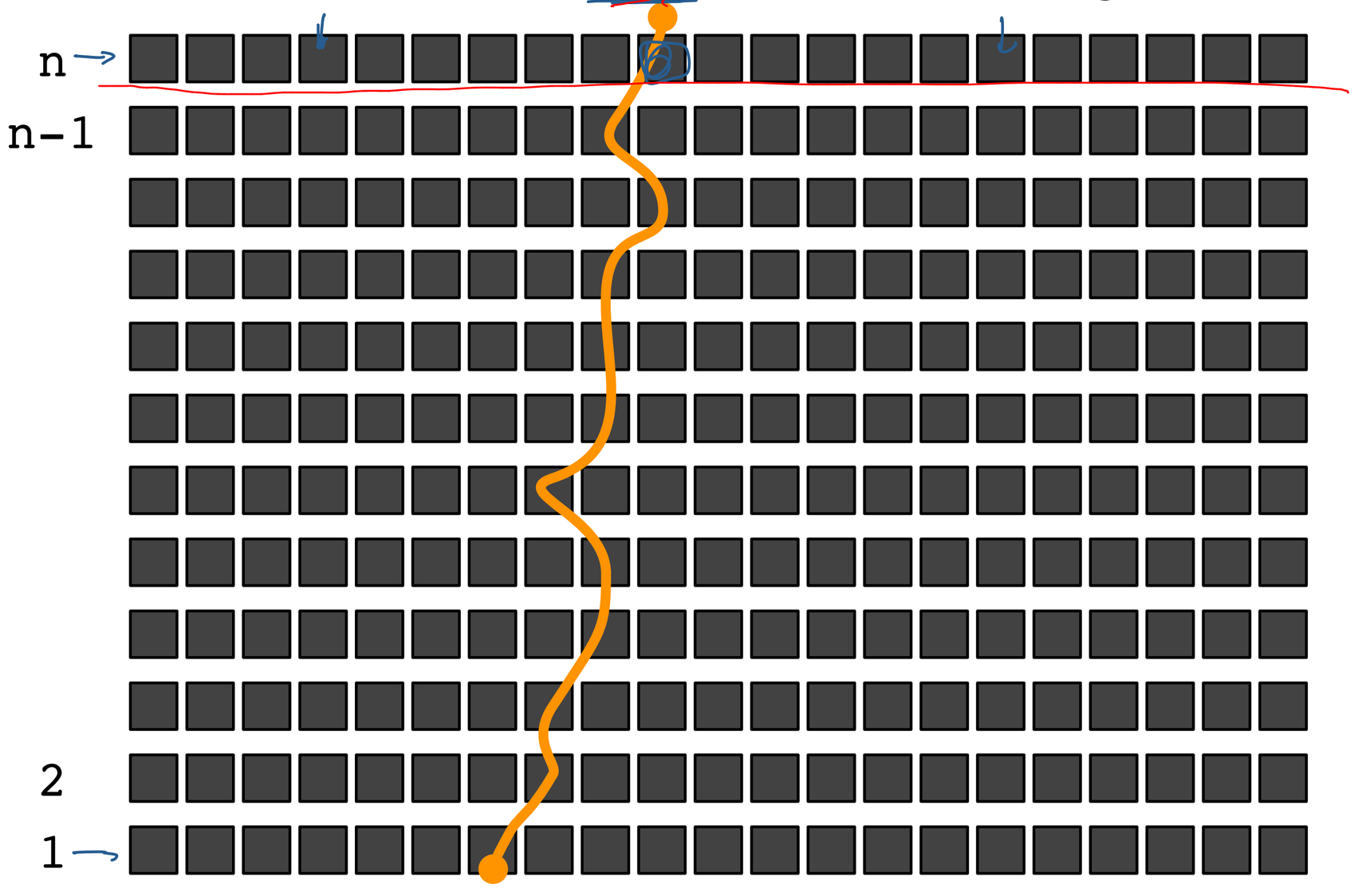
$S_i(j)$:

definition: $S_n(j)$: value of the least energy seam that begins in row 1 and ends at row n, pixel j



definition:

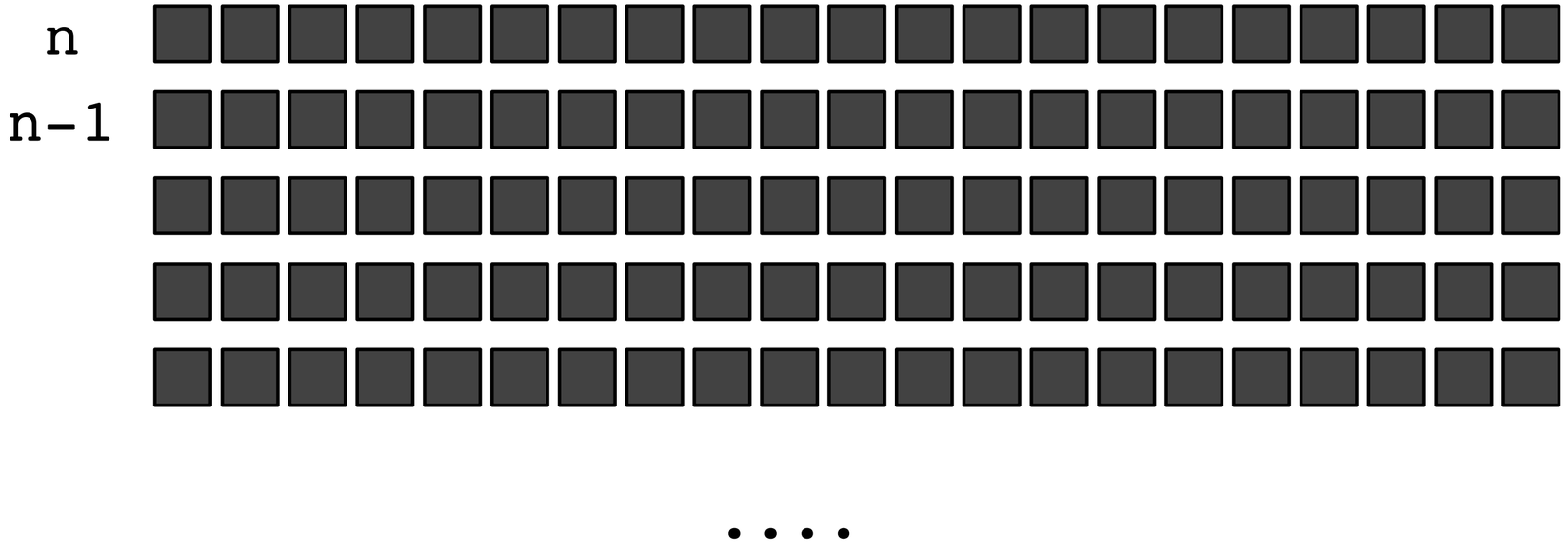
$S_n(j)$ best seam ending at (n,j)



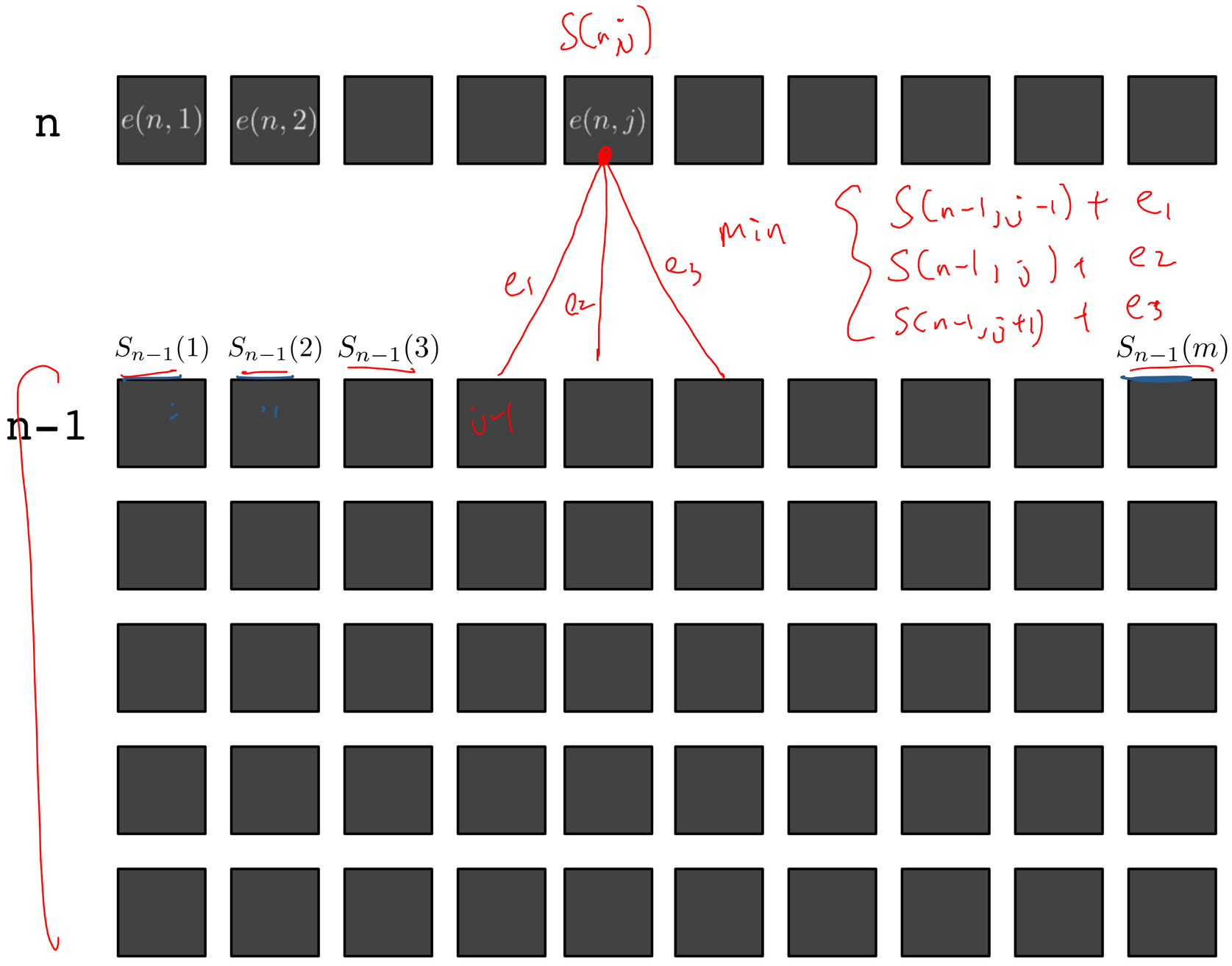
BEST SEAM TO DELETE HAS TO BE THE BEST AMONG

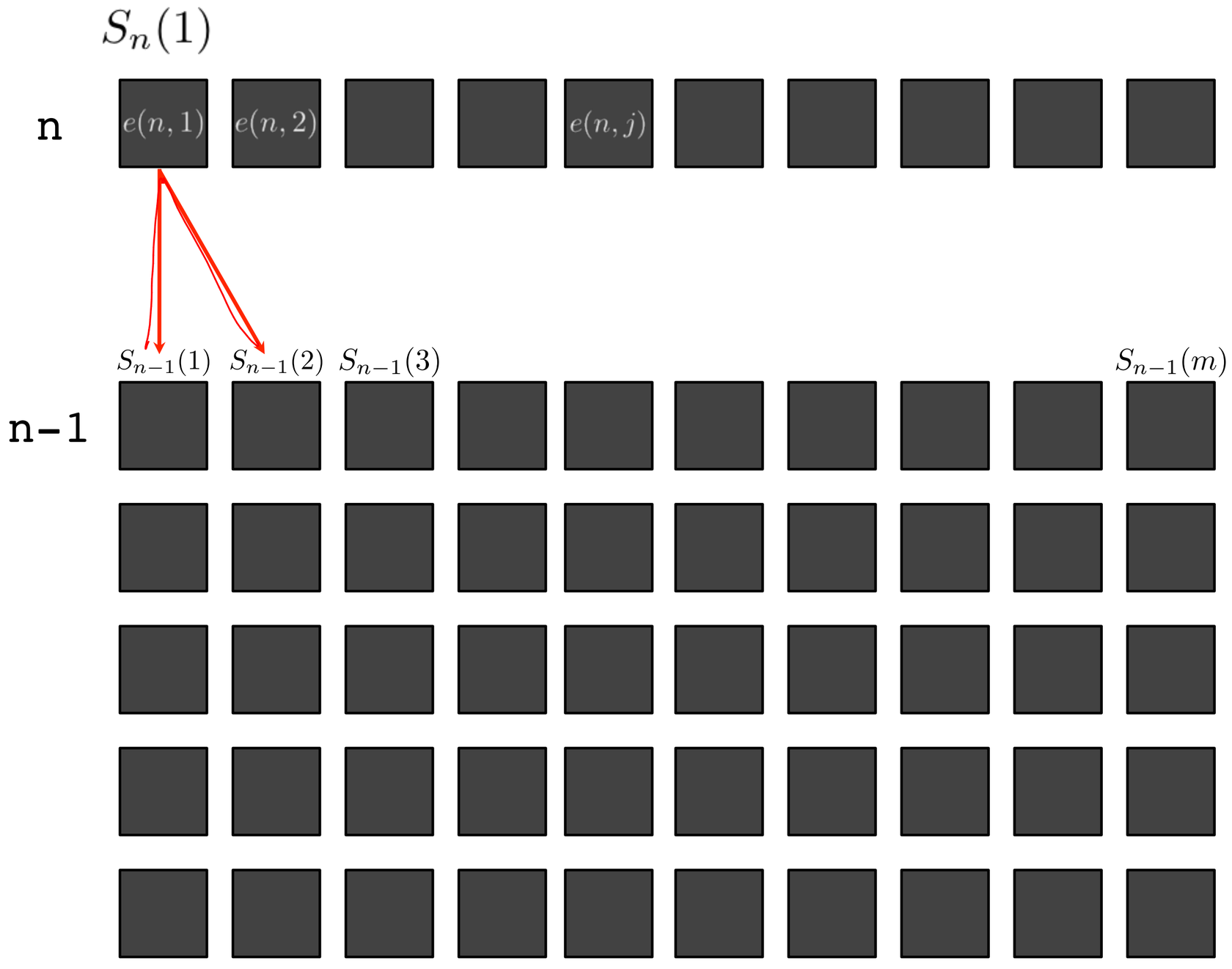
$$S_n(1), S_n(2), \ldots, S_n(m)$$

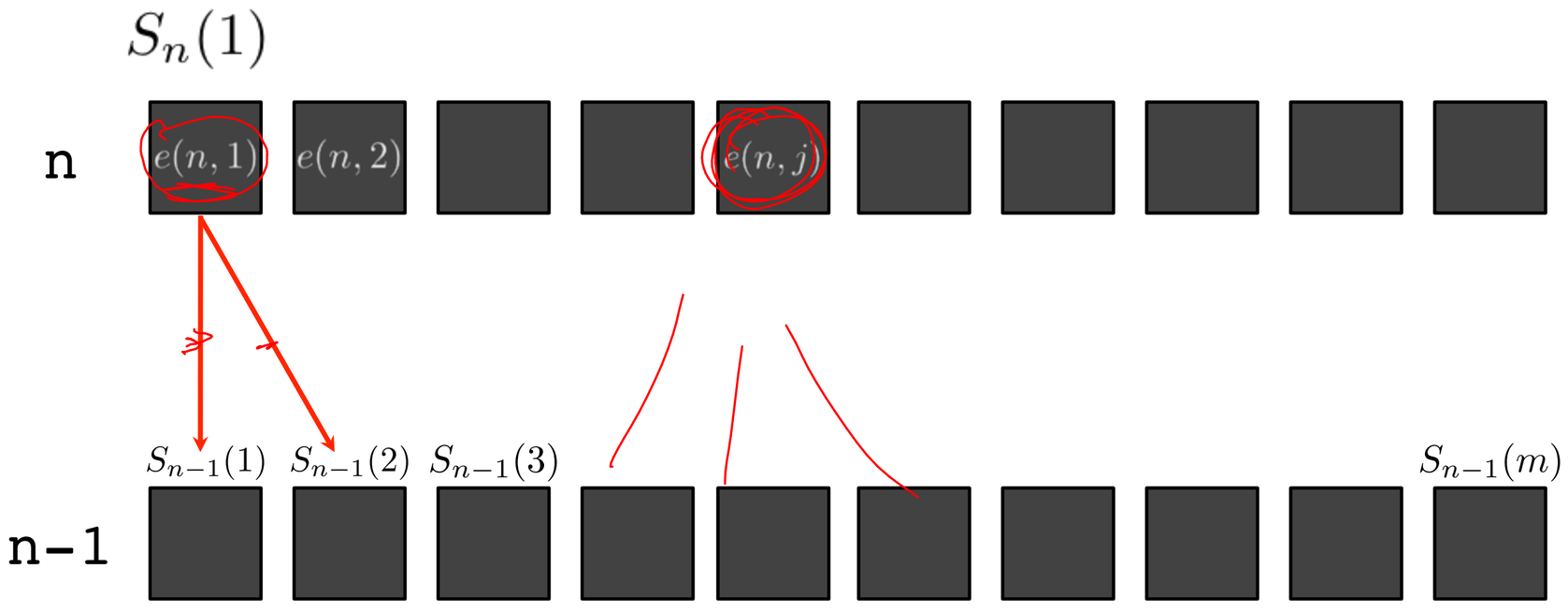
IDEA: COMPUTE + COMPARE



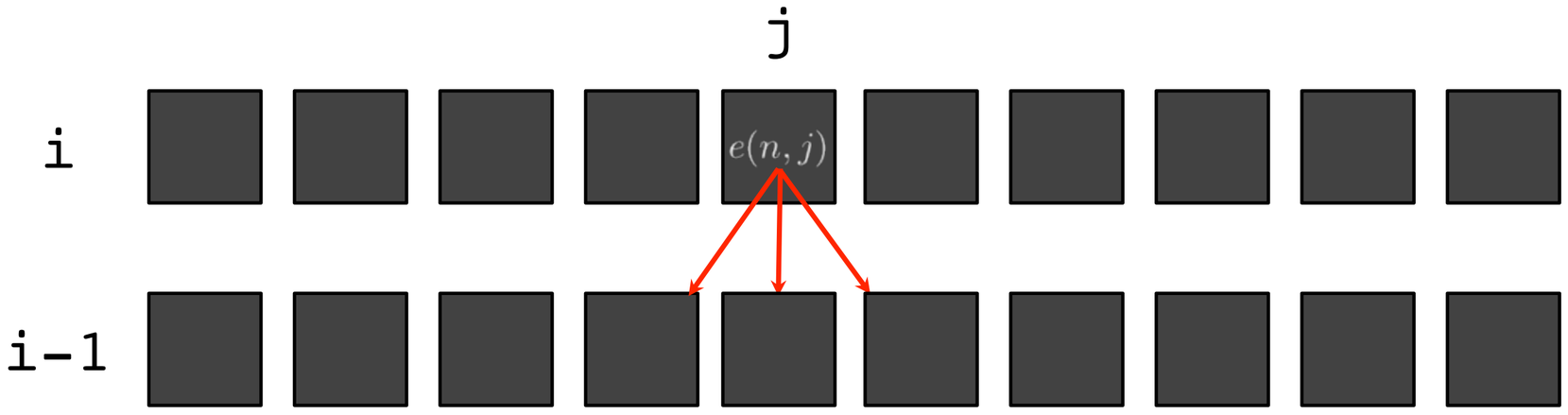
IMAGINE YOU HAVE THE SOLUTION TO THE FIRST N-1 ROWS


S(n,j)
n
e(n,1)
e(n,2)
e(n,j)
e1
e2
e3
min
S(n-1,j-1) + e1
S(n-1,j) + e2
S(n-1,j+1) + e3
$S_{n-1}(1)$
$S_{n-1}(2)$
$S_{n-1}(3)$
$S_{n-1}(m)$
n-1
j-1

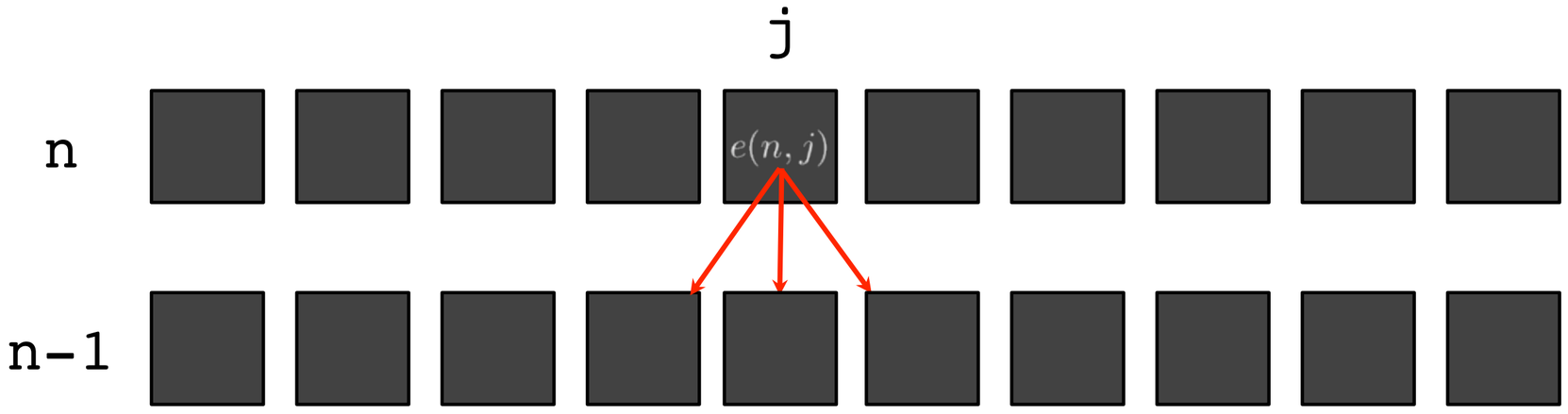

$S_n(1)$
n
$e(n,1)$
$e(n,2)$
$e(n,j)$
$S_{n-1}(1)$
$S_{n-1}(2)$
$S_{n-1}(3)$
$S_{n-1}(m)$
n-1



$$S_n(1) = e(n,1) + \min\{S_{n-1}(1), S_{n-1}(2)\}$$



$$S_i(j) =$$

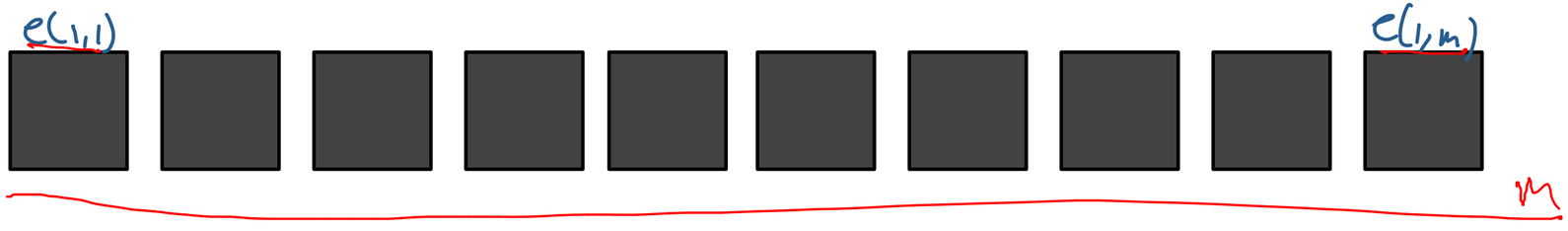


$$S_i(j) = e(i,j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

ALGORITHM

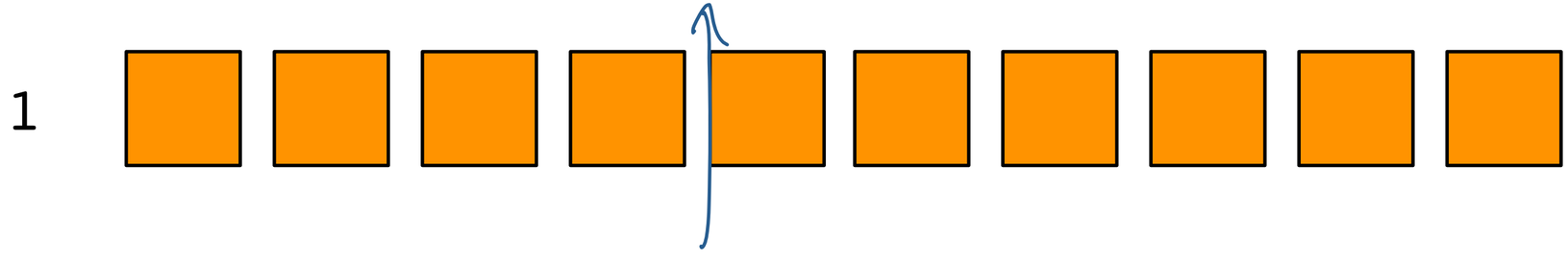
start at bottom of picture

1



ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1, i)$

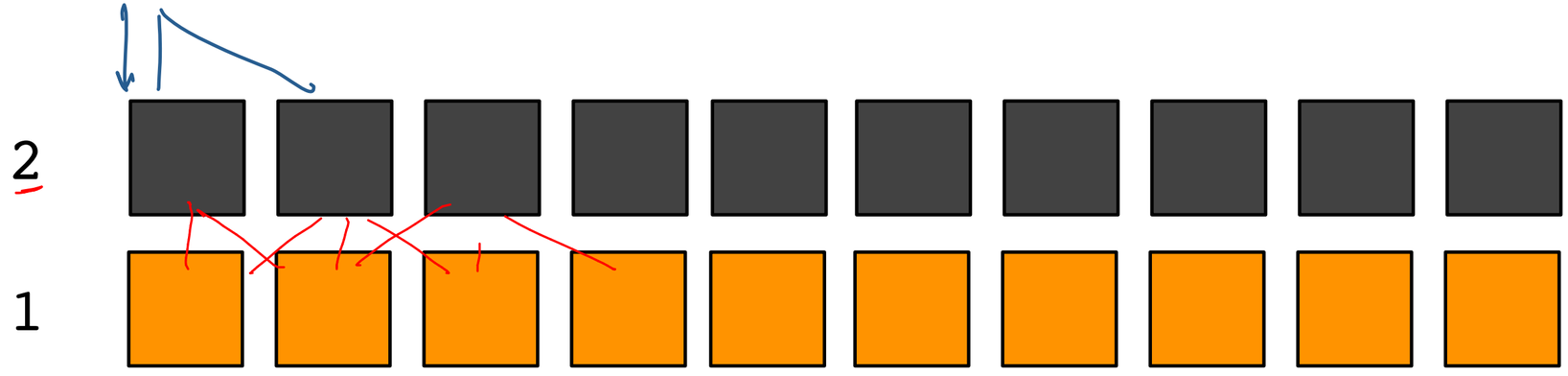


ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1, i)$

for `i=2 to n` use formula to compute $S_{i+1}(\cdot)$

$$S_i(j) = e(i, j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

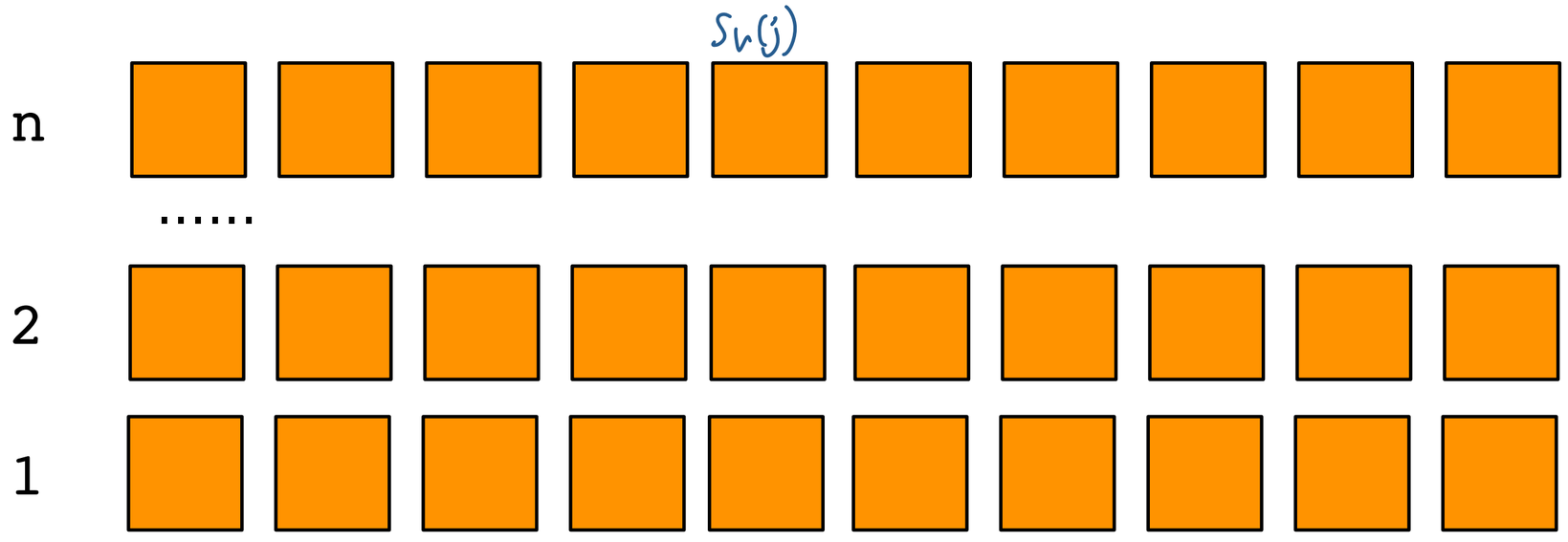


ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1,i)$

for `i=2,n` use formula to compute $S_{i+1}(\cdot)$

$$S_i(j) = e(i,j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$



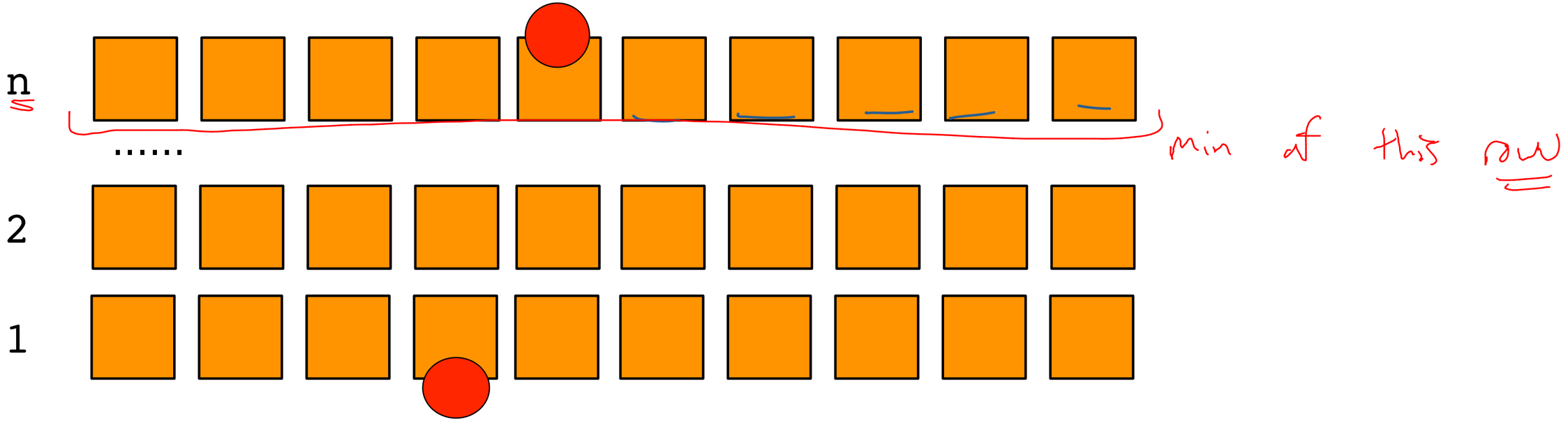
ALGORITHM

start at bottom of picture. initialize $S_1(i) = e(1,i)$

for `i=2,n` use formula to compute $S_{i+1}(\cdot)$

$$S_i(j) = e(i,j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

pick best among top row, backtrack.



RUNNING TIME

start at bottom of picture. initialize $S_1(i) = e(1, i)$

for `i=2,n` use formula to compute $S_{i+1}(\cdot)$

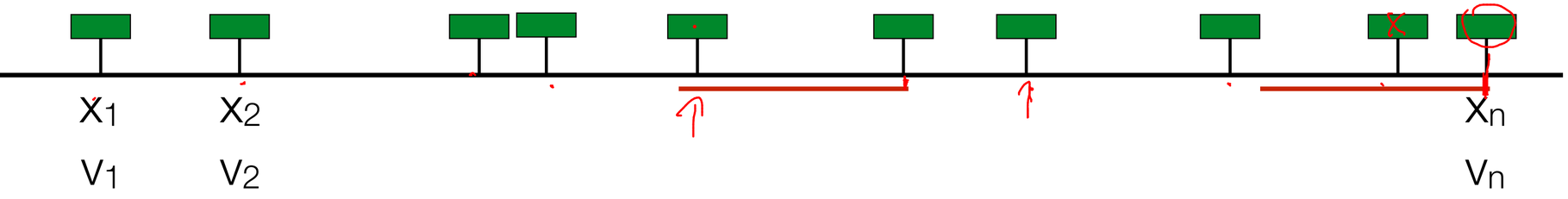
$$S_i(j) = e(i, j) + \min \begin{cases} S_{i-1}(j-1) \\ S_{i-1}(j) \\ S_{i-1}(j+1) \end{cases}$$

pick best among top row, backtrack.

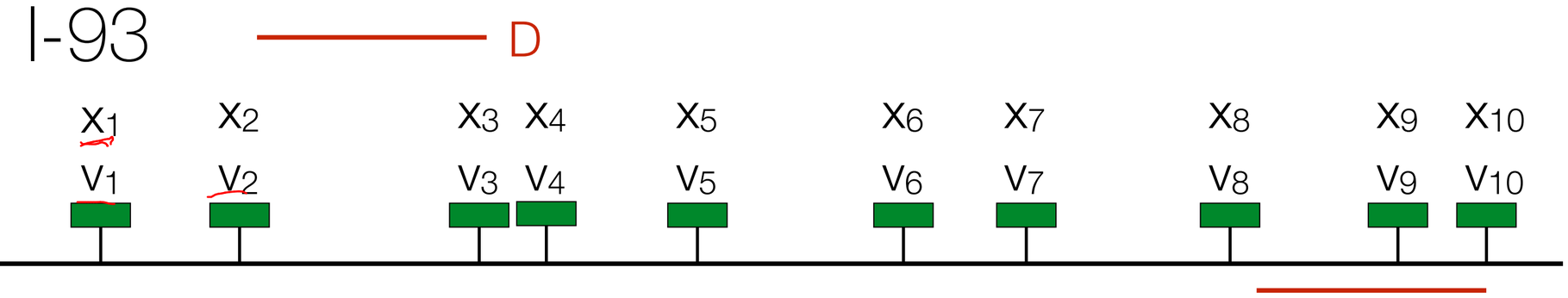
$\Theta(n \cdot m)$

Billboard problem

I-93



distance parameter
Cannot place ads that are closer than D miles apart

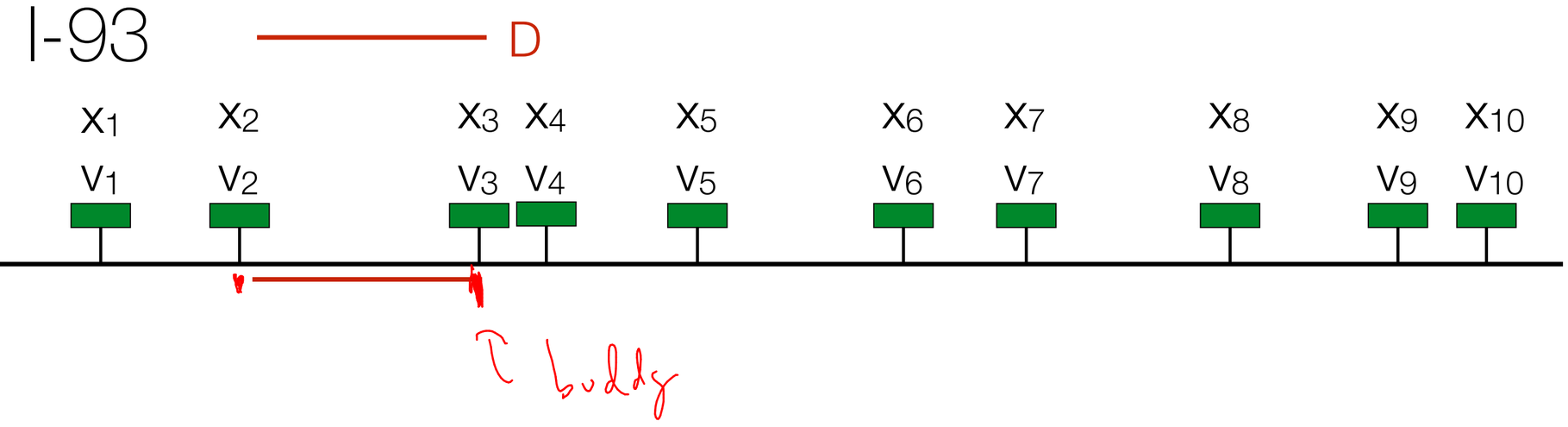


Input is $((x_1, \ldots, x_n)(v_1, \ldots, v_n), D)$

Best_n = max viewers for a valid campaign that uses billboards $\{1 \ldots n\}$ & D

$$\text{Best}_n = \max \begin{cases} \text{Best}_{n-1} \\ v_n + \text{Best} \end{cases}$$

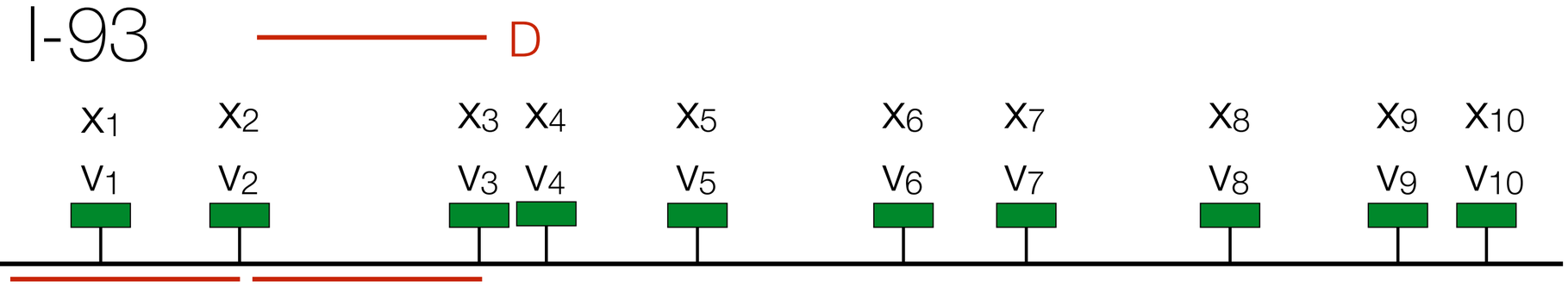
"closest billboard to n that is D away



$Best_1 = v_1$

$Best_2 = v_1 + v_2$

$Best_3 = \max \{ Best_2, \; v_3 + Best_1 \}$



$Best_1 =$

$Best_2 =$

$Best_3 =$

Billboard Problem

$$\text{BEST}_j = \max \begin{cases} \text{BEST}_{j-1} \\ v_j + \text{BEST}_{cl(j)} \end{cases}$$

```
best[0] = 0
for i=1 to n




return best[n]
```

Billboard Problem

$$\text{BEST}_j = \max \begin{cases} \text{BEST}_{j-1} \\ v_j + \text{BEST}_{cl(j)} \end{cases}$$

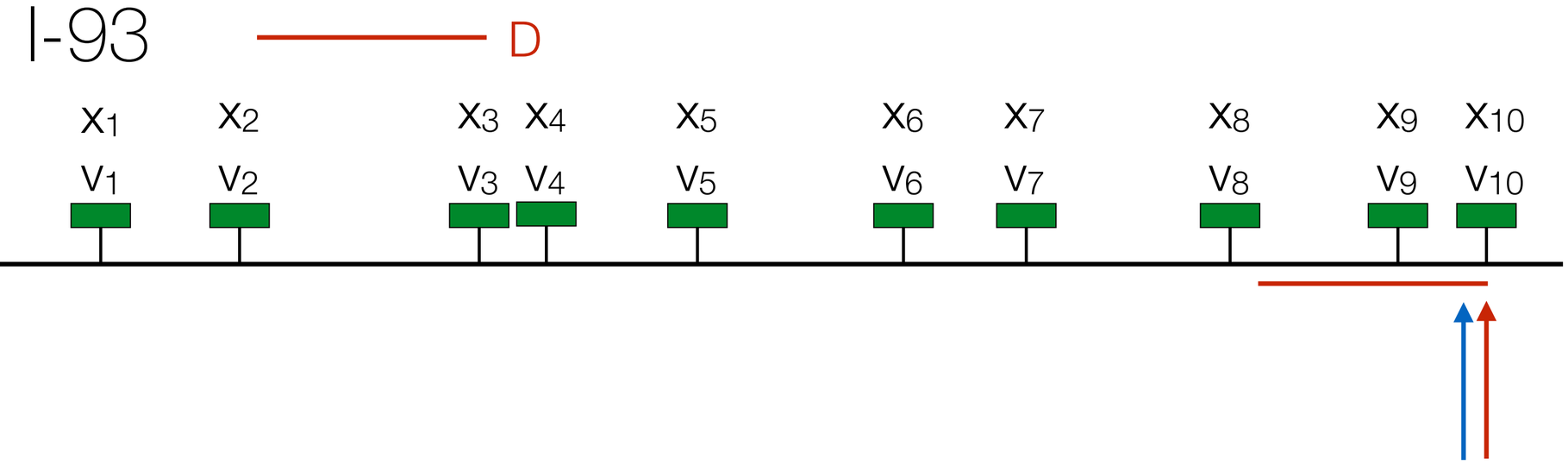
```
best[0] = 0
for i=1 to n
    cl = i-1
    while( (x[i]-x[cl])< D  && cl>0) cl=cl-1
    best[i] = max(best[i-1], vj+best[cl])
return best[n]
```

$\Theta(1)$

$\Theta(n)$

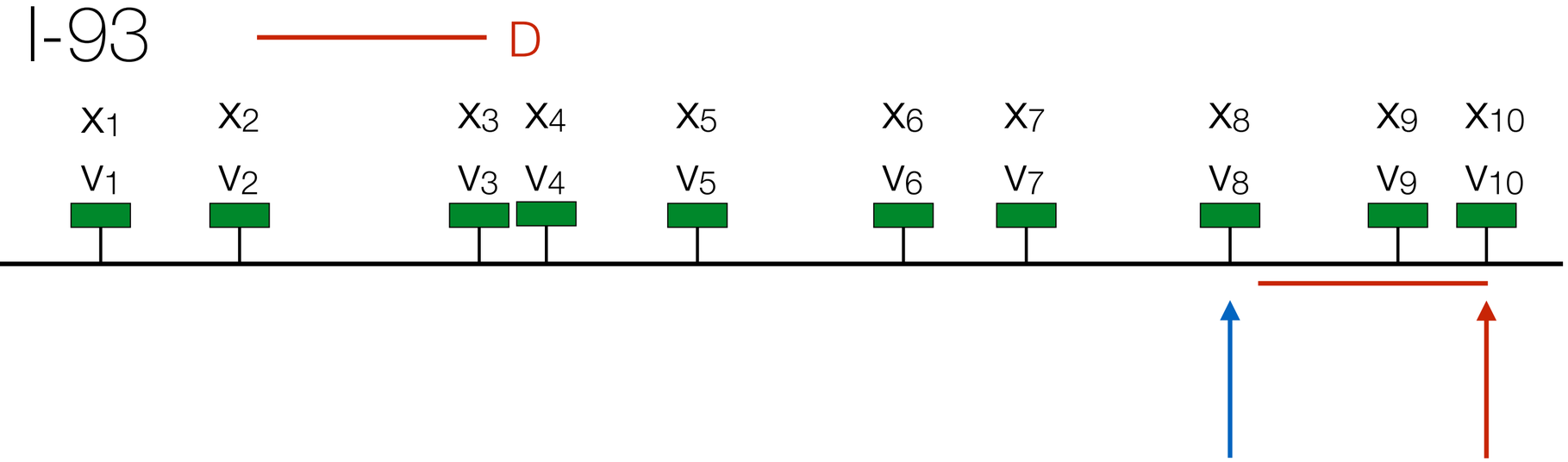
$\Theta(1)$

$\Theta(n^2)$ algorithm



Pre-process to find every board's buddy.

right = n, left = n



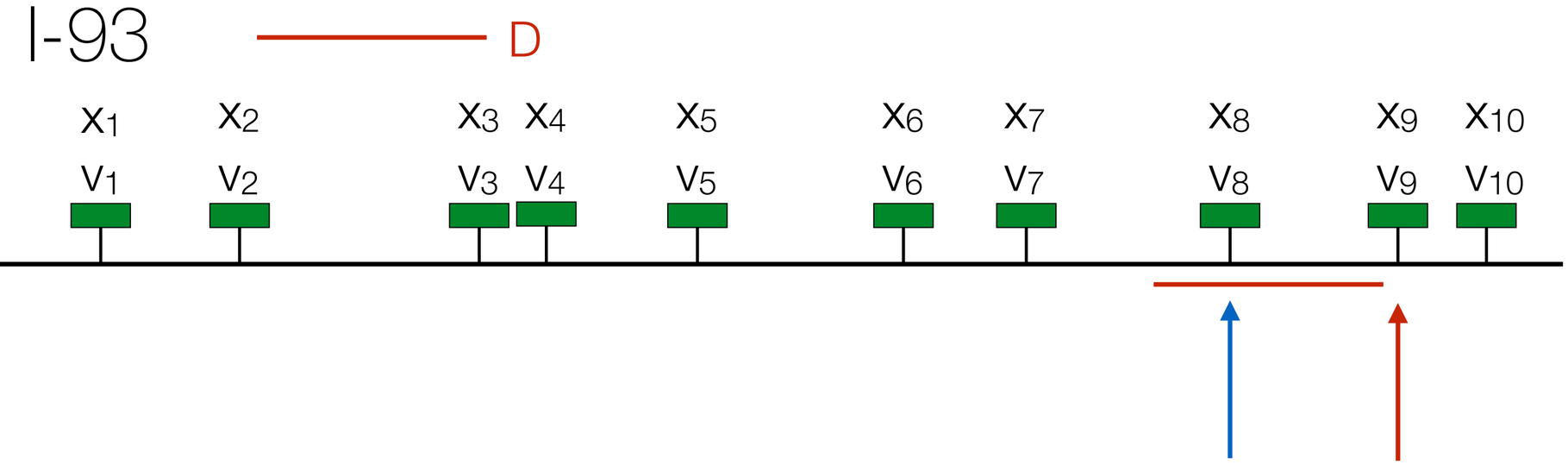
Pre-process to find every board's buddy.

right = n, left = n

move left until dist(x[right], x[left]) > D

buddy[right] = left

I-93



Pre-process to find every board's buddy.

right = n, left = n

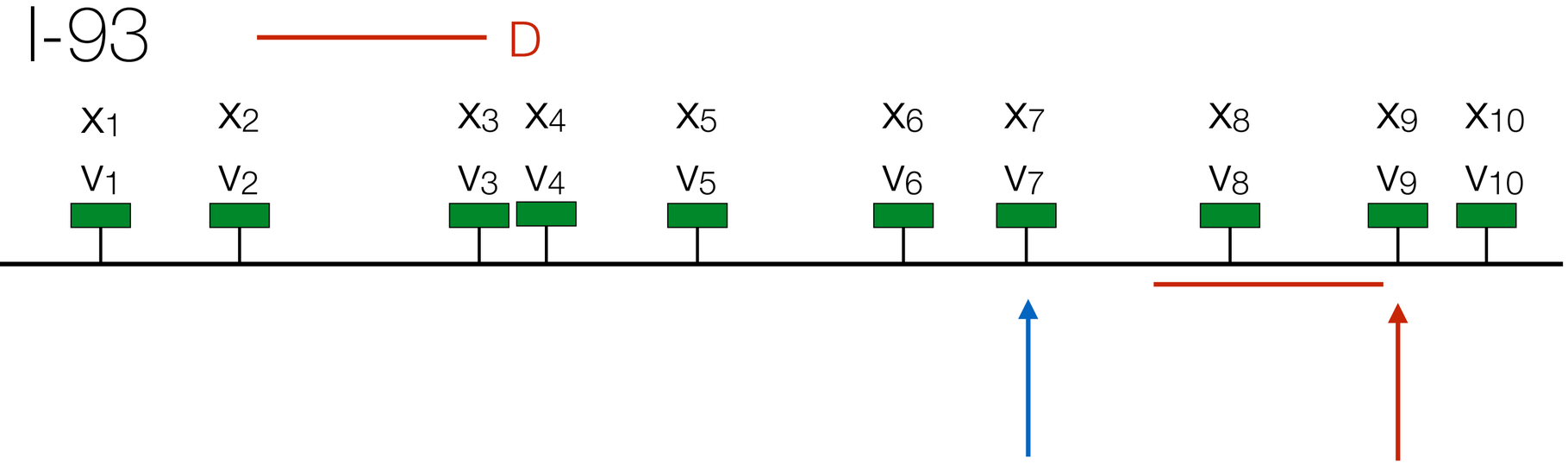
move left until dist(x[right], x[left]) > D

buddy[right] = left

move right to right

b[10]=8

I-93



Pre-process to find every board's buddy.

b[10]=8

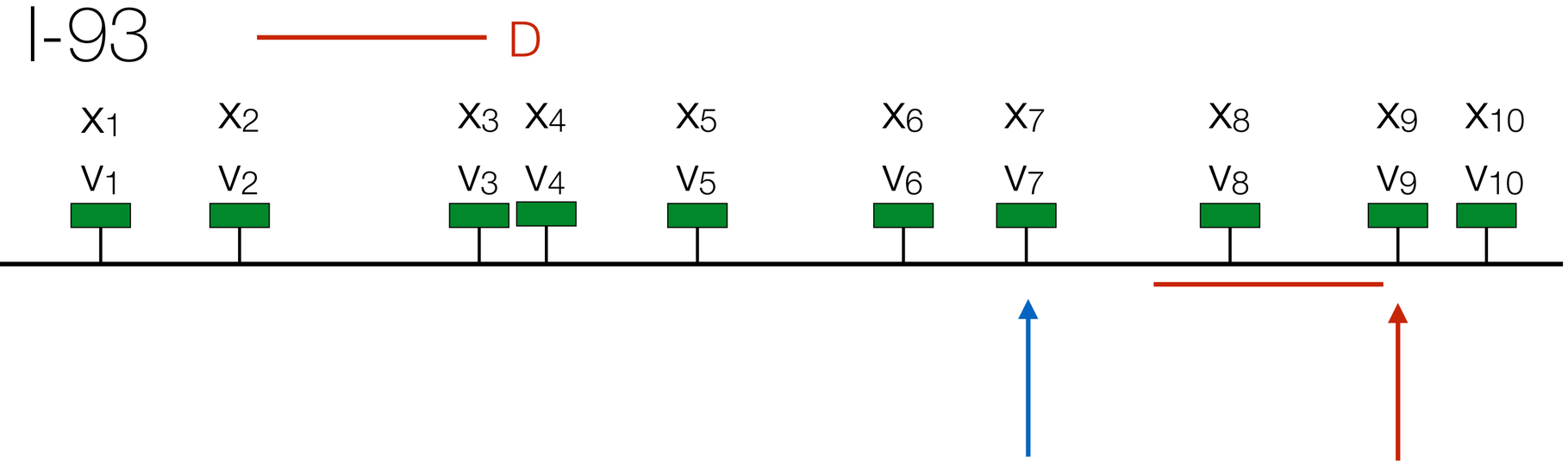
right = n, left = n

while right and left are valid

move left until dist(x[right], x[left]) > D

buddy[right] = left

move right to right



Pre-process to find every board's buddy.

b[10]=8

right = n, left = n

while right and left are valid

move left until dist(x[right], x[left]) > D

buddy[right] = left

move right to right

handle any leftover right

Better Billboard

$$\text{BEST}_j = \max \begin{cases} \text{BEST}_{j-1} \\ v_j + \text{BEST}_{cl(j)} \end{cases}$$

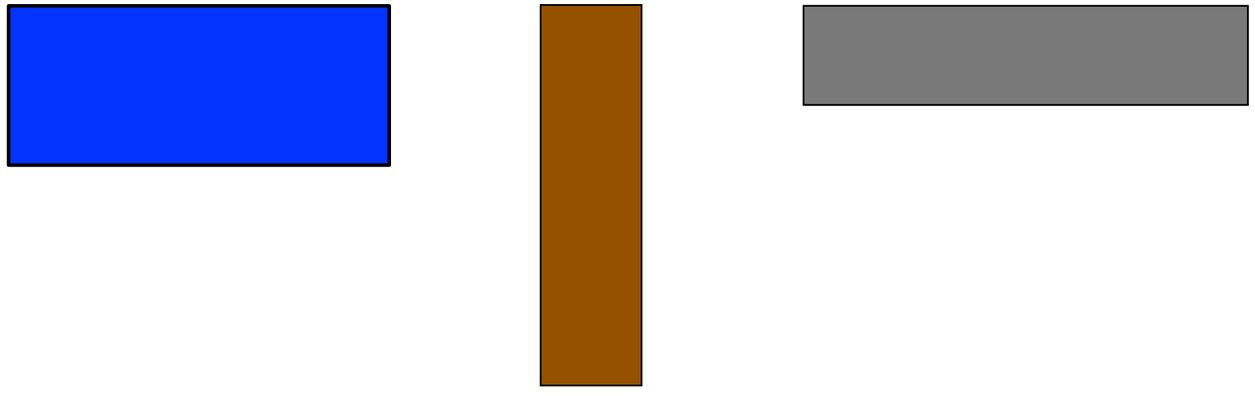
```
<Preprocess buddies>
best[0] = 0
for i=1 to n
   cl = i-1
   while( (x[i]-x[cl])< D  && cl>0) cl=cl-1
   best[i] = max(best[i-1], v[j]+best[buddy[i]])
return best[n]
```

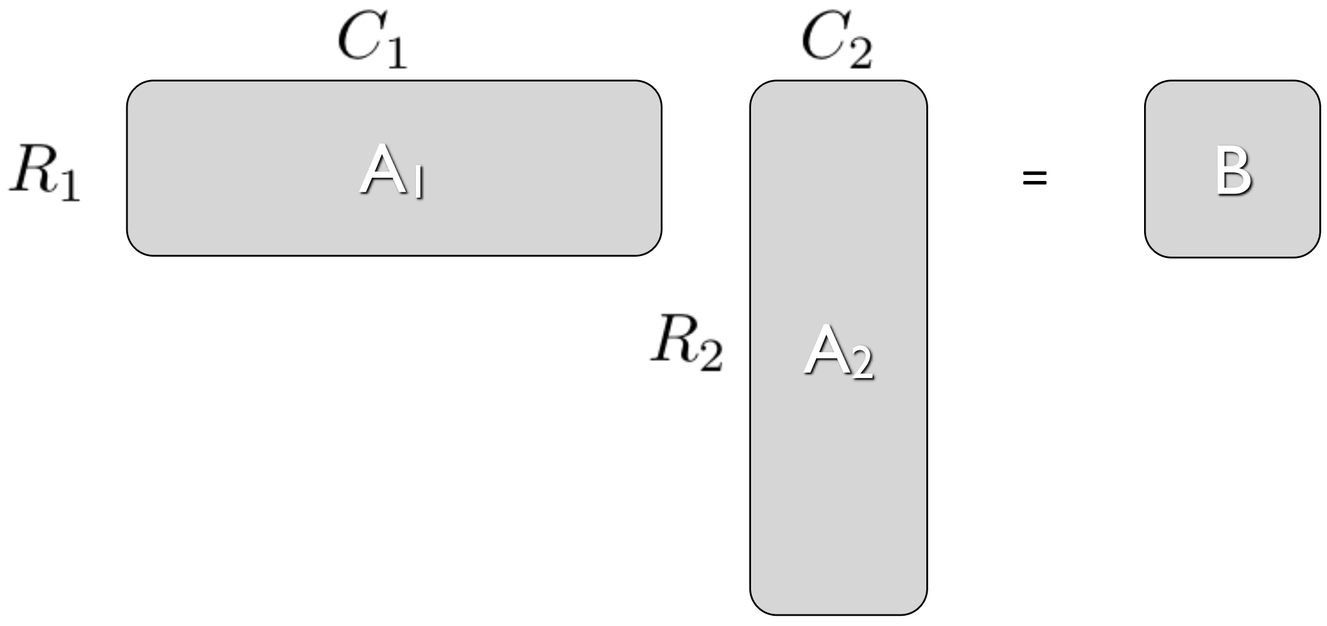
Θ(n)

Θ(n)

Θ(n)

Matrix

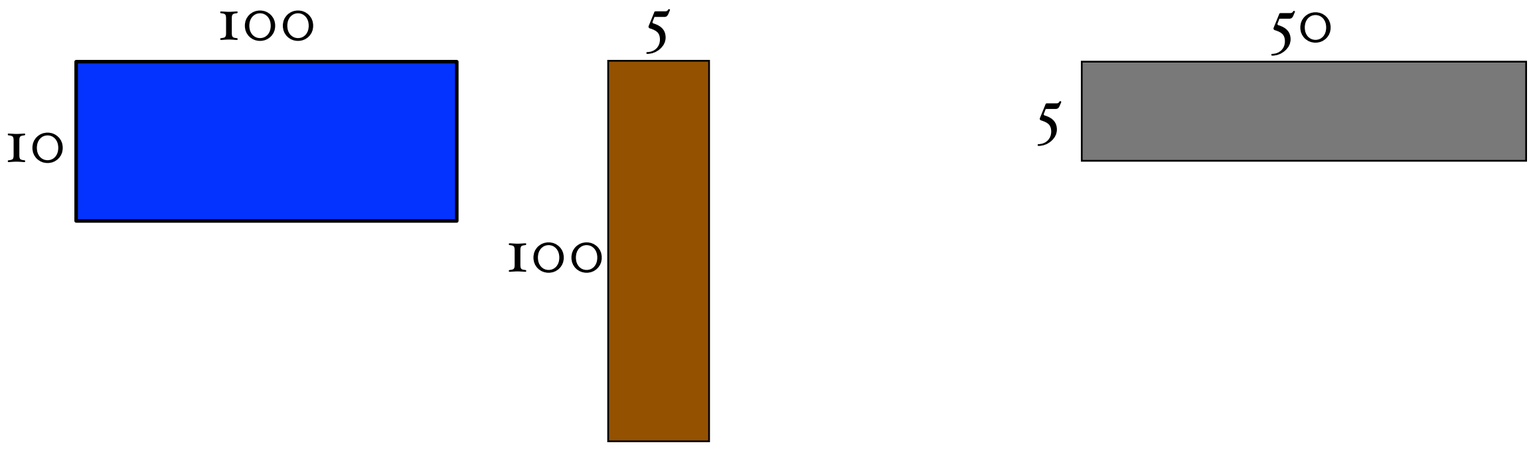



C_1
C_2
R_1
A_1
=
B
R_2
A_2

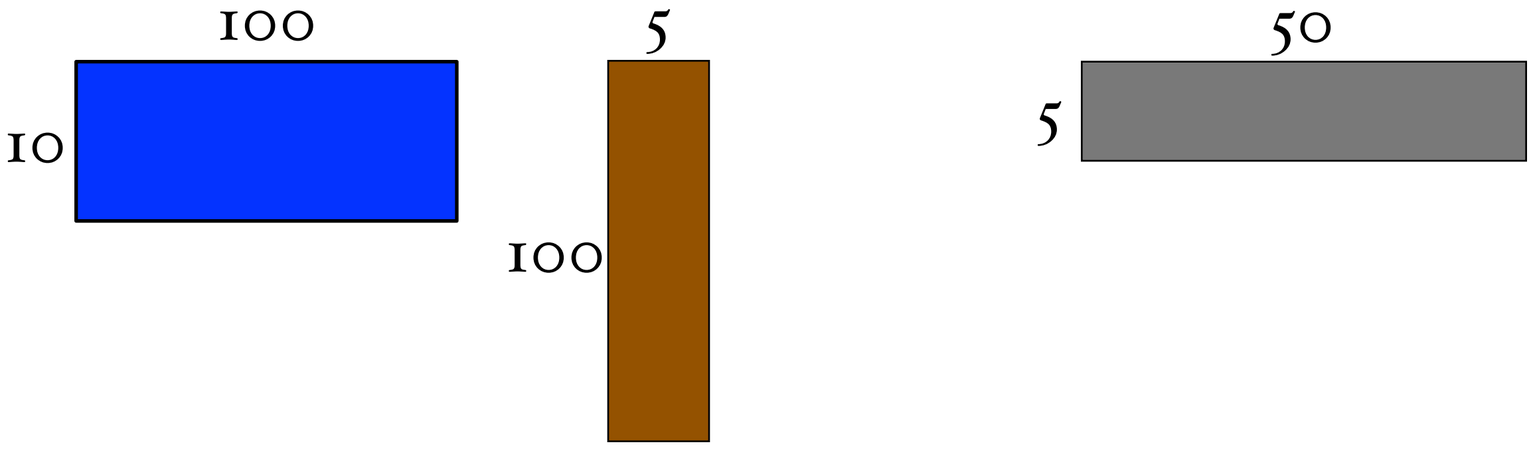
$$A_1 \cdot A_2 \cdot A_3$$

$$(A_1 \cdot A_2) \cdot A_3 \qquad A_1 \cdot (A_2 \cdot A_3)$$

$$(A_1 \cdot A_2) \cdot A_3$$



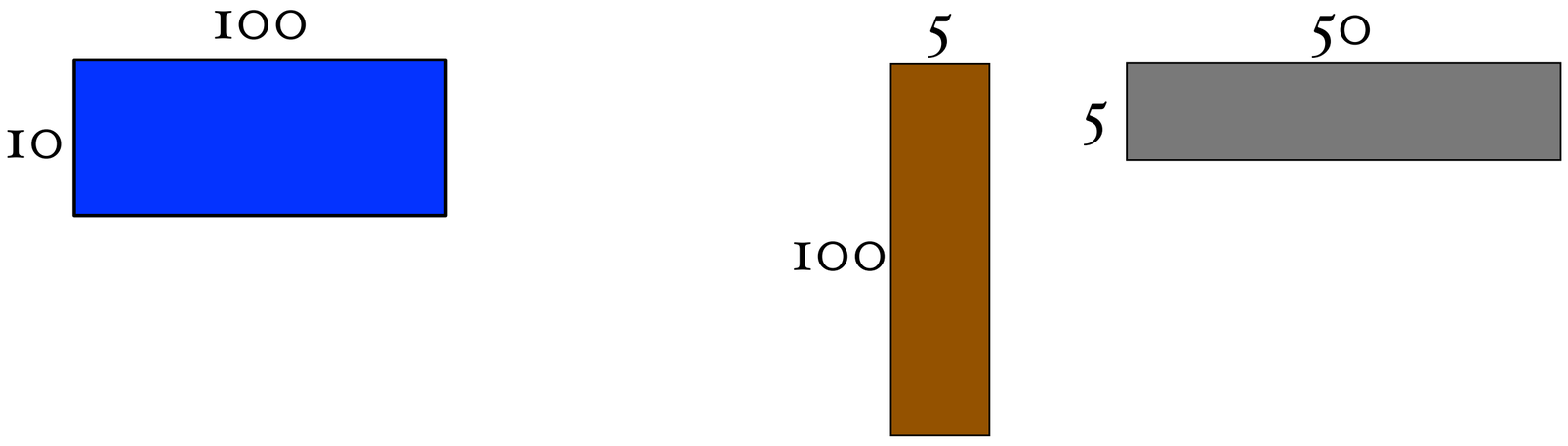
$$(A_1 \cdot A_2) \cdot A_3$$



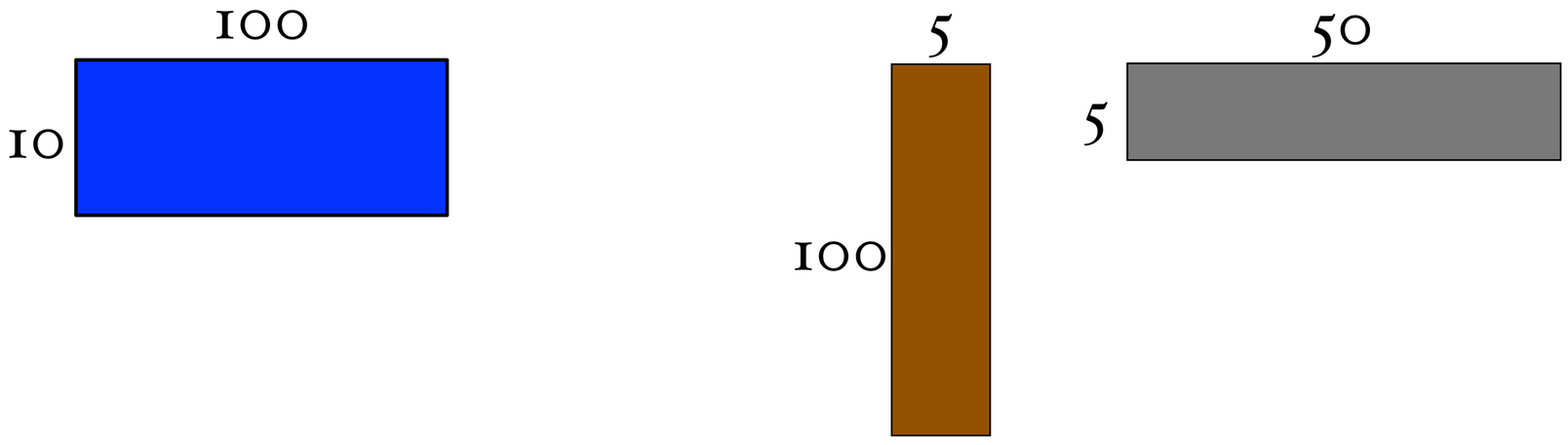
$$10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50$$

operations

$$A_1 \cdot A_2 \cdot A_3$$



$$A_1 \cdot A_2 \cdot A_3$$



$$100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50$$

operations

order matters

(for efficiency)

how many ways to compute?

$$A_1 A_2 A_3 \ldots A_n$$

how many ways to compute?

$A_1 A_2 A_3 \ldots A_n$

$A_1 A_2 A_3 \ldots A_n$

how many ways to compute?

$A_1 A_2 A_3 \ldots A_n$

$A_1 A_2 A_3 \ldots A_n$

$A_1 A_2 A_3 \ldots A_n$

how many ways to compute?

$A_1 A_2 A_3 \ldots A_n$

$A_1 A_2 A_3 \ldots A_n$

$A_1 A_2 A_3 \ldots A_n$

how do we solve it?

identify smaller instances of the problem

devise method to combine solutions

small # of different subproblems

solved them in the right order

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

B[1,n]

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

B[1,n]

B[1,1]

B[2,n]

$R_1 C_1 C_n$

optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

B[1,n]

B[1,1]	B[1,2]	...	B[1,n-2]	B[1,n-1]
B[2,n]	B[3,n]	...	B[n-1,n]	B[n,n]

$R_1 C_1 C_n$ $R_1 C_2 C_n$ $R_1 C_{n-2} C_n$ $R_1 C_{n-1} C_n$

$$B(i,i) = 1$$

$$B(1,n) = \min \left\{ \right.$$

$$B(i,i) = 1$$

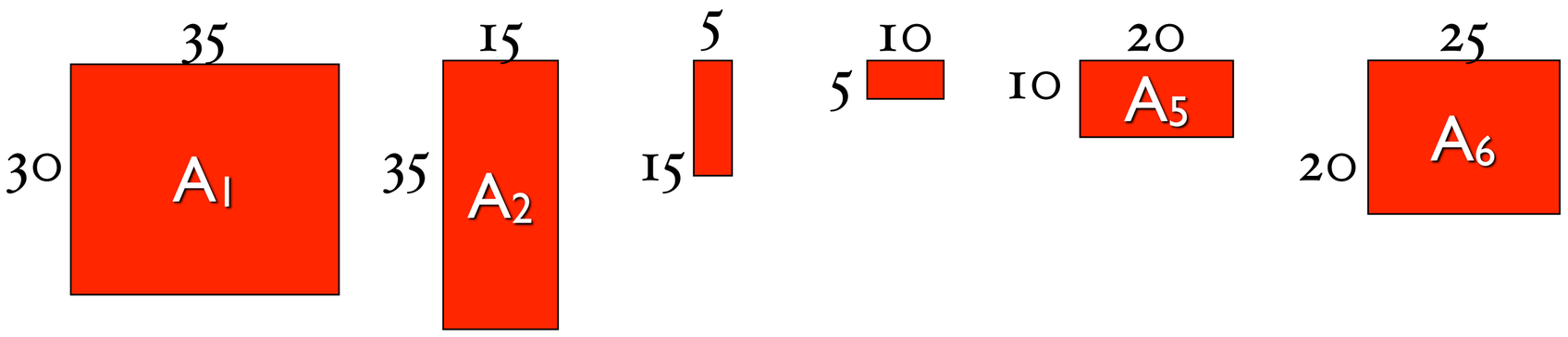
$$B(1,n) = \min \begin{cases} B(1,1) + B(2,n) + r_1 c_1 c_n \\ B(1,2) + B(3,n) + r_1 c_2 c_n \\ \vdots \\ B(1,n-1) + B(n,n) + r_1 c_{n-1} c_n \end{cases}$$

$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

$$B(i,j) =$$

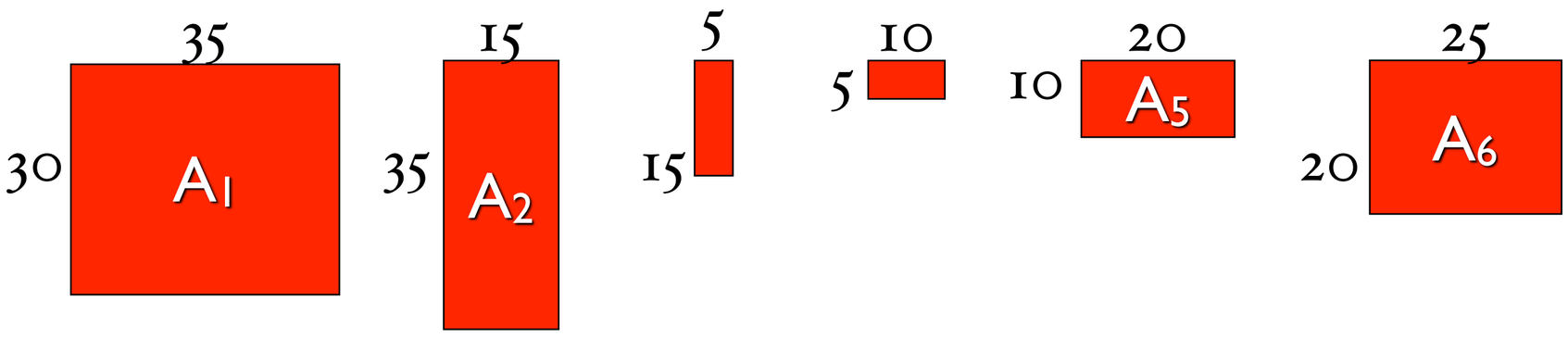
$$\begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

which order to solve?

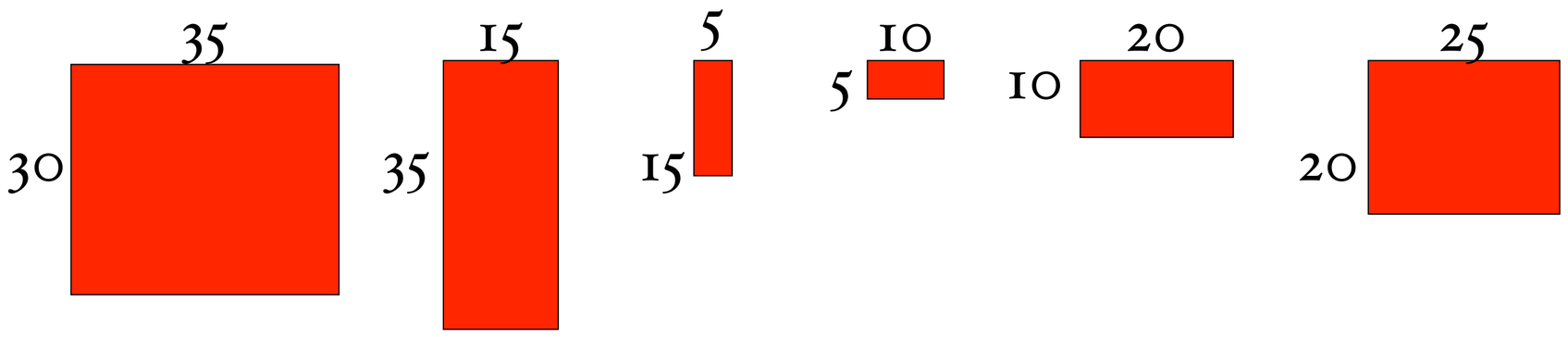


6						0
5					0	
4				0		
3			0			
2		0				
1	0					
	1	2	3	4	5	6

$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

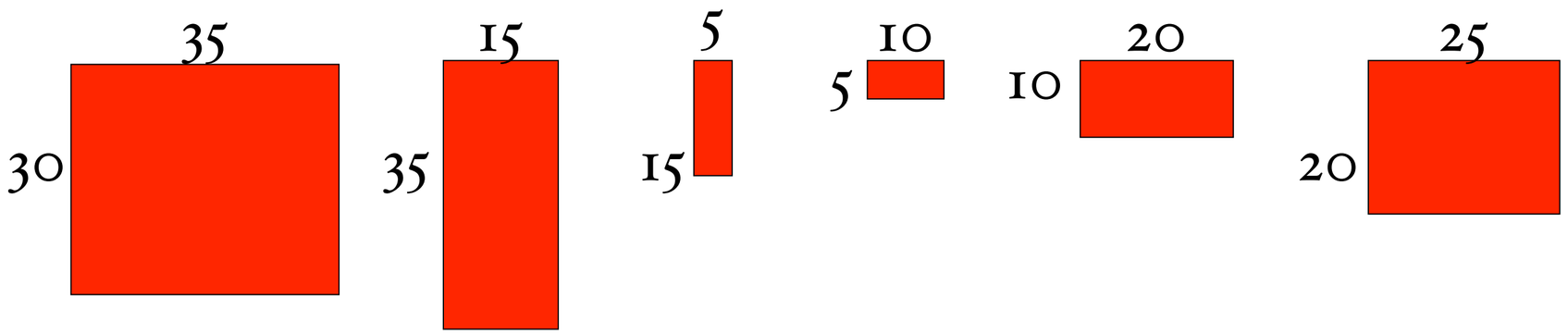


$B(1,2) =$



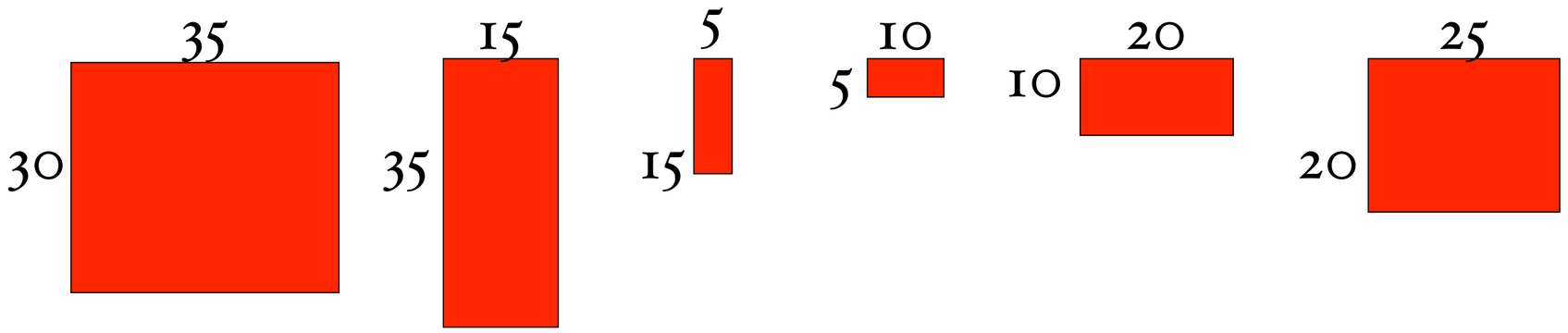
6					10*20*25 = 5000	0
5				5*10*20 = 1000	0	
4			15*5*10 = 750	0		
3		35*15*5 = 2625	0			
2	30*35*15 = 15750	0				
1	0					
	1	2	3	4	5	6

$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$



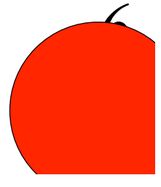
3		35*15*5 = 2625	0
2	30*35*15 = 15750	0	
1	0		
	1	2	3 4 5 6

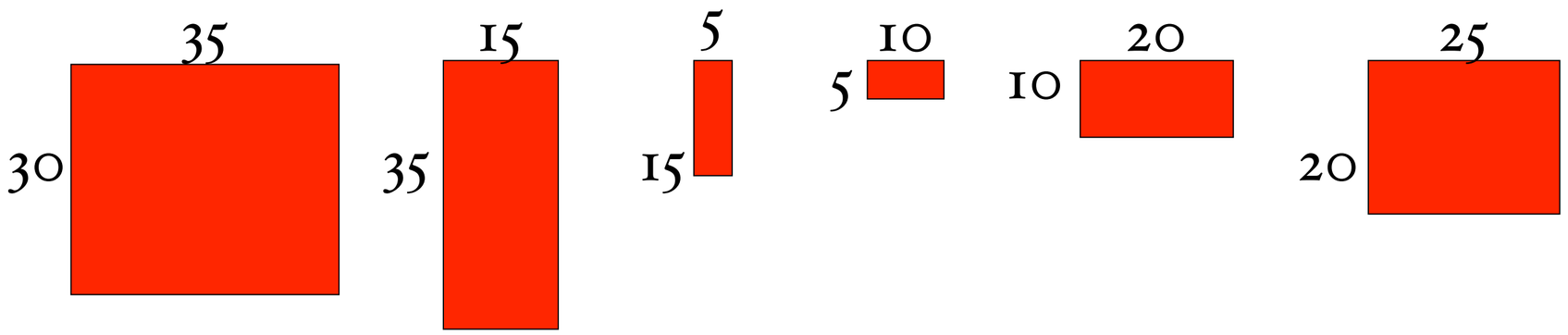
$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$



6		10500	5375	3500	10*20*25 = 5000	0
5	11875	7125	2500	5*10*20 = 1000	0	
4	9375	4375	15*5*10 = 750	0		
3	7875	35*15*5 = 2625	0			
2	30*35*15 = 15750	0				
1	0					
	1	2	3	4	5	6

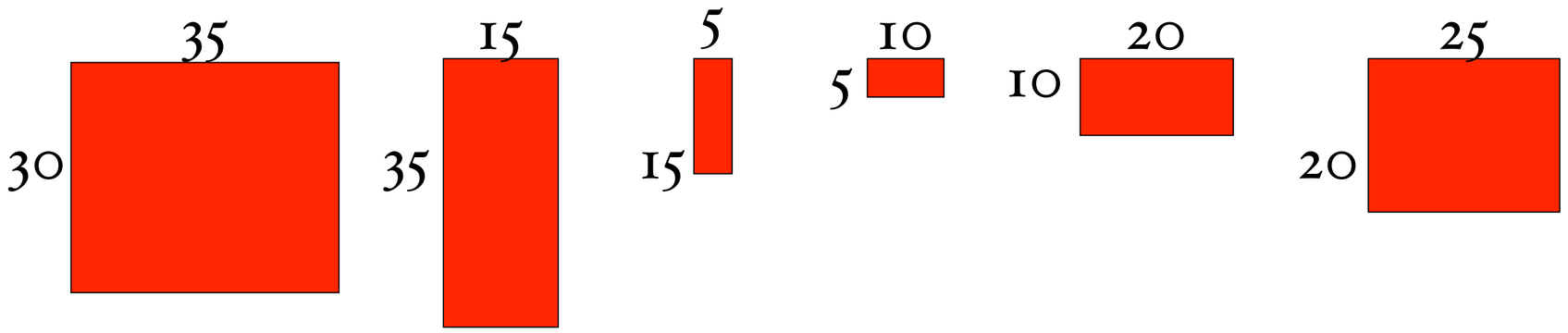
$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$





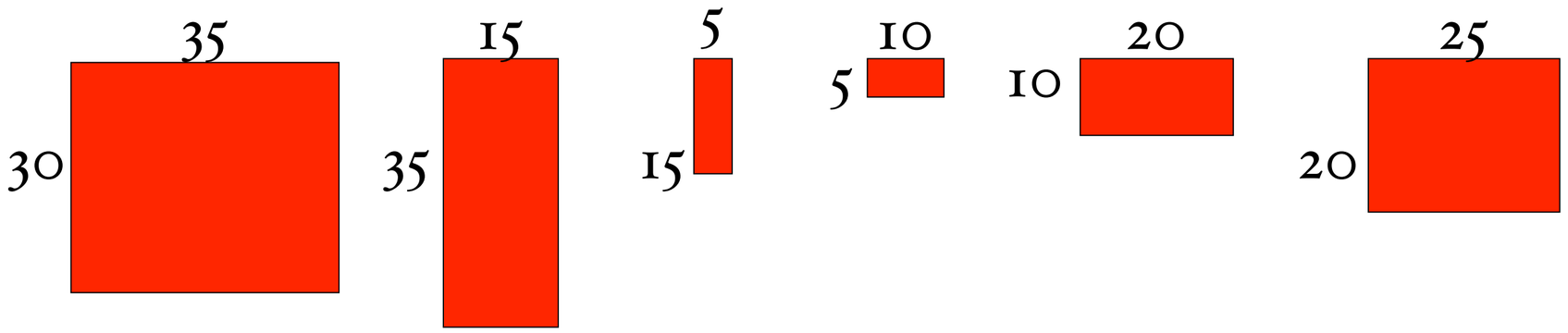
6

$$C(1,6) = \min \begin{cases} k=1 & C(1,1) + C(2,6) + r_1 c_1 c_6 \\ k=2 & C(1,2) + C(3,6) + r_1 c_2 c_6 \\ k=3 & C(1,3) + C(4,6) + r_1 c_3 c_6 \\ k=4 & C(1,4) + C(5,6) + r_1 c_4 c_6 \\ k=5 & C(1,5) + C(6,6) + r_1 c_5 c_6 \end{cases}$$



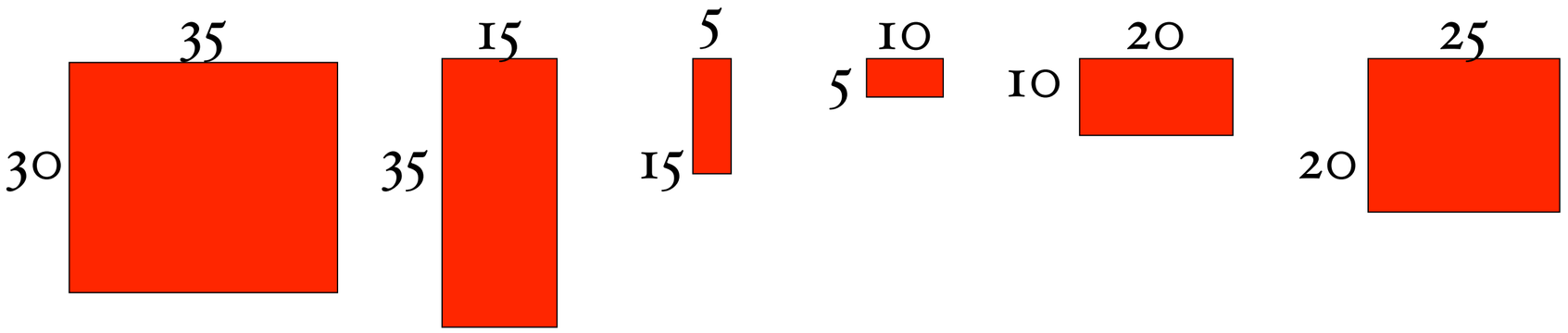
6

$$C(1,6) = \min \begin{cases} k=1 & 0 + 10500 + 30 \cdot 35 \cdot 25 \\ k=2 & 15750 + 5375 + 30 \cdot 15 \cdot 25 \\ k=3 & 7875 + 3500 + 30 \cdot 5 \cdot 25 \\ k=4 & 9375 + 5000 + 30 \cdot 10 \cdot 25 \\ k=5 & 11875 + 0 + 30 \cdot 20 \cdot 25 \end{cases}$$

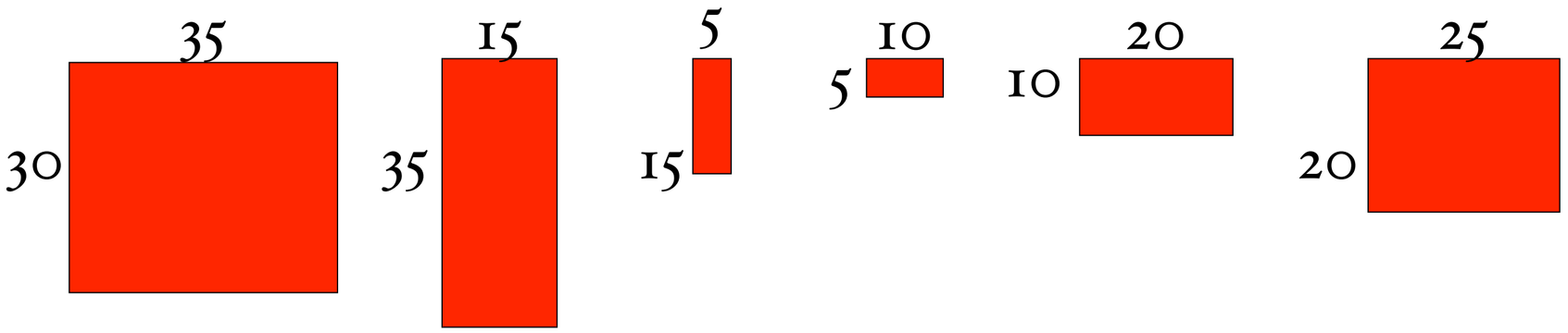


6

$$C(1,6) = \min \begin{cases} k=1 & 0 + 10500 + 26250 \\ k=2 & 15750 + 5375 + 11250 \\ k=3 & 7875 + 3500 + 3750 \\ k=4 & 9375 + 5000 + 7500 \\ k=5 & 11875 + 0 + 15000 \end{cases}$$



6	15125 3	10500	5375	3500 ★	10*20*25 = 5000	0
5	11875	7125	2500	5*10*20 = 1000	0	
4	9375	4375	15*5*10 = 750	0		
3	7875 ★	35*15*5 = 2625	0			
2	30*35*15 = 15750	0				
1	0					
	1	2	3	4	5	6



6	15125 3	10500	5375	3500 ★	10*20*25 = 5000	0
5	11875	7125	2500	5*10*20 = 1000 ★	0	
4	9375	4375	15*5*10 = 750	0		
3	7875 ★	35*15*5 = 2625 ★	0			
2	30*35*15 = 15750	0				
1	0					
	1	2	3	4	5	6

matrix-chain-mult(p)

initialize array m[x,y] to zero

matrix-chain-mult(p)

initialize array m[x,y] to zero

starting at diagonal, working towards upper-left

compute m[i,j] according to

$$\begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

running time?

initialize array m[x,y] to zero

starting at diagonal, working towards upper-left

compute m[i,j] according to

$$\begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$