# Dynamic Programming part 10

Omkar Deshpande

#8 → Counting
#9 → Enumeration
#10 → Optimization

} problems involved DP on implicit/explicit trees

- Chaining Matrix Multiplication
- Burst Balloons

$M_1 \times M_2 \times \cdots M_n$

$M_1 \times M_2 \neq M_2 \times M_1$

Not commutative

$(M_1 \times M_2) \times M_3 = M_1 \times (M_2 \times M_3) = M_1 \times M_2 \times M_3$

Associative

$(n_1 \times n_2) \times n_3 = n_1 \times (n_2 \times n_3)$

$n_1 \times n_2 = n_2 \times n_1$

} in arithmetic

$M_1 \times M_2$

$a \times b$ $\quad$ $b \times c$

$b$

$a$ $\quad$ ith row $\quad$ $\times$ $b$ $\quad$ $c$ $\quad$ jth col

$M_1 \times M_2$ $\quad$ Time $= O(abc)$

$\langle a_1, a_2, \ldots a_k \rangle$ $\quad$ $=$ $\quad$ $i$ $\quad$ $j$

$\langle b_1, \ldots - b_k \rangle$

$= a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_k b_k = O(k)$ $\quad$ $a \times c$

$C_0 = 1$ $C_1 = 1$ $C_2 = 2$ $C_3 = 5$ $\frac{\binom{2n}{n}}{n+1} = C_n$

Find the most "efficient" parse tree.

## Problem Description

**Matrix Chain Multiplication**

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have:

(ABC)D = (AB)(CD) = A(BCD) = ....

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. For example, suppose A is a 10 × 30 matrix, B is a 30 × 5 matrix, and C is a 5 × 60 matrix. Then,

(AB)C = (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations

A(BC) = (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations.

Clearly, the first parenthesization requires less number of operations.

((AxB)xC) → 10x30x5 + 10x5x60 = 1500 + 3000 = 4500

(Ax(BxC)) → 30x5x60 + 10x30x60 = 9000 + 18000 = 27000 multiplications

Given an array *mtxSizes*[], which represents the chain of matrices such that the i-th matrix Ai is of dimension *mtxSizes*[i-1] x *mtxSizes*[i], we need to write a function that should return the minimum number of multiplications needed to multiply the chain. Length of the chain of matrices is *n* and thus the size of *mtxSizes* is (*n*+1).

**Example**

Input: [10, 30, 5, 60]

Output: 4500

**Notes**

Input Format: You will be given an integer array *mtxSizes*.

Output: Return an integer *minOps*, denoting the minimum number of operations needed.

Constraints:

- 3 <= len(*mtxSizes*) <= 100
- 0 <= *mtxSizes*[i] <= 100
- For any matrix, either both the dimensions will be zero, or both the dimensions will be non zero.

10, 30, 5, 60

$M_1 \otimes M_2 \otimes \cdots \otimes M_{n+1}$

↑ n multiplication operators

$M_1 \rightarrow C_0 = 1$

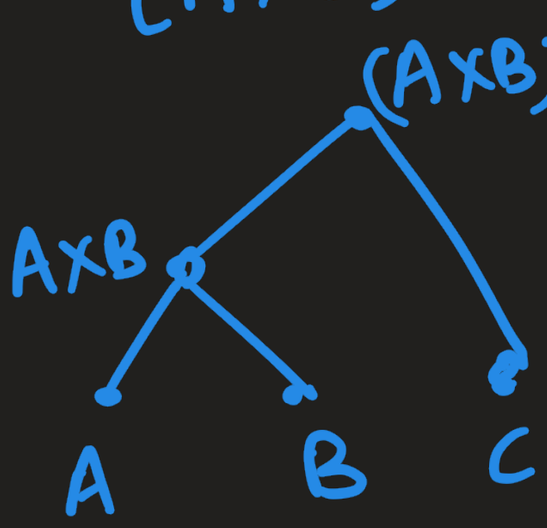
$M_1 \times M_2 \rightarrow C_1 = 1$

$M_1 \times M_2 \times M_3 \rightarrow C_2 = 2$

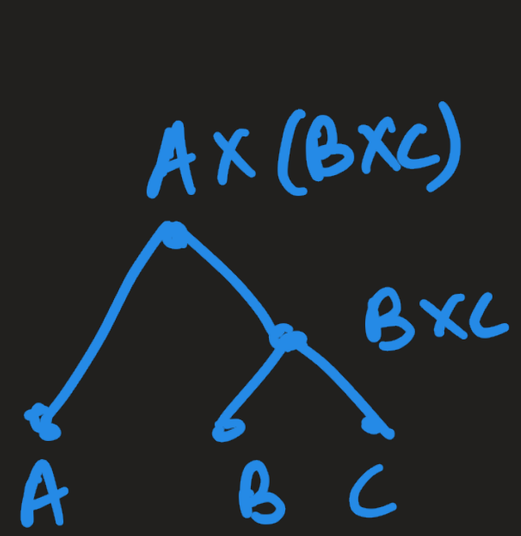
# ways to parenthesize the above $= C_n$ (nth Catalan number)

exponential in n.

$(A \times B) \times C$




$A \times (B \times C)$




2 Parse Trees

Associativity avoids ambiguity even with multiple parse trees.

① **Brute force:** Exhaustively enumerate & evaluate all parse trees (& their costs) ⟶ exponential time.
& pick the min cost parse tree

② **Greedy strategy:** Perform the cheapest multiplication first.

$A = 50 \times 20$

$B = 20 \times 1$

$C = 1 \times 10$

$D = 10 \times 100$

$A \times B \times C \times D$

$(B \times C) \rightarrow 20 \times 1 \times 10 = 200$

$20 \times 10$

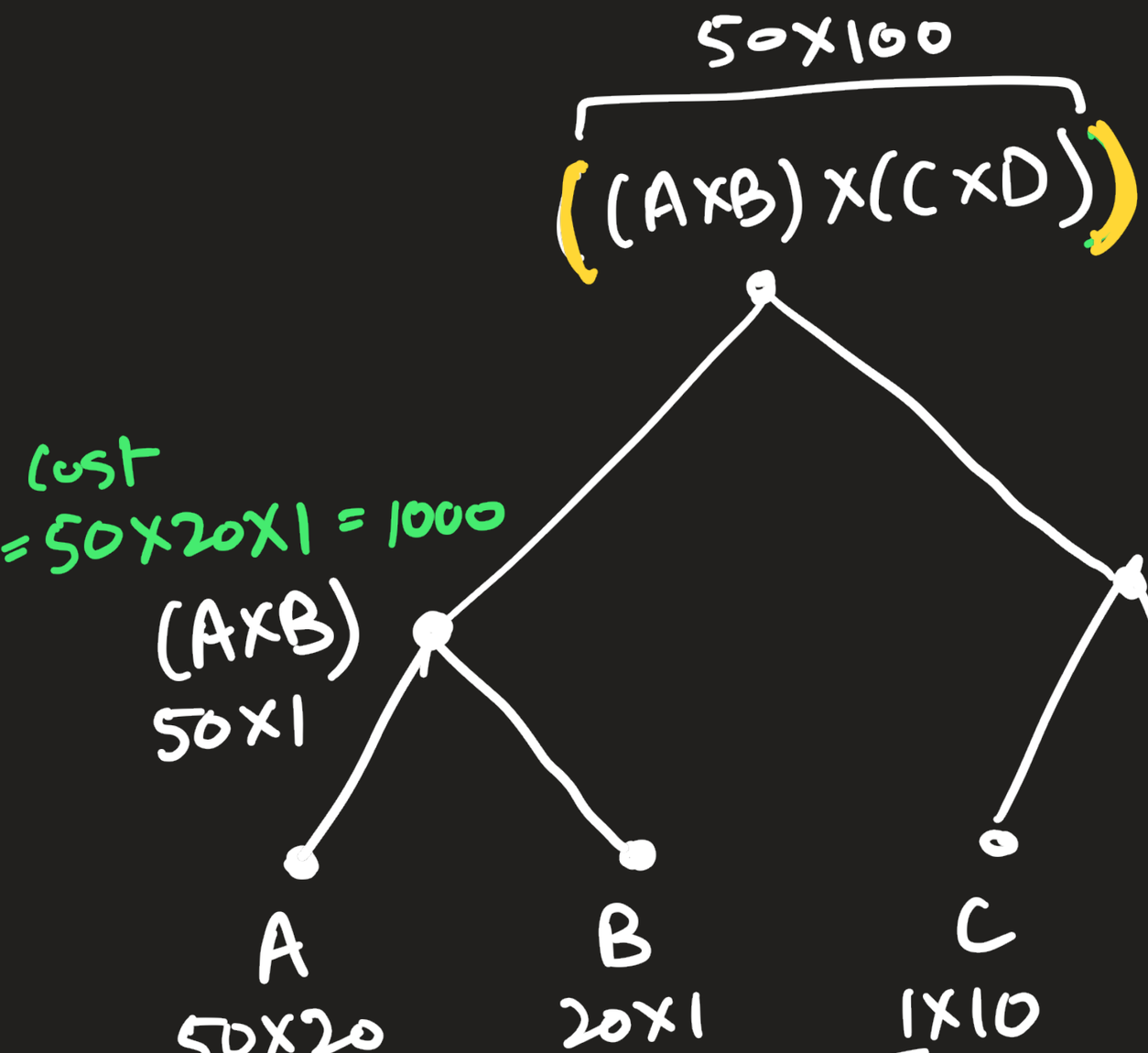
$(A \times (B \times C)) \rightarrow 50 \times 20 \times 10 = 10000$

$50 \times 10$

$(A \times (B \times C)) \times D \rightarrow 50 \times 10 \times 100 = 50000$

$50 \times 100$

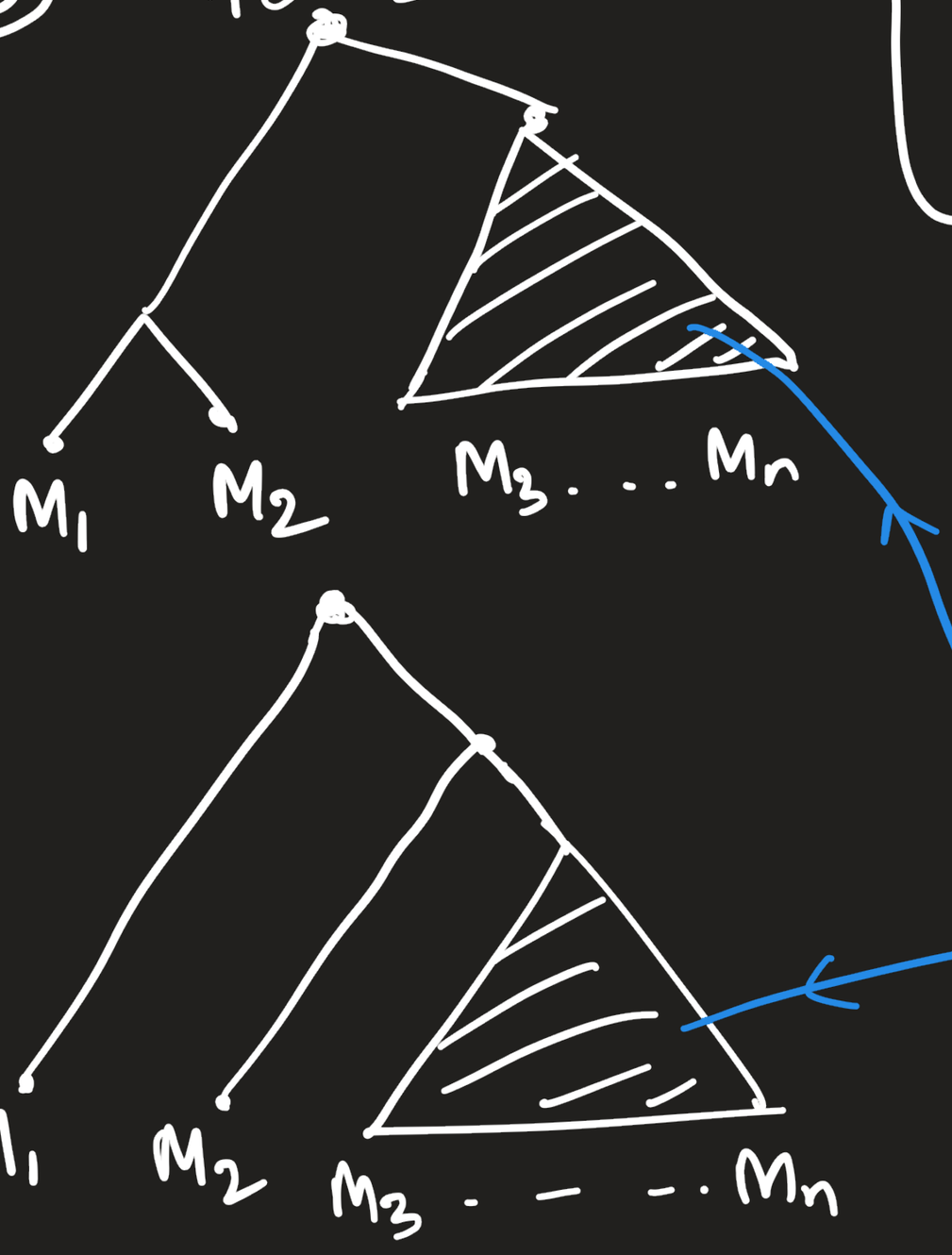
60,200 ops




Total cost = 5000 + 1000 + 1000 = 7000 operations

Greedy strategy does not work!

(3) $M_1 \otimes M_2 \otimes \dots M_n$




$(M_1 \times M_2) \times (M_3 \times \dots \times M_n)$

$M_1 \times (M_2 \times M_3 \times \dots \times M_n)$

$M_1 \times$ $M_2 \times (M_3 \times \dots M_n)$

Overlapping Subproblems

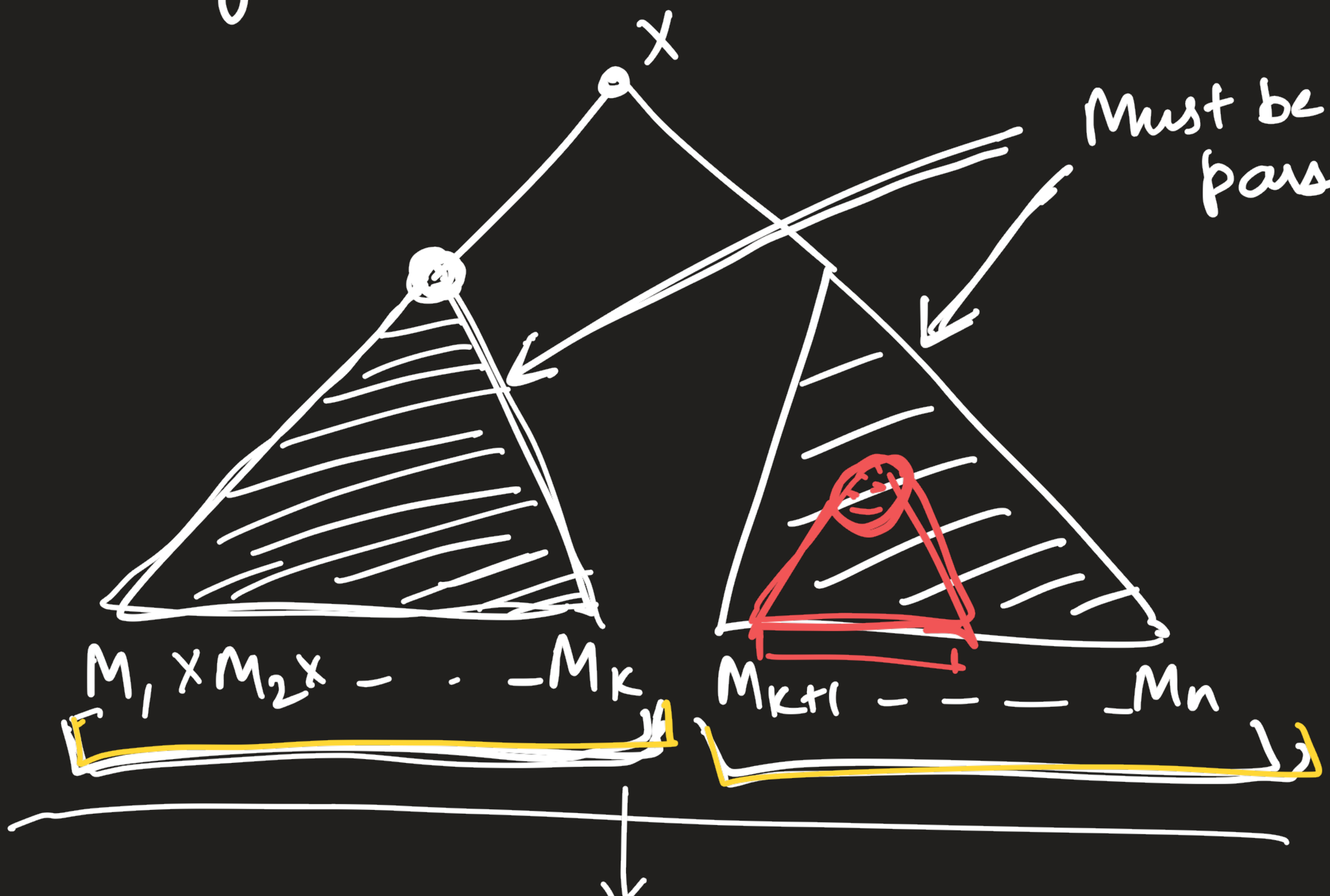
Subproblems are subtrees of the original problem

(NOT prefixes)

Permutation combination Seqalignment (2 strings)

Optimal substructure ? Yes

Imagine the optimal parse tree



Decrease-and-conquer: What is the last operation?

(Look at the rightmost parenthesis)

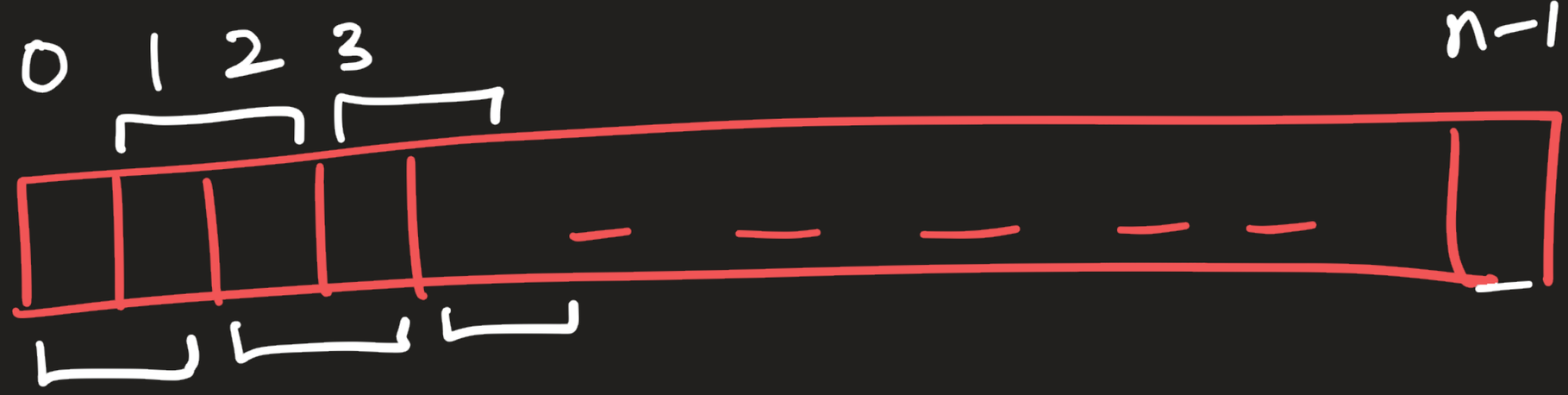
paired with the leftmost parenthesis.

$E \to$ matrix or $(E \times E)$

# choices = # matrices − 1


Cost 1 + Cost 2 + Cost 3

An arbitrary worker will get $M_i \times M_{i+1} \times \cdots M_j$ as the subproblem definition.



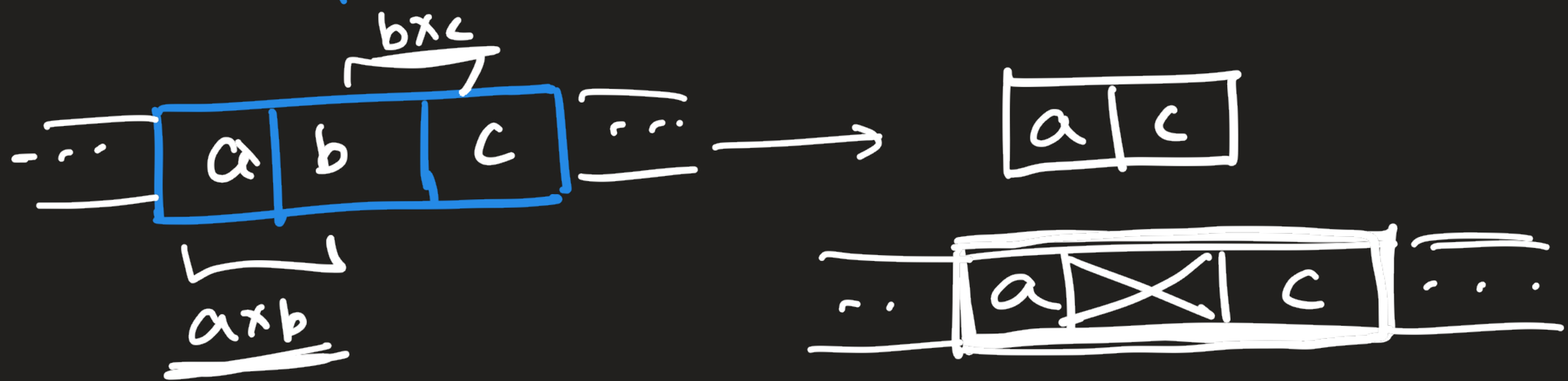

Array size → n numbers

n-1 matrices

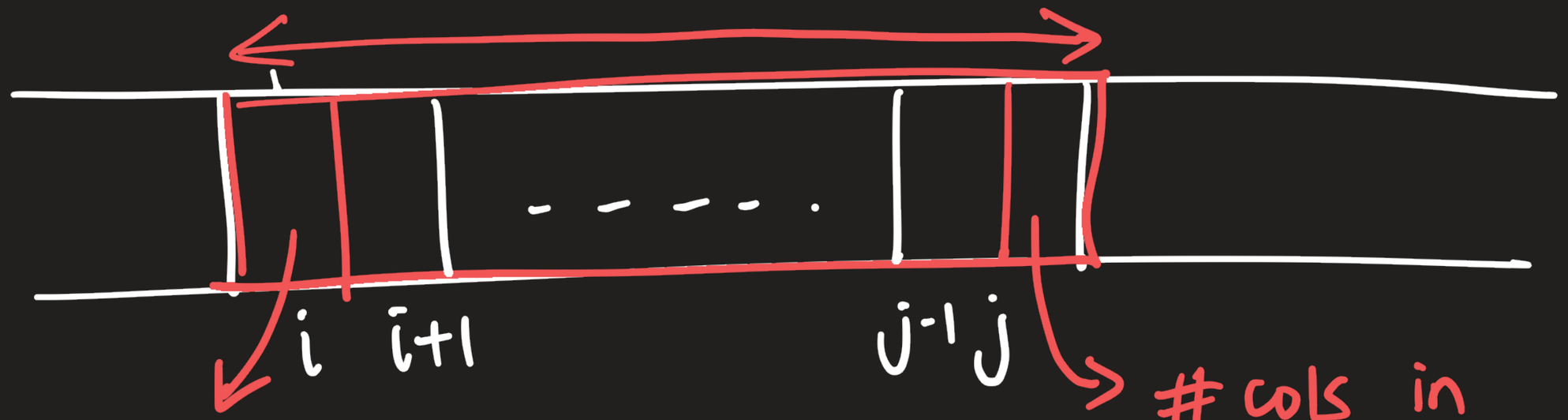
For there to be at least one (X) operation

$n-1 \geq 2$

$n \geq 3$

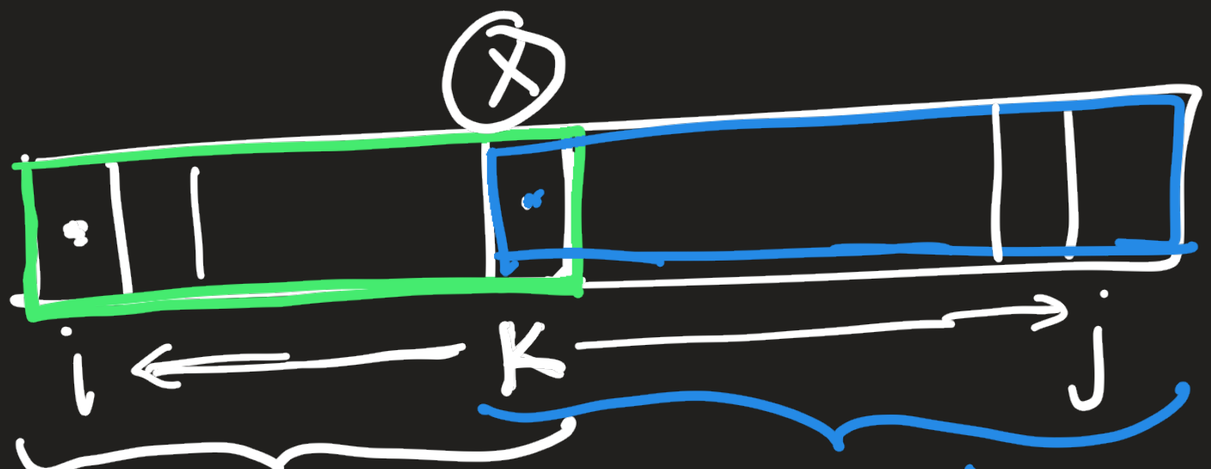


$M' \times M'' \times \cdots \times M'''$




What is the least cost way to multiply these matrices?

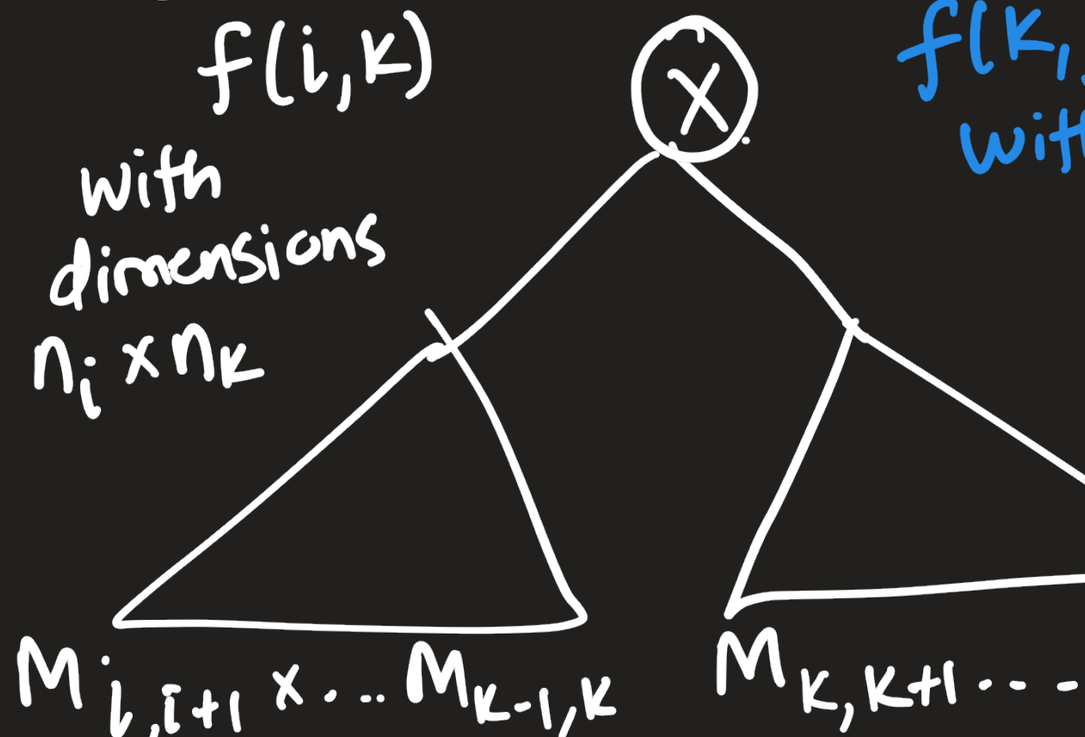
$f(i,j)$ = Min cost of multiplying $M_{i,i+1} \times M_{i+1,i+2} \times \cdots \times M_{j-1,j}$




Final ⊗ will have cost = $n_i \times n_K \times n_j$

With dimensions $n_i \times n_K$

$f(K,j)$ with dimensions $n_K \times n_j$

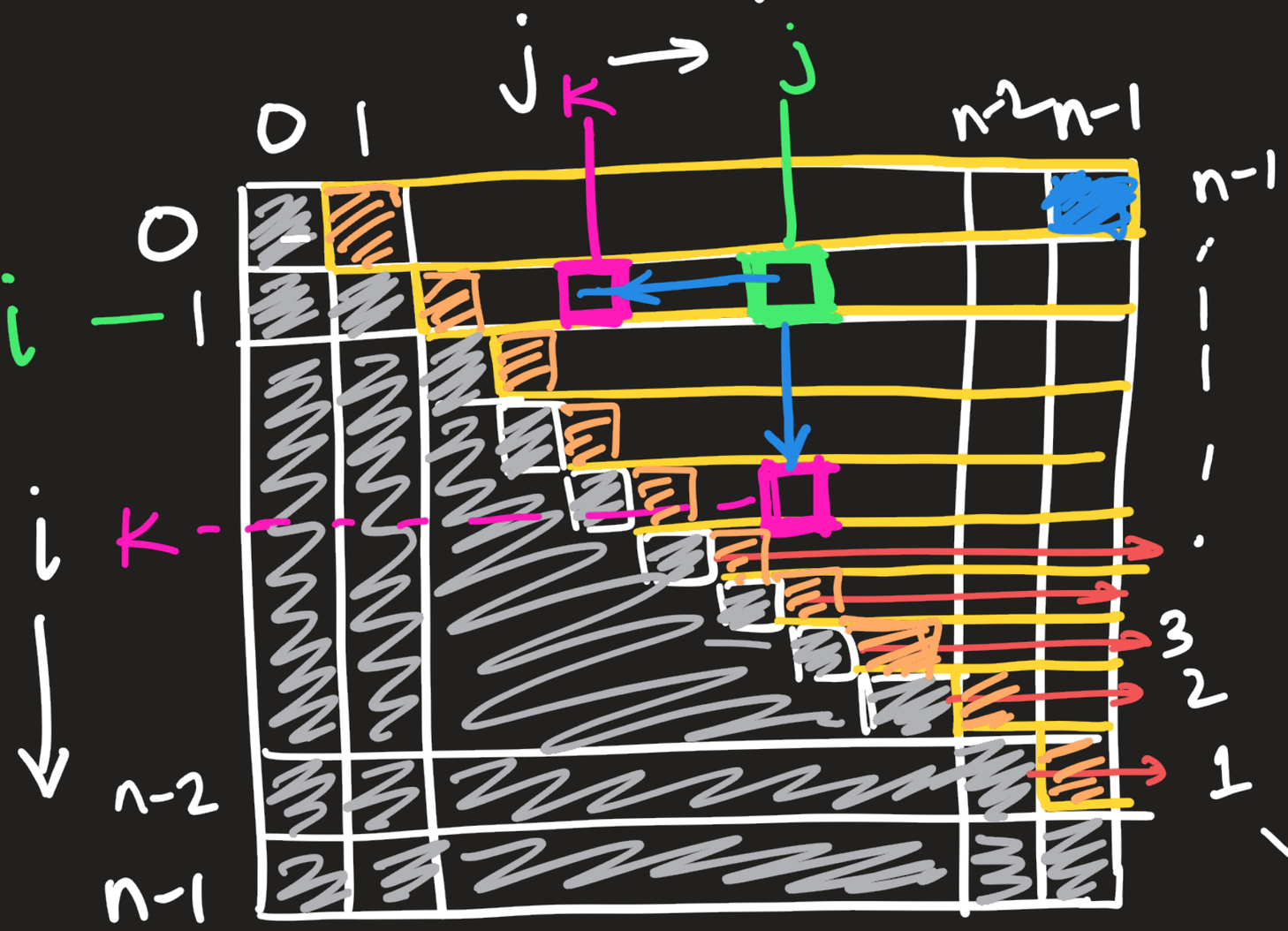


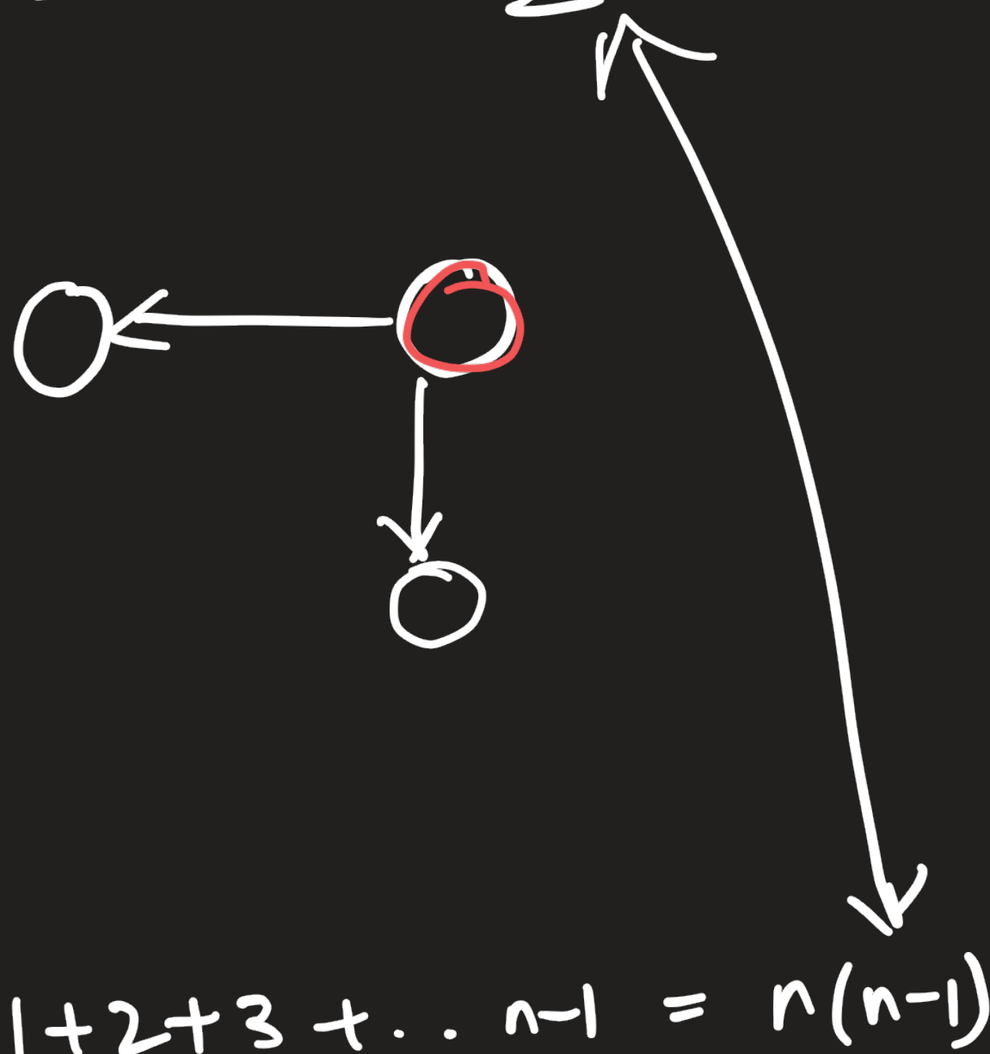

$$f(i,j) = \underset{K=i+1}{\overset{j-1}{MIN}} \left[ f(i,K) + f(K,j) + M[i] \times M[K] \times M[j] \right]$$

- $f(i,K)$: min cost for building left subtree
- $f(K,j)$: min cost for building right subtree
- $M[i] \times M[K] \times M[j]$: cost of final X operation

$\boxed{f(i,j)} \longleftarrow 0 \le i < j \le n-1$

# distinct subproblems $= {}^{n}C_2 = \frac{n(n-1)}{2}$



$1+2+3+\ldots n-1 = \frac{n(n-1)}{2}$

Base case : $f(i,i+1) = 0 \quad \forall \; i \in [0, n-2]$

table = a 2D table of size n x n
initialized to ∞ (n = length(mtxSizes))

∞ for minimization problem

# Base Cases
for row in 0 to n-2:
    table[row][row+1] = 0

# Recursive Case
for row in n-2 down to 0:
    for col in row+2 to n-1:
        // Calculate f(row, col) and store it in table[row][col]
        for k in row+1 to col-1:
            table[row][col] = min(table[row][col],
                (table[row][k] + table[k][col]
                + mtxSizes[row] * mtxSizes[k] *
                mtxSizes[col]))

max for maximization

O(n)

return table[0][n-1]

$$T(n) = \#cells \times O(n)$$
$$= O(n^2) \times O(n) = O(n^3)$$

$$\text{Aux Space} = O(n^2)$$

```python
def  minMultiplicationCost(mtxSizes):

    n = len(mtxSizes)
    table = [[float("inf")]*n for _ in range(n)]

    for i in range(n-1):
        table[i][i+1] = 0

    for i in range(n, -1, -1):
        for j in range(i+2, n):
            for k in range(i+1, j):
                table[i][j] = min(table[i][j], table[i][k] + table[k][j] + mtxSizes[i]*mtxSizes[k]*mtxSizes[j])

    return table[0][-1]
```

## 312. Burst Balloons

Hard 4191 122 Add to List Share

You are given `n` balloons, indexed from `0` to `n - 1`. Each balloon is painted with a number on it represented by an array `nums`. You are asked to burst all the balloons.

If you burst the `i^th` balloon, you will get `nums[i - 1] * nums[i] * nums[i + 1]` coins. If `i - 1` or `i + 1` goes out of bounds of the array, then treat it as if there is a balloon with a `1` painted on it.

Return *the maximum coins you can collect by bursting the balloons wisely*.

**Example 1:**

```
Input: nums = [3,1,5,8]
Output: 167
Explanation:
nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []
coins =  3*1*5    +   3*5*8   +  1*3*8  + 1*8*1 = 167
```

**Example 2:**

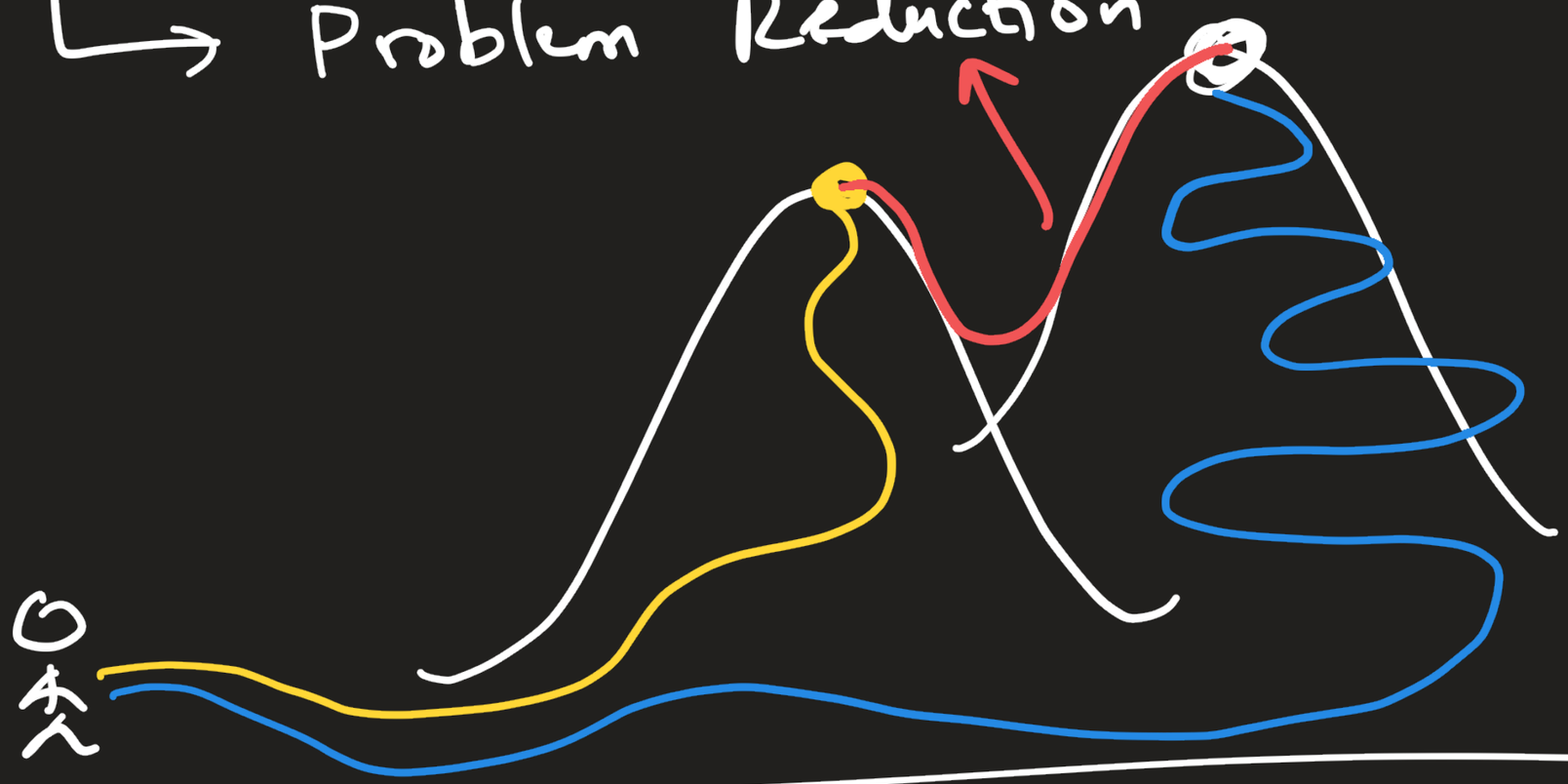
```
Input: nums = [1,5]
Output: 10
```

**Constraints:**

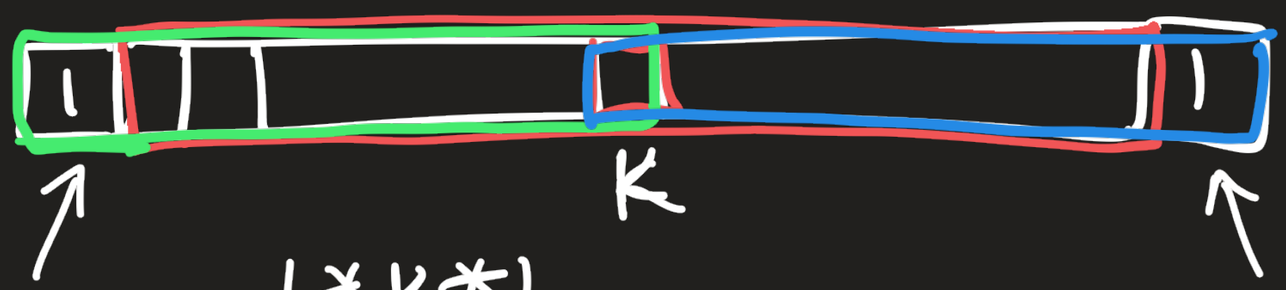
- `n == nums.length`
- `1 <= n <= 500`
- `0 <= nums[i] <= 100`

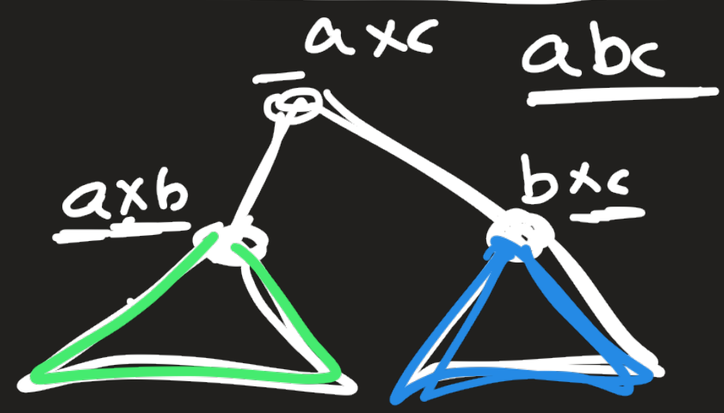
# Transform & Conquer

↳ Problem Reduction



Maybe bursting a balloon ≡ Multiplying two matrices ~~together.~~



newarray = [1] + nums + [1]

```python
    def maxCoins(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        newnums = [1] + nums + [1]

        n = len(newnums)
        table = [[0]*n for _ in range(n)]

        for i in range(n-1):
            table[i][i+1] = 0

        for i in range(n, -1, -1):
            for j in range(i+2, n):
                for k in range(i+1, j):
                    table[i][j] = max(table[i][j], table[i][k] + table[k][j] + newnums[i]*newnums[k]*newnums[j])

        return table[0][-1]

```