£7 5800

feb 8/10 2022

shelat

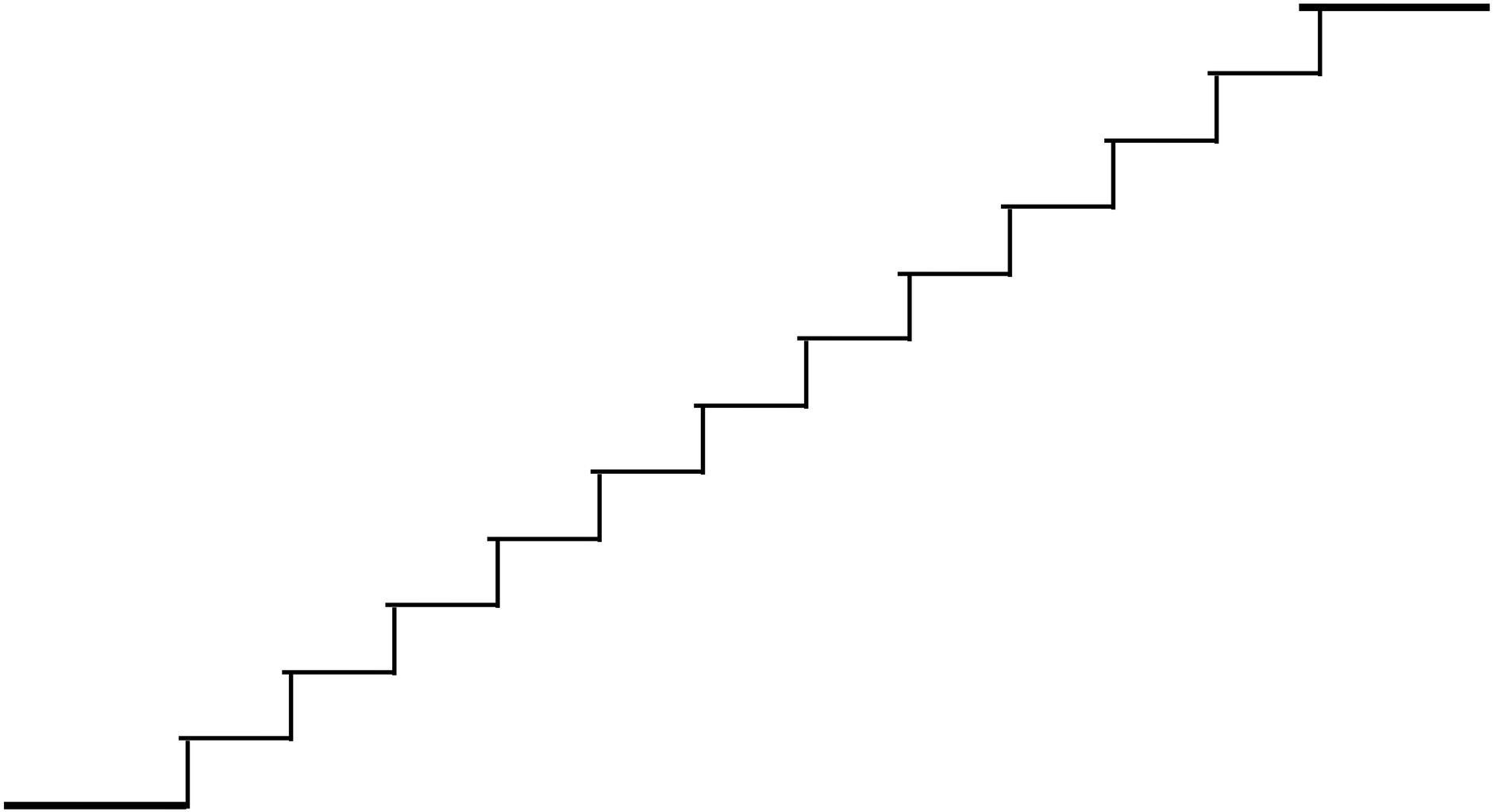
We are introducing a new algorithmic technique


West 81st Street, New York, ...
Add a photo



MIGROS

The Stairs opportunity



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

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

Stairs(n)

```
if n<=1 return 1
 Return Stairs(n-1) + Stairs(n-2)
```

Stairs(5)

Stairs(n)

```
if n<=1 return 1
 Return Stairs(n-1) + Stairs(n-2)
```

Stairs(5)

Stairs(4)

Stairs(3)

Stairs(n)

```
if n<=1 return 1
 Return Stairs(n-1) + Stairs(n-2)
```

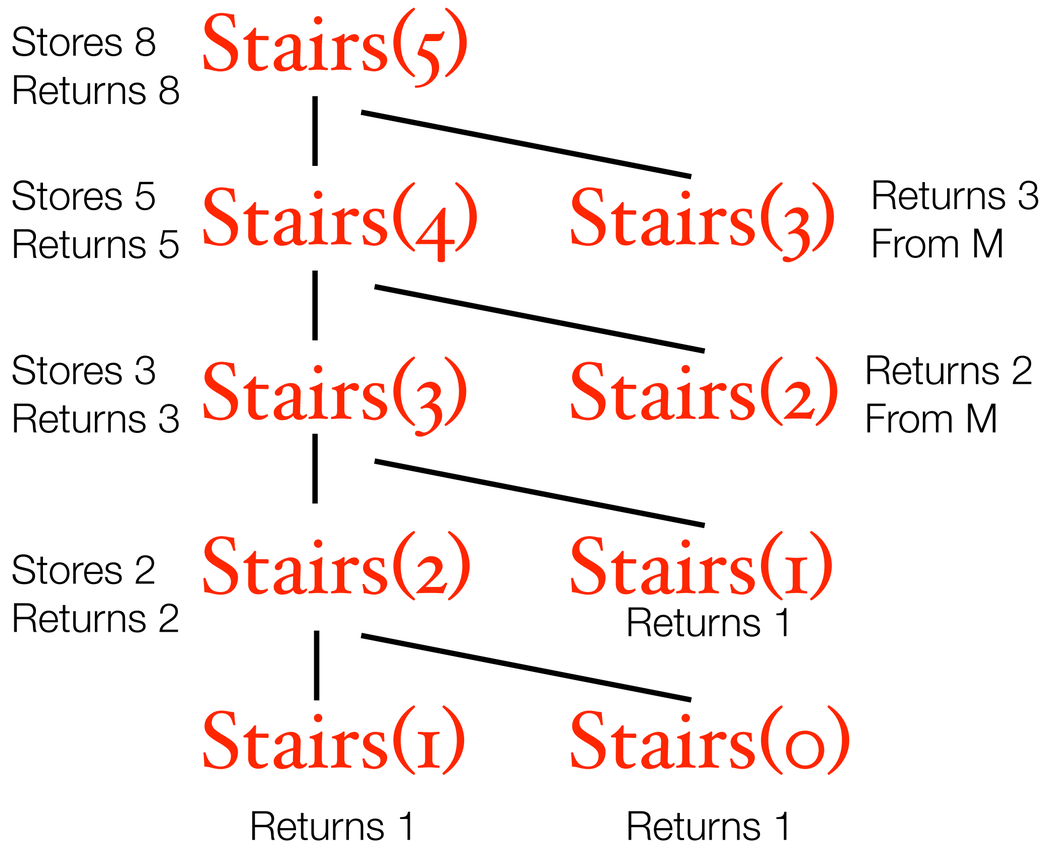
```
initialize memory M

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)
    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(n)

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

Stairs(n)

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

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

For this simple example, you might have started with this structure. But the same pattern applies to more complicated examples.

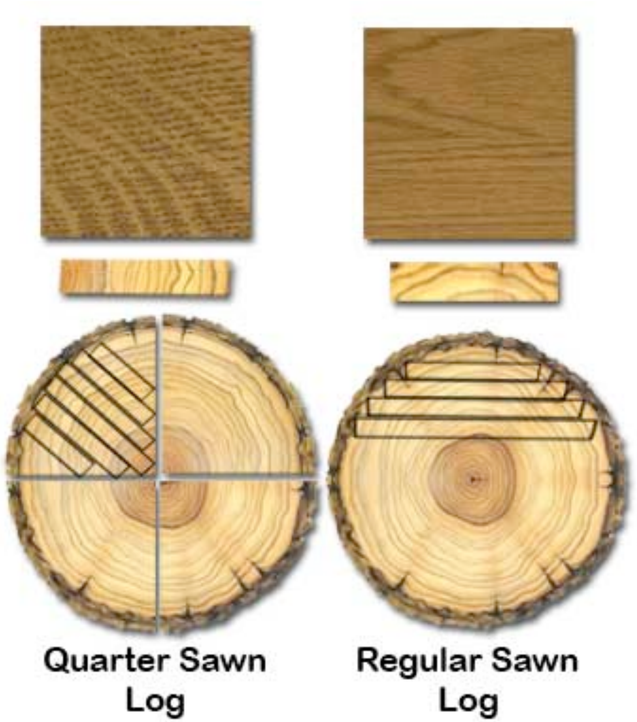
Dynamic Programming

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”

Log cutter dilemna

input to the problem: width n and prices $(p_1, \ldots, p_n)$

goal:

Log cutter dilemna

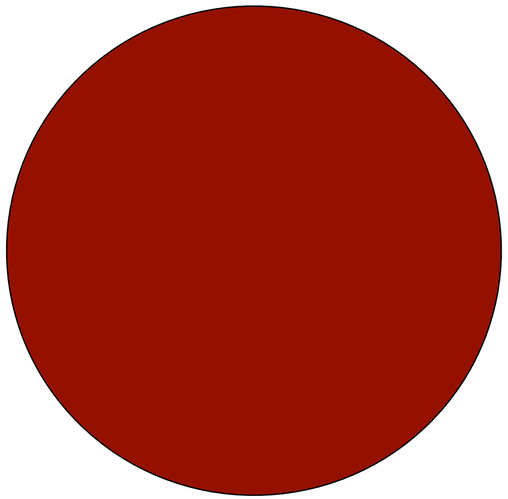
input to the problem: width n and prices $(p_1, \ldots, p_n)$

goal: Output plank widths $i_1, \ldots, i_k$ such that the sum of the plank widths is less than n which maximizes the revenue $\sum_{j=1}^{k} p_{i_j}$

Greedy fails

1”	2”	3”	4”	5”
1$	6$	7$	8$	10$

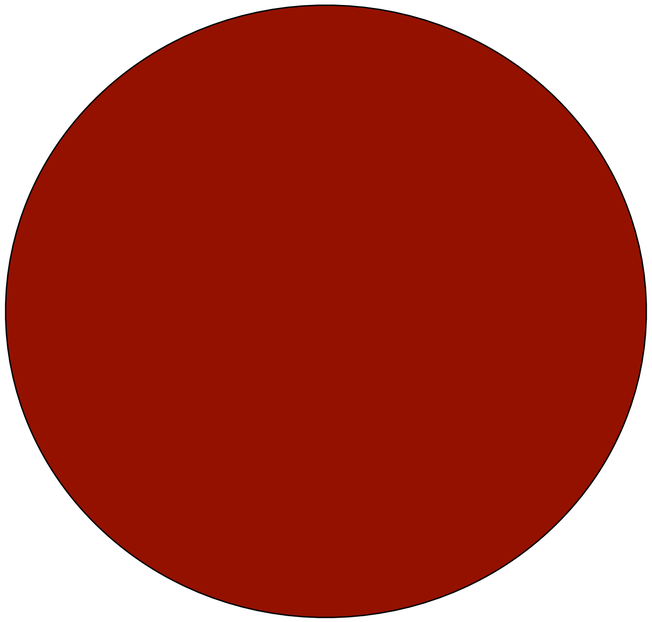
5” log



Greedy “Avg” fails

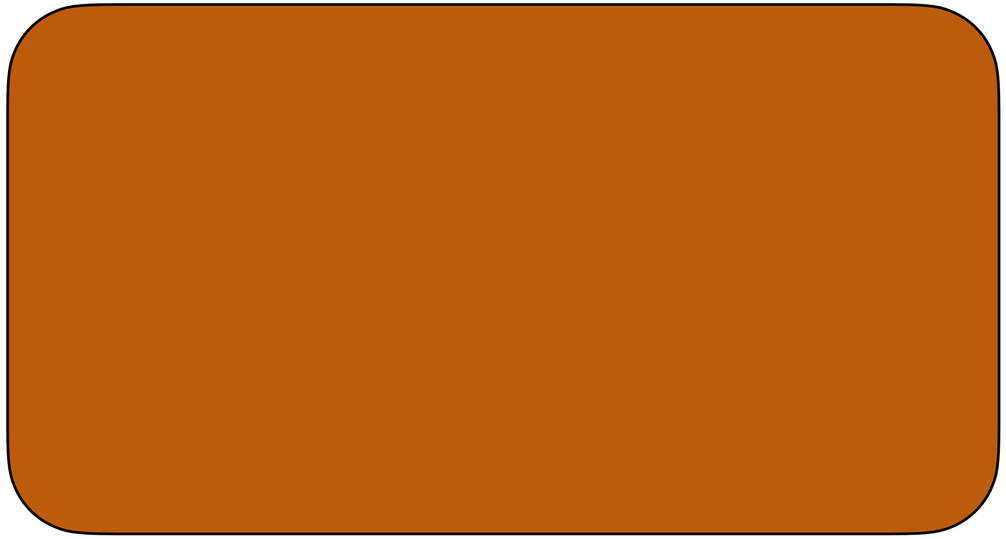
1”	2”	3”	4”	5”	6”
1$	18$	24$	36$	50$	50$

6” log



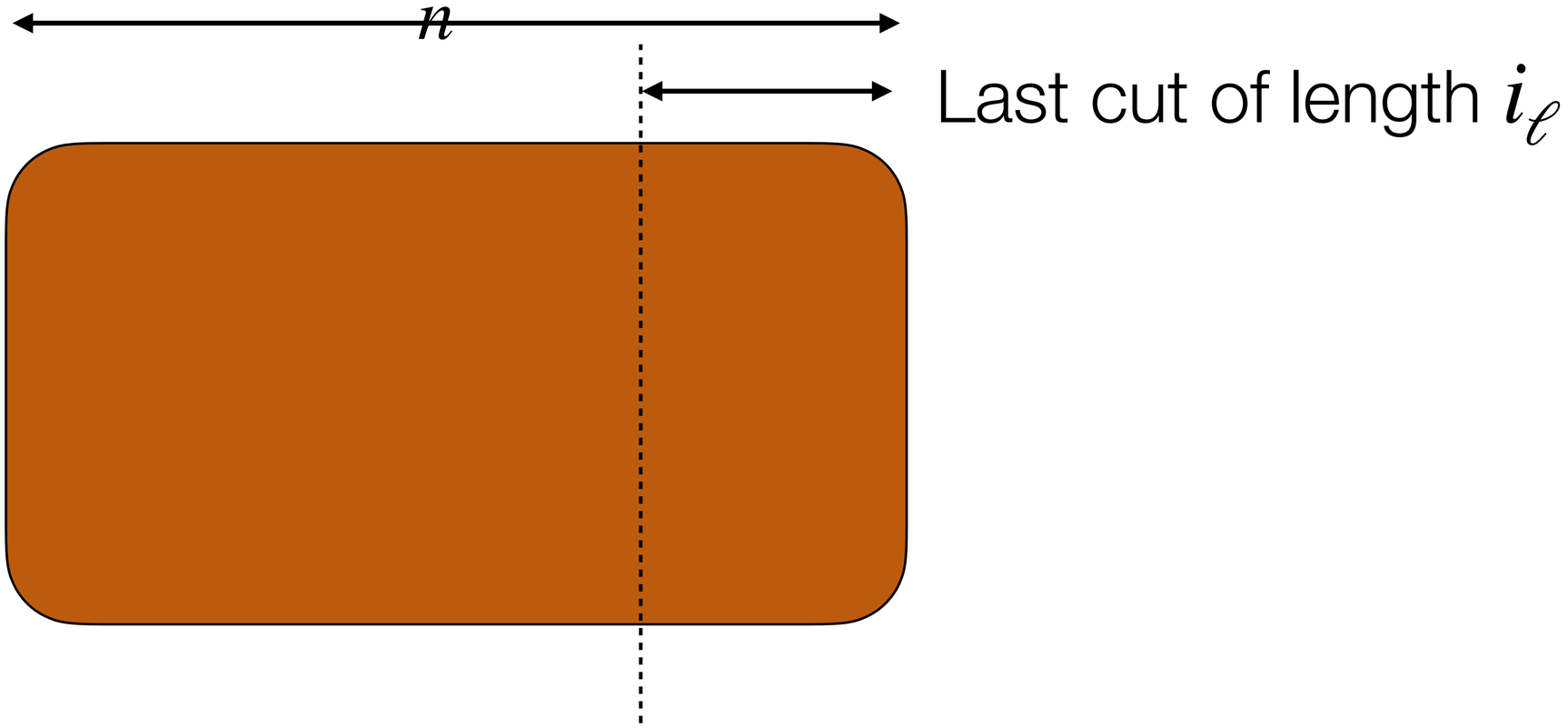
Observation

This is our log



Observation

This is our log



Think about the very last cut that is made in the optimal solution. From this, we know that the best revenue is the price of this cut plus the best solution for the rest of the log.

Solution equation

Solution equation

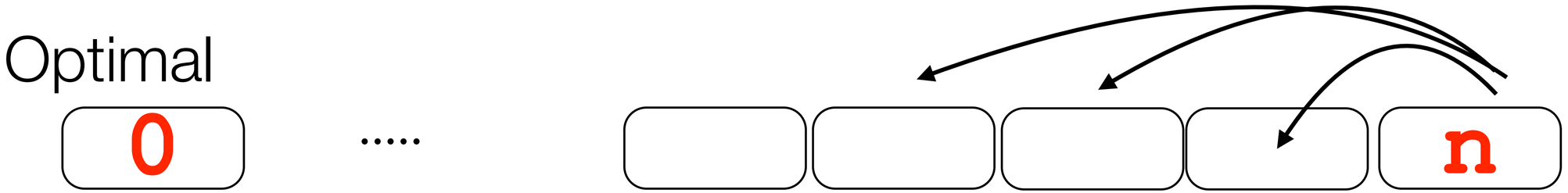
$$opt_n = \max_{1 \ldots n} \left\{ p_i + opt_{n-i} \right\}$$

The optimal revenue for a log of n is the max of the price for the last cut, plus the optimal revenue for the remainder of the log.

Approach

Optimal

Approach



opt_n depends on the optimal values for n-1, n-2, …

This suggests to build the array from 0 to n.

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

```
    if n<=0 return 0
```

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

```
if n<=0 return 0
for i=1 to n
 Best[i] = 
```

$$\max_{k=1\ldots i} \{p_k + \text{Best}[i-k]\}$$

```
return Best[n]
```

Example

1”	2”	3”	4”	5”	6”
1$	18$	24$	36$	50$	50$

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

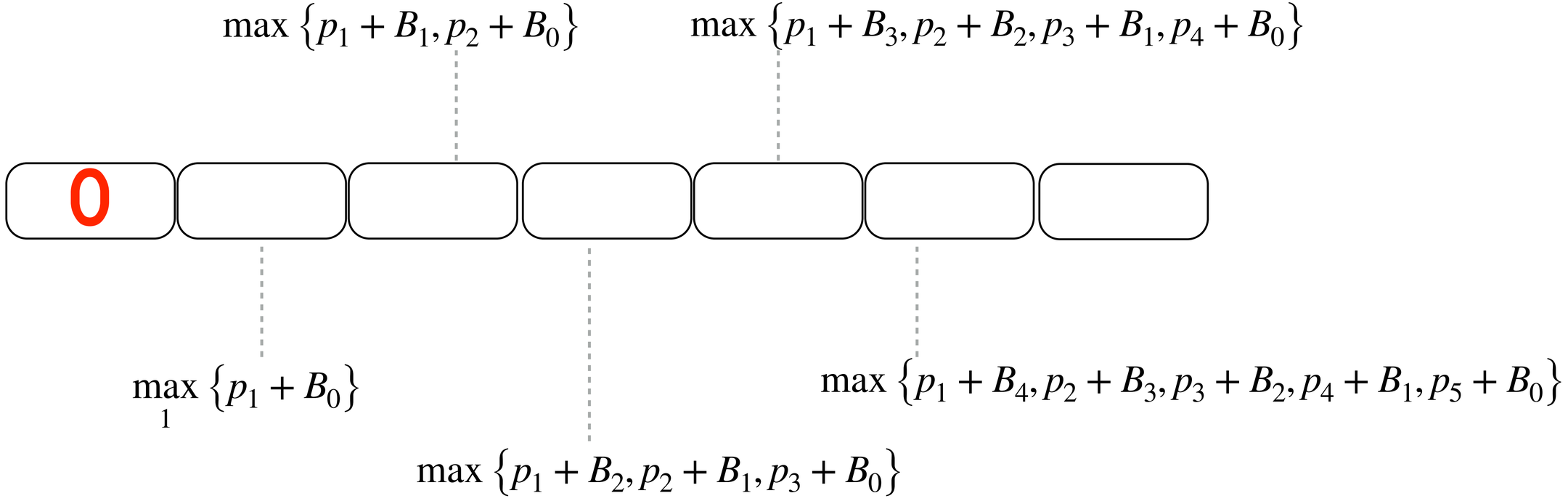
```
if n<=0 return 0
for i=1 to n
 Best[i] = max_{k=1...i} {p_k + Best[i − k]}
return Best[n]
```

Example

1"	2"	3"	4"	5"	6"
1$	18$	24$	36$	50$	50$

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

```
if n<=0 return 0
for i=1 to n
 Best[i] = max_{k=1...i} {p_k + Best[i − k]}
return Best[n]
```



The actual cuts?

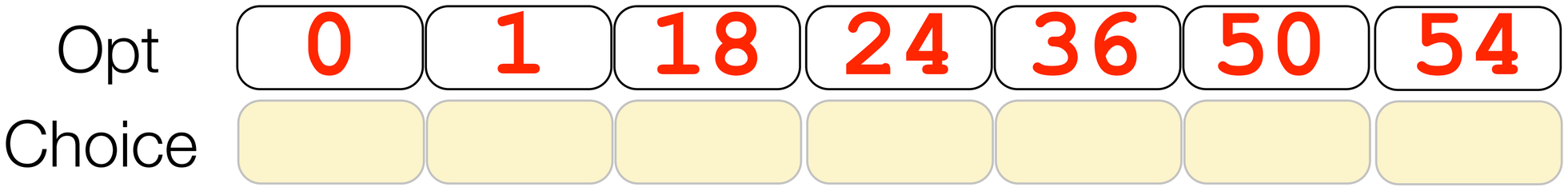
The previous algorithm just returns the optimal revenue.
How can you find the optimal cuts?

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

```
if n<=0 return 0
for i=1 to n
 Best[i] = max_{k=1...i} {p_k + Best[i − k]}
  choice[i] = k*
return Best[n]
```

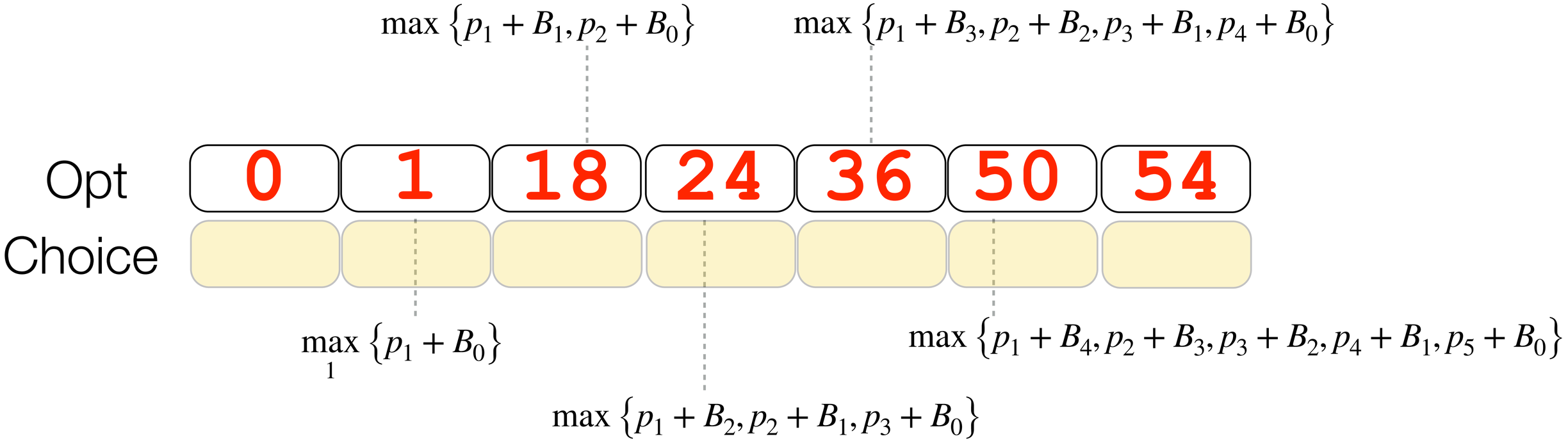
Example

1”	2”	3”	4”	5”	6”
1$	18$	24$	36$	50$	50$



Example

1”	2”	3”	4”	5”	6”
1$	18$	24$	36$	50$	50$



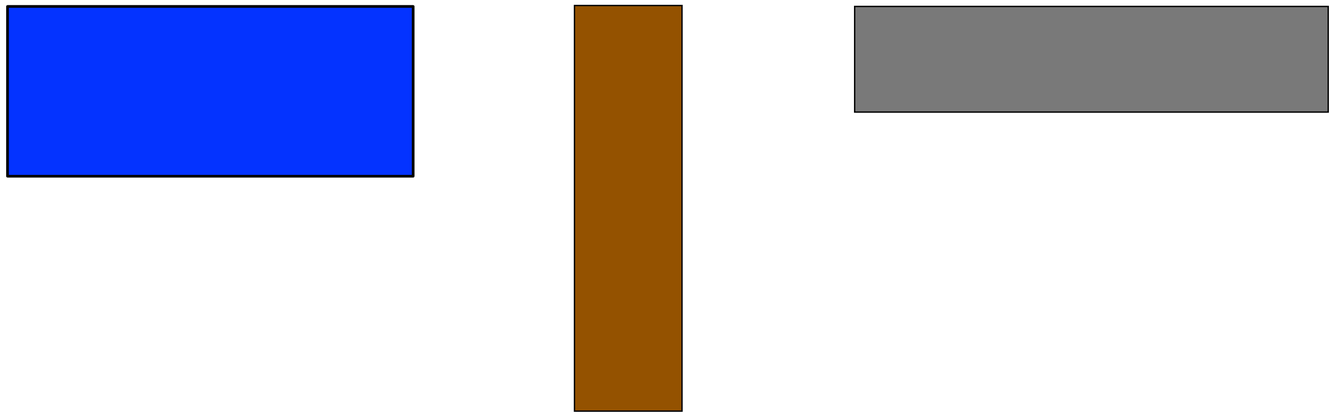
Main ideas

Identify the recursive structure of the problem with an equation.

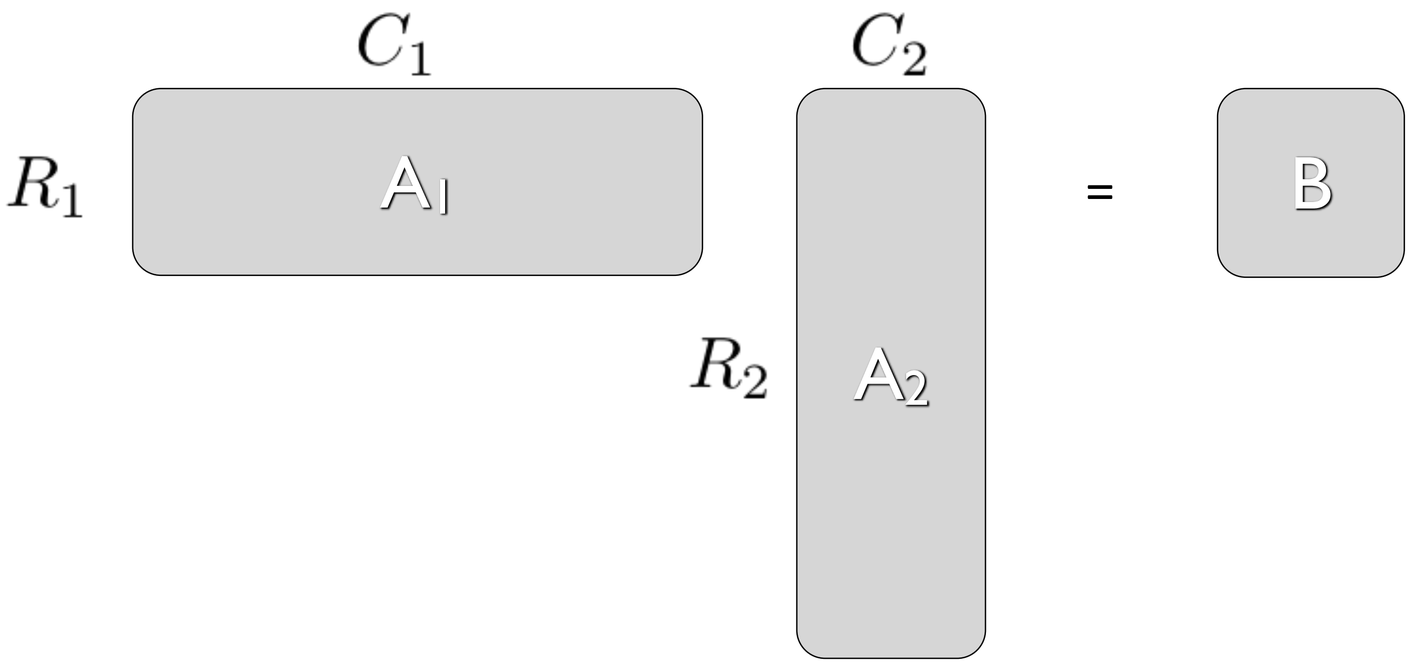
Design a way to compute this equation through an iterative loop that follows a good order.

Use another variable to record optimal parameters at each step to reconstruct a solution.

Matrix



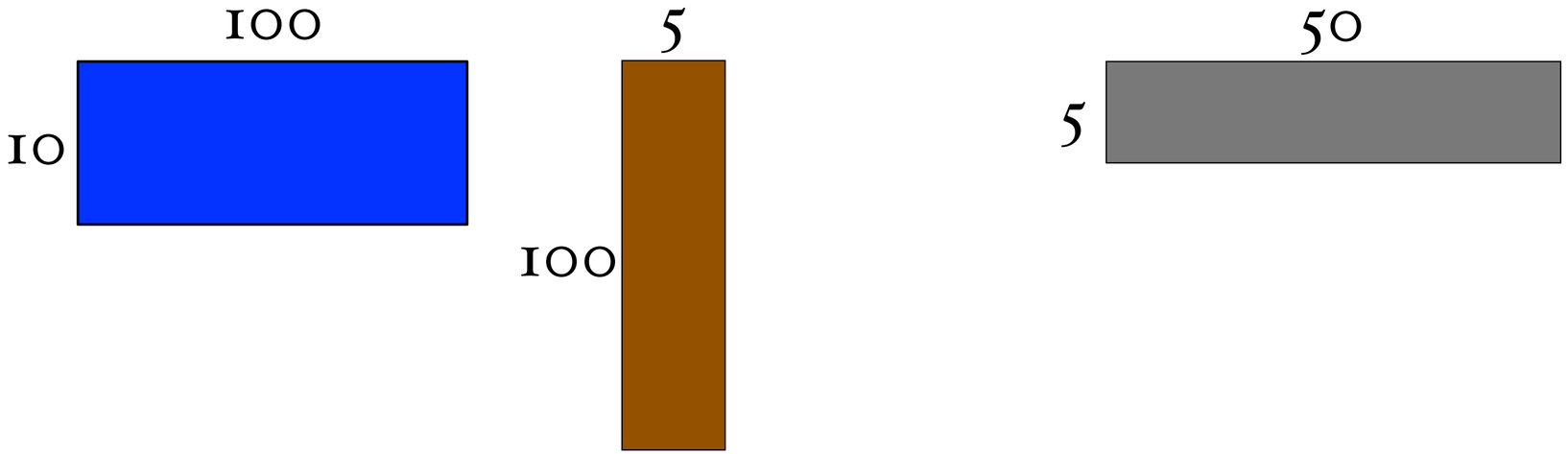
Dynamic programming in 2 dimensions!


C_1
C_2
R_1
A_1
R_2
A_2
=
B

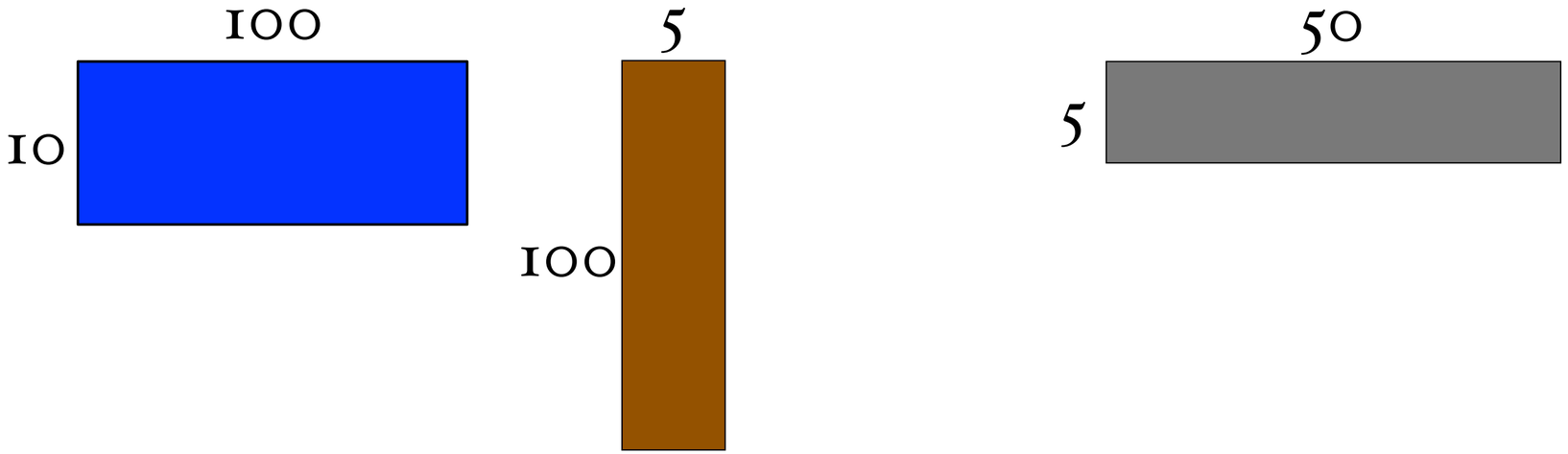
$$A_1 \cdot A_2 \cdot A_3$$

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

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



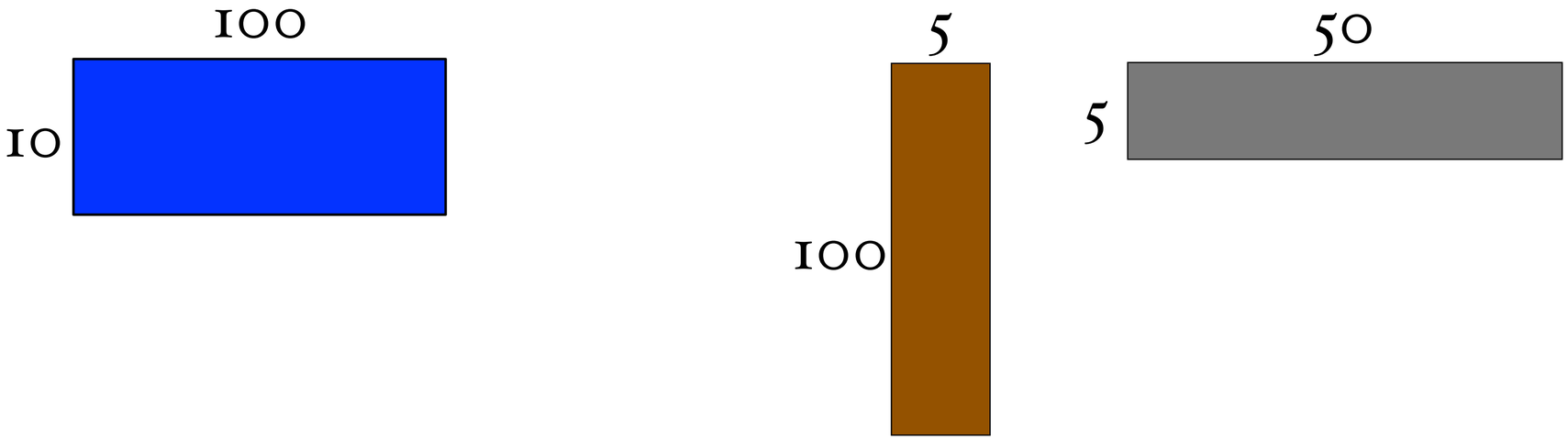
$$(A_1 \cdot A_2) \cdot A_3$$



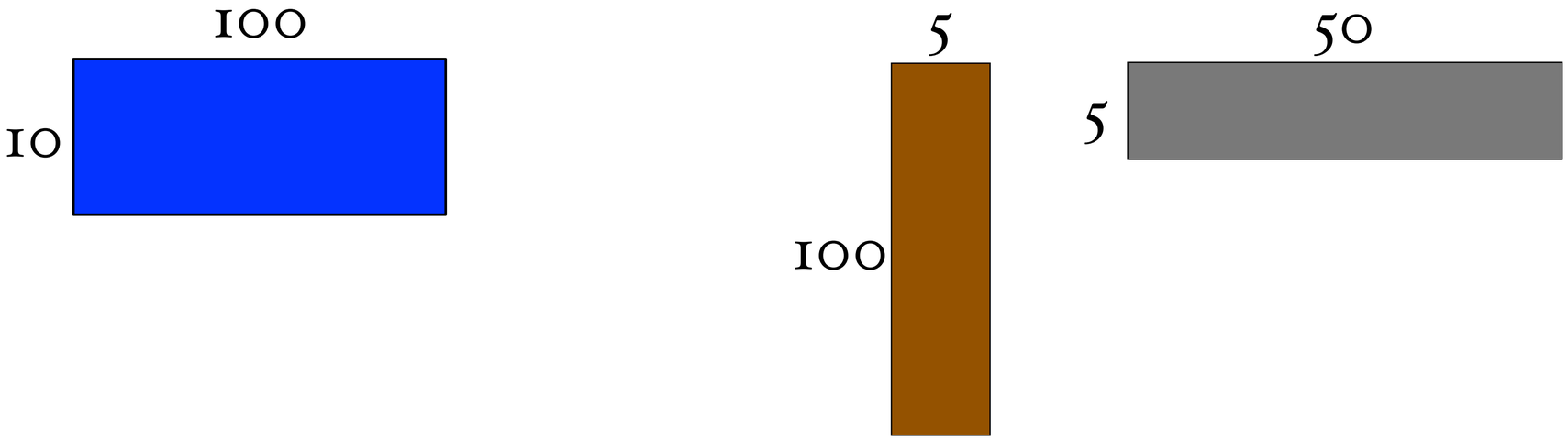
$$10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50$$

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)

Key Question: what is the best order in which to multiply the matrices?

how do we solve it?

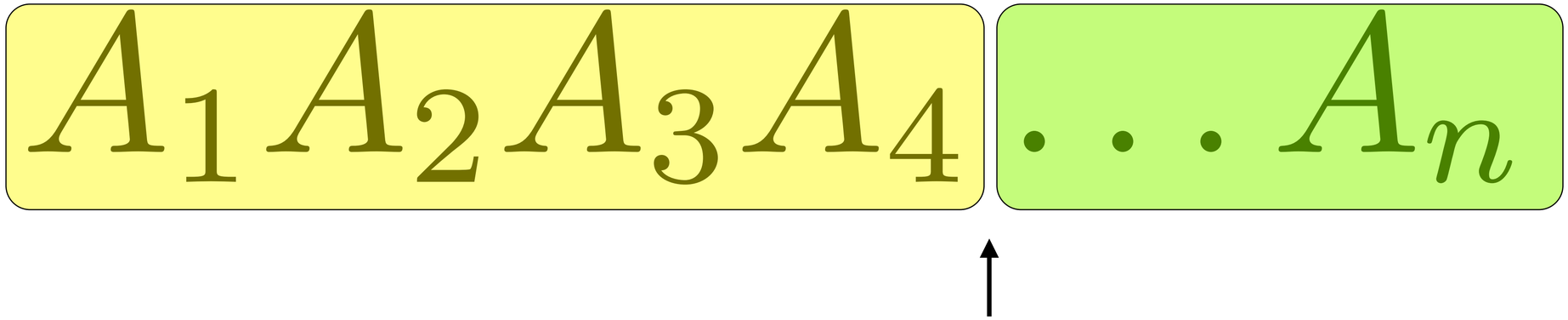
identify smaller instances of the problem

devise method to combine solutions

small # of different subproblems

solved them in the right order

optimal way to compute



Suppose this mult was the
last step in the optional order
for computing the product.

optimal way to compute

$$A_1 A_2 A_3 A_4 \quad \ldots A_n$$

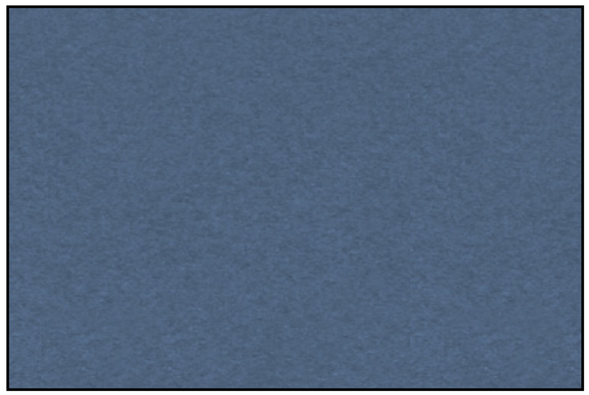
Suppose this mult was the last step in the optional order for computing the product.

$$B(1,n) = B(1,4) + B(5,n) + ?$$

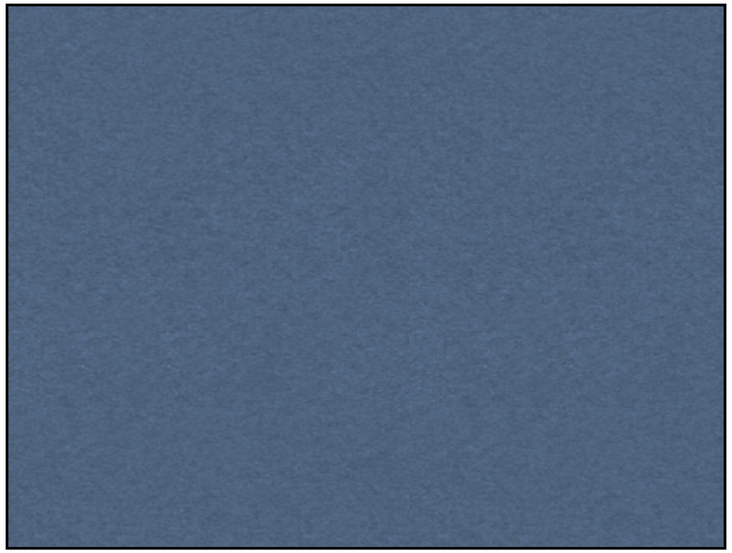
optimal way to compute

$$A_1 A_2 A_3 A_4 \; \ldots A_n$$

Size of $A_1 A_2 A_3 A_4$



Size of $A_5 \ldots A_n$

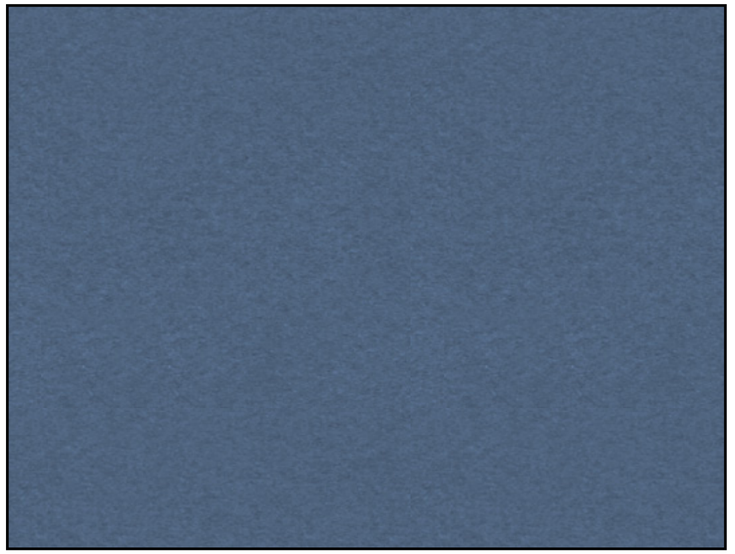


$A_1A_2A_3A_4$

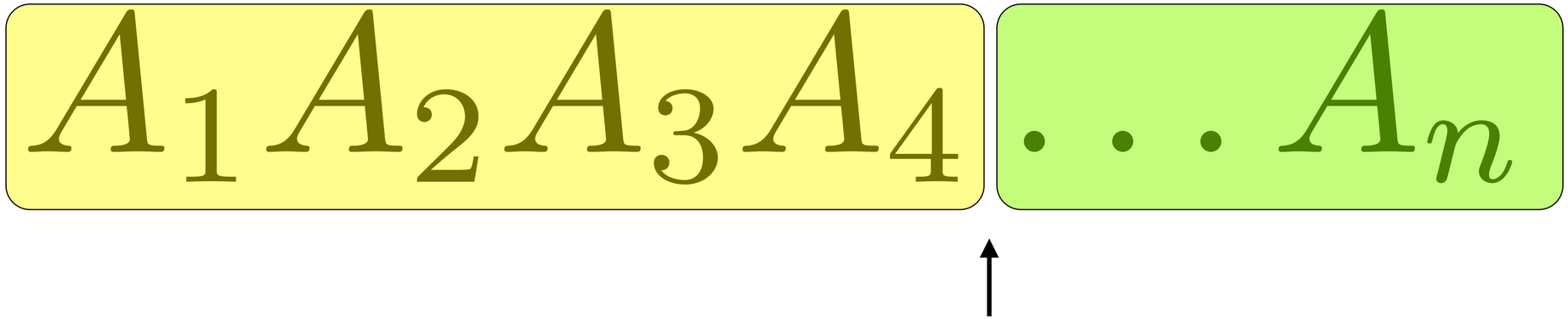
optimal way to compute

$$A_1 A_2 A_3 A_4 \ldots A_n$$

Cost of multiplying $A_1 A_2 A_3 A_4$ and $A_5 \ldots A_n$

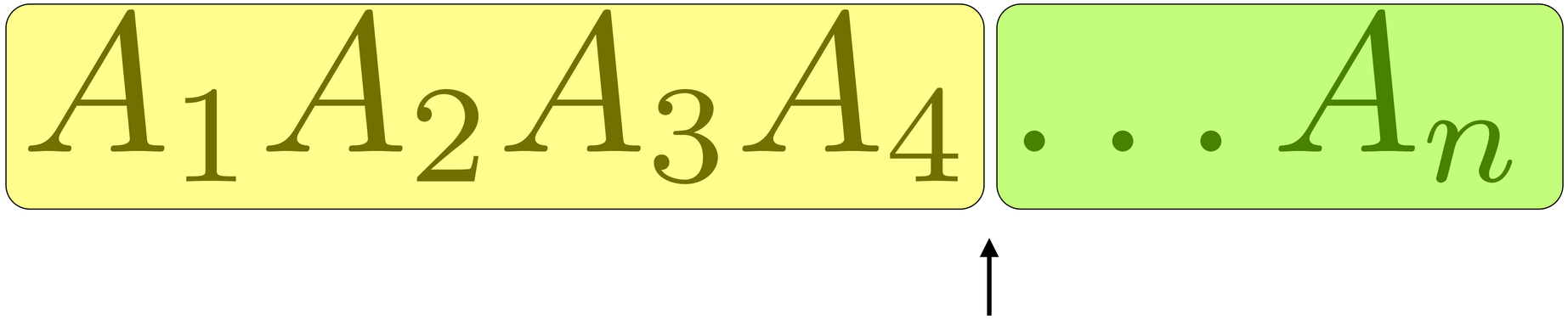


optimal way to compute



Suppose this mult was the last step in the optional order for computing the product.

optimal way to compute



Suppose this mult was the last step in the optional order for computing the product.

$$B(1,n) = B(1,4) + B(5,n) + ?$$

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]

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

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]

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

The optimal is the minimum among all of these choices.

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

$$B(1,n) = \min \begin{cases} \\ \\ \\ \\ \end{cases}$$

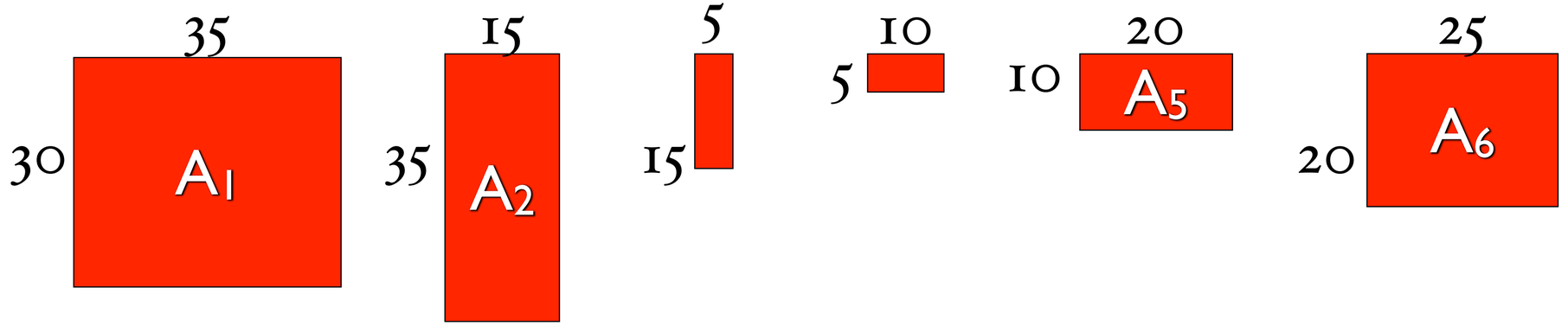
$$B(i,i) = 1$$

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

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

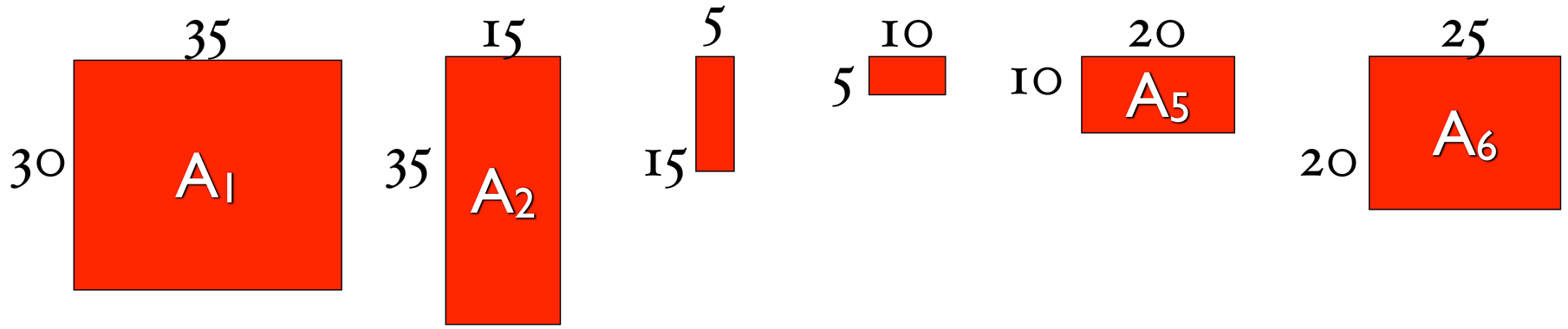
$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

which order to solve?



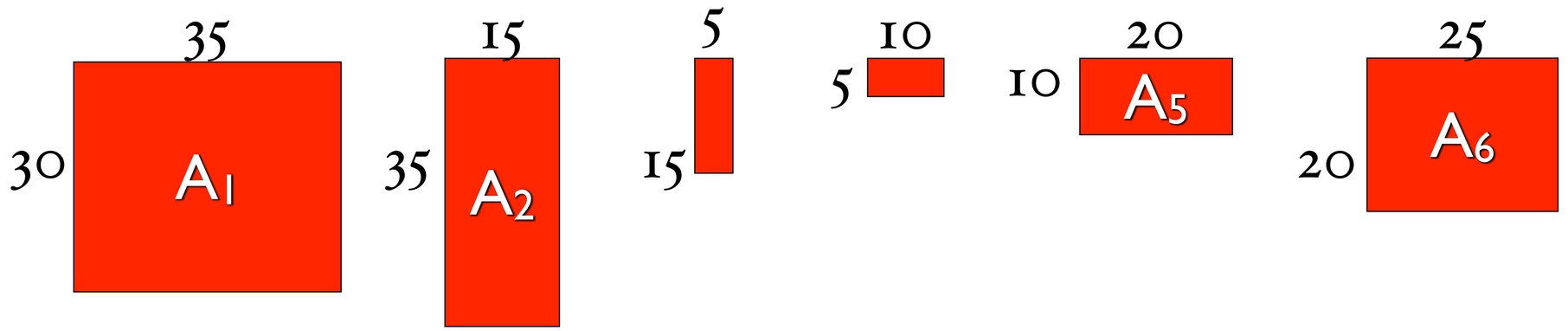
1	6					0
	5				0	
	4			0		
	3		0			
	2	0				
	1	0				
		1	2	3	4	5	6

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



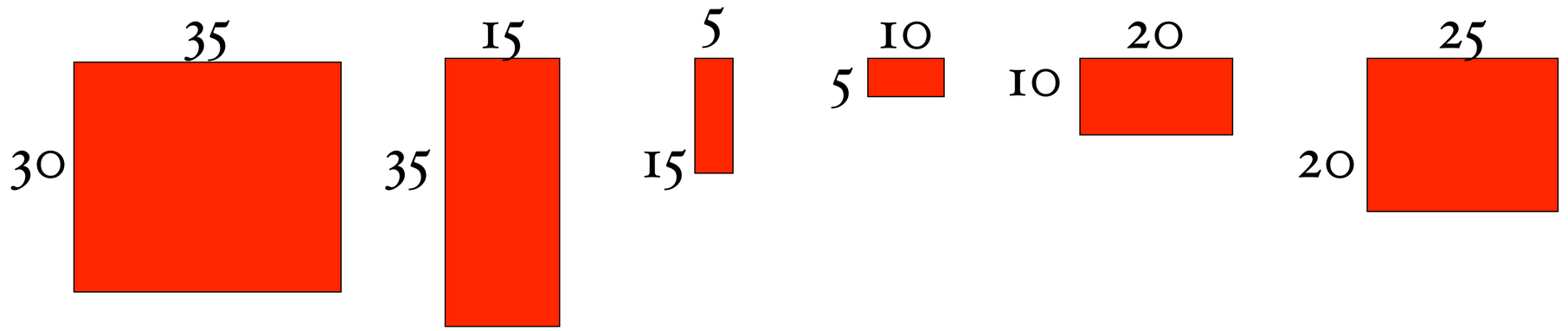
1

$$B(1,2) =$$



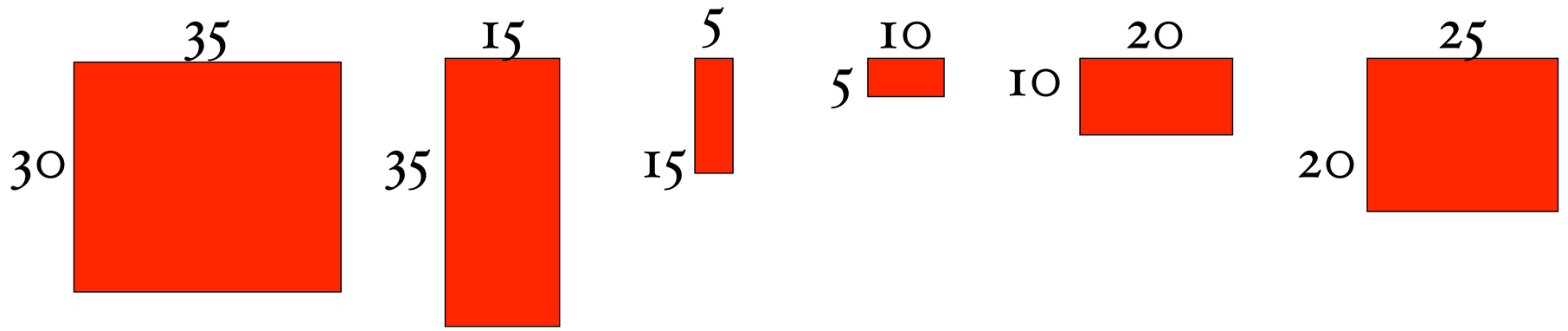
1

$$B(1,2) = \min\left\{B(1,1) + B(2,2) + r_1 c_1 c_2\right\}$$



1	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

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

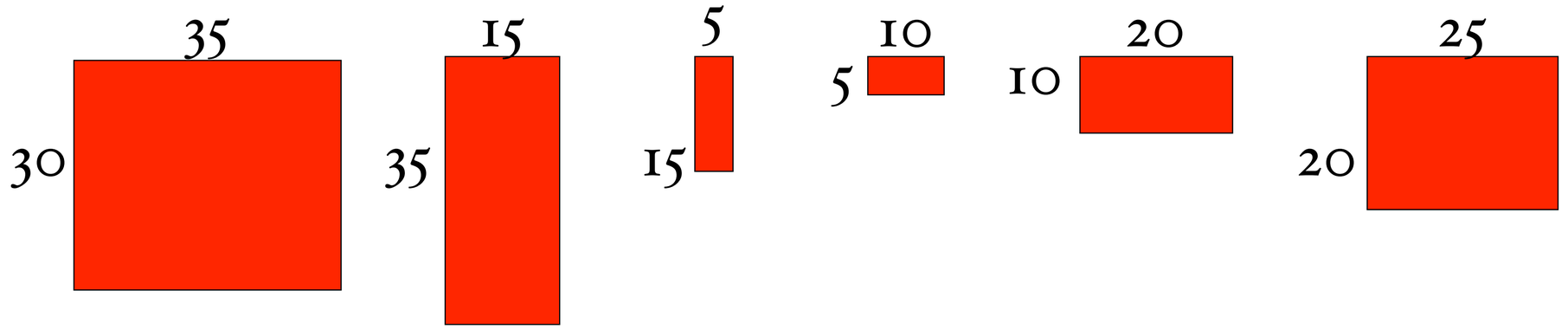


$$B(1,3) = \min \{$$

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

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

1 2 3 4 5 6

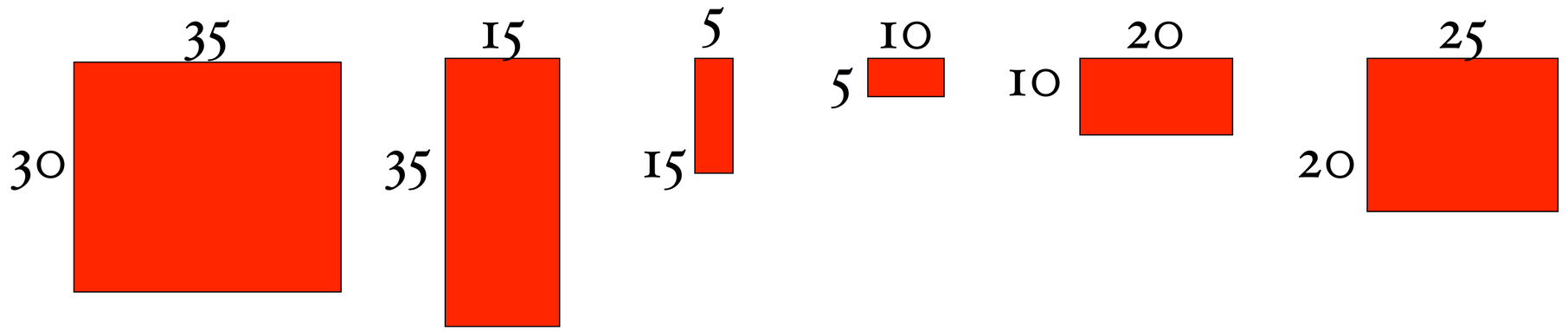


$$B(1,3) = \min \begin{cases} B(1,1) + B(2,3) + 30 \cdot 35 \cdot 5 & = 7875 \\ B(1,2) + B(3,3) + 30 \cdot 15 \cdot 5 & = 18000 \end{cases}$$

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

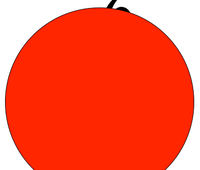
$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$

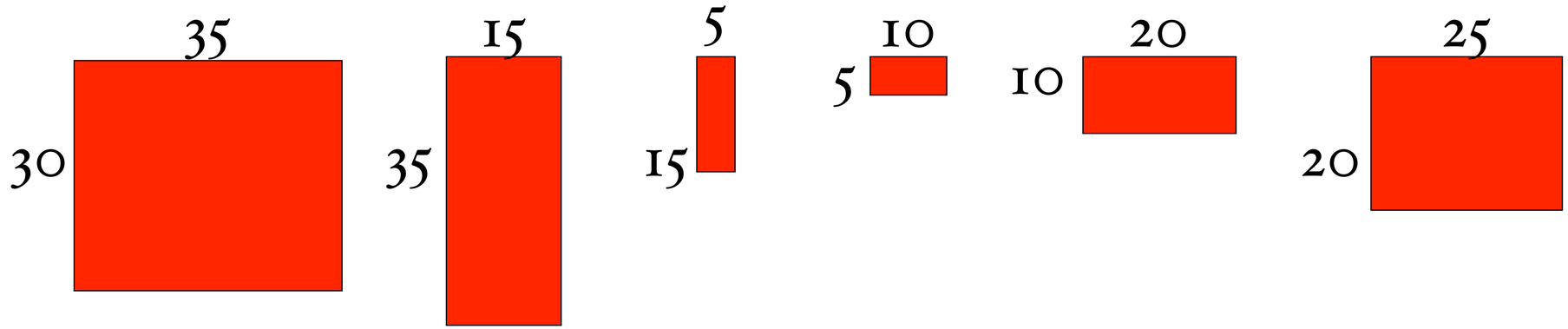
1 2 3 4 5 6



1	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

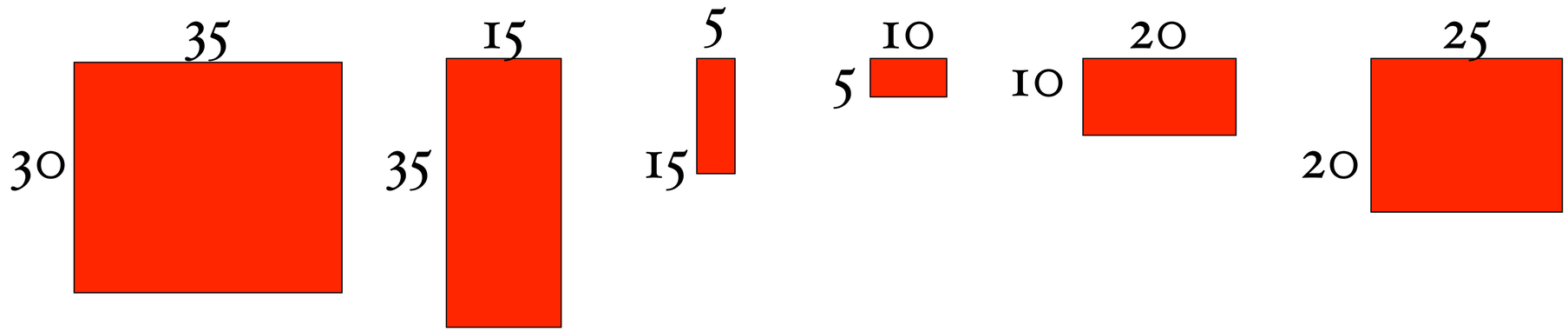
$$B(i,j) = \begin{cases} 0 \text{ if } i = j \\ \min_k \{B(i,k) + B(k+1,j) + r_i c_k c_j \end{cases}$$





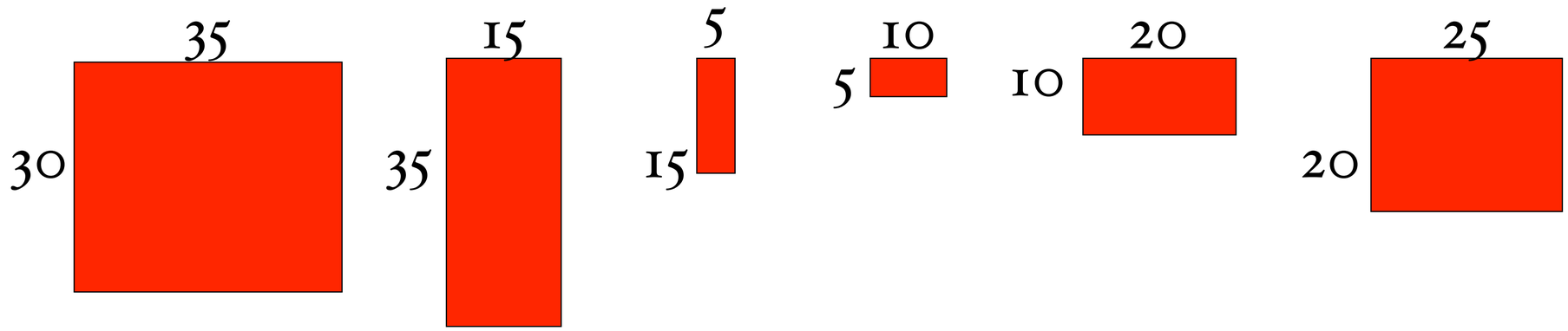
1 6

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



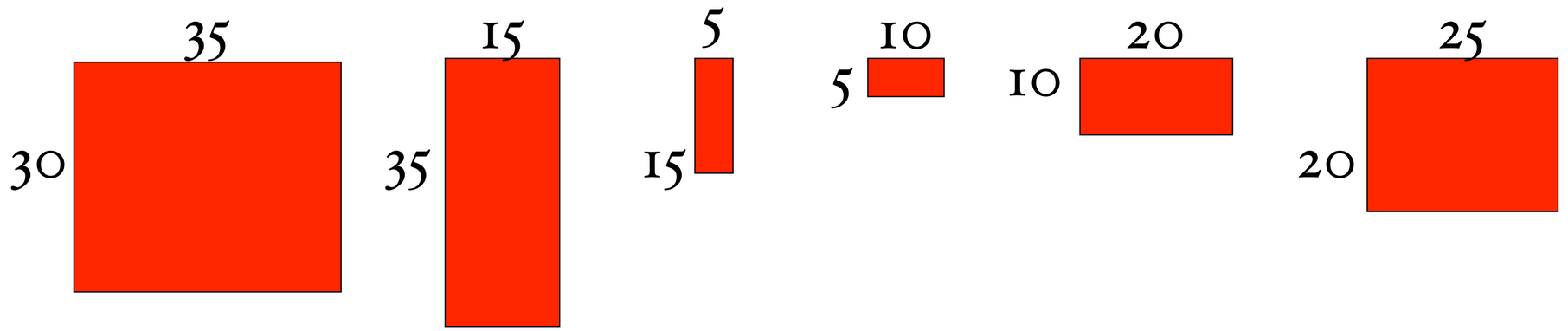
1 6

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

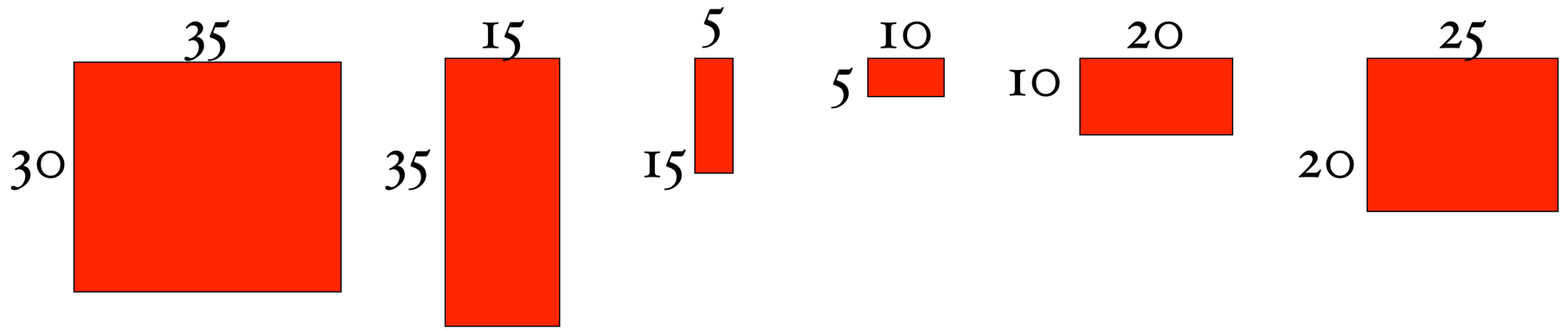


1 6

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



1	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



1	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

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