Dynamic Programming 4: Matrix Chain Product

- What's the goal of the matrix chain product algorithm?
  - We want to find the most efficient way to multiply a set of j matrices using the fewest scalar products.
  - Given an array of their sizes, we want to find the *order* to multiply them in such that the number of operations is minimized.
- What is the array D used for in MCP?
  - It's used to store the sizes of each matrix, such that each matrix M_j has dimensions $D[j-1]$ rows and $D[j]$ columns.
- Example

  M_1, M_2, M_3 where $D[0...3] = \{10, 20, 5, 30\}$. M_1 is a 10-by-20, M_2 is a 20-by-5, and M_3 is a 5-by-30.

  $(M_1 M_2) M_3$ takes $10 \times 20 \times 5 + 10 \times 5 \times 30 = 2500 \text{ scalar products}$

  $M_1(M_2 M_3)$ takes $20 \times 5 \times 30 + 10 \times 20 \times 30 = 9000$ scalar produts

  So as you can see, the order of multiplication matters!
- How do we simplify this problem with a recursive algorithm?
  - We find the best paranthesization for $(M_i ... M_k)(M_{k+1} ... M_j)$.
  - Basically we split the problem in half at a high level in the most optimal way, and do the same thing on each of the subproblems.
- What's the definition of the array we use for this problem?

  $\text{Cost}[i][j]$ = fewest scalar products to compute $M_i ... M_j$.

  $(M_i ... M_k)$ is $\text{Cost}[i][k]$

  $(M_{k+1} ... M_j)$ is $\text{Cost}[k+1][j]$.
- What's the recursive formula for this problem?

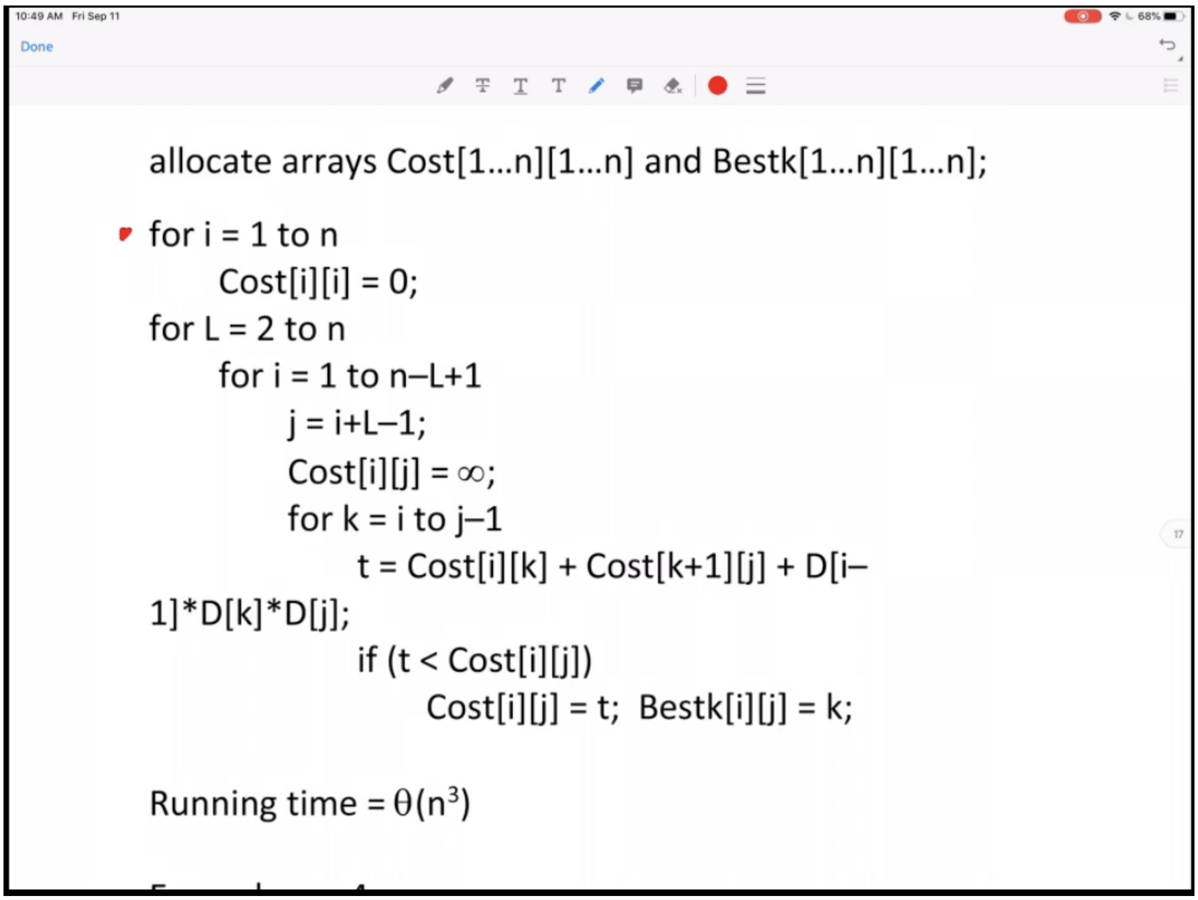
$\text{Cost}[i][j] = 0$ when $i = j$.

$\text{Cost}[i][j] = \min\{\text{Cost}[i][k] + \text{Cost}[k+1][j] + D[i-1] \times D[k] \times D[j] \mid i \leq k \leq j-1\}$ when $i < j$.

This part is over *all values of* k. It's every k value between i and j.

It's the sum of the two costs of the subproblems plus the current cost.

▼ What's the dynamic programming algorithm for this problem?



```
allocate arrays Cost[1…n][1…n] and Bestk[1…n][1…n];

for i = 1 to n
      Cost[i][i] = 0;
for L = 2 to n
      for i = 1 to n–L+1
            j = i+L–1;
            Cost[i][j] = ∞;
            for k = i to j–1
                  t = Cost[i][k] + Cost[k+1][j] + D[i–1]*D[k]*D[j];
                  if (t < Cost[i][j])
                        Cost[i][j] = t;  Bestk[i][j] = k;

Running time = θ(n³)
```

▼ What does the outer loop on the algorithm go from and to?

for L=2 to n

▼ What does the inner loop on the algorithm go from and to?

i=1 to n-L+1

▼ What does the innermost loop on the algorithm go from and to?

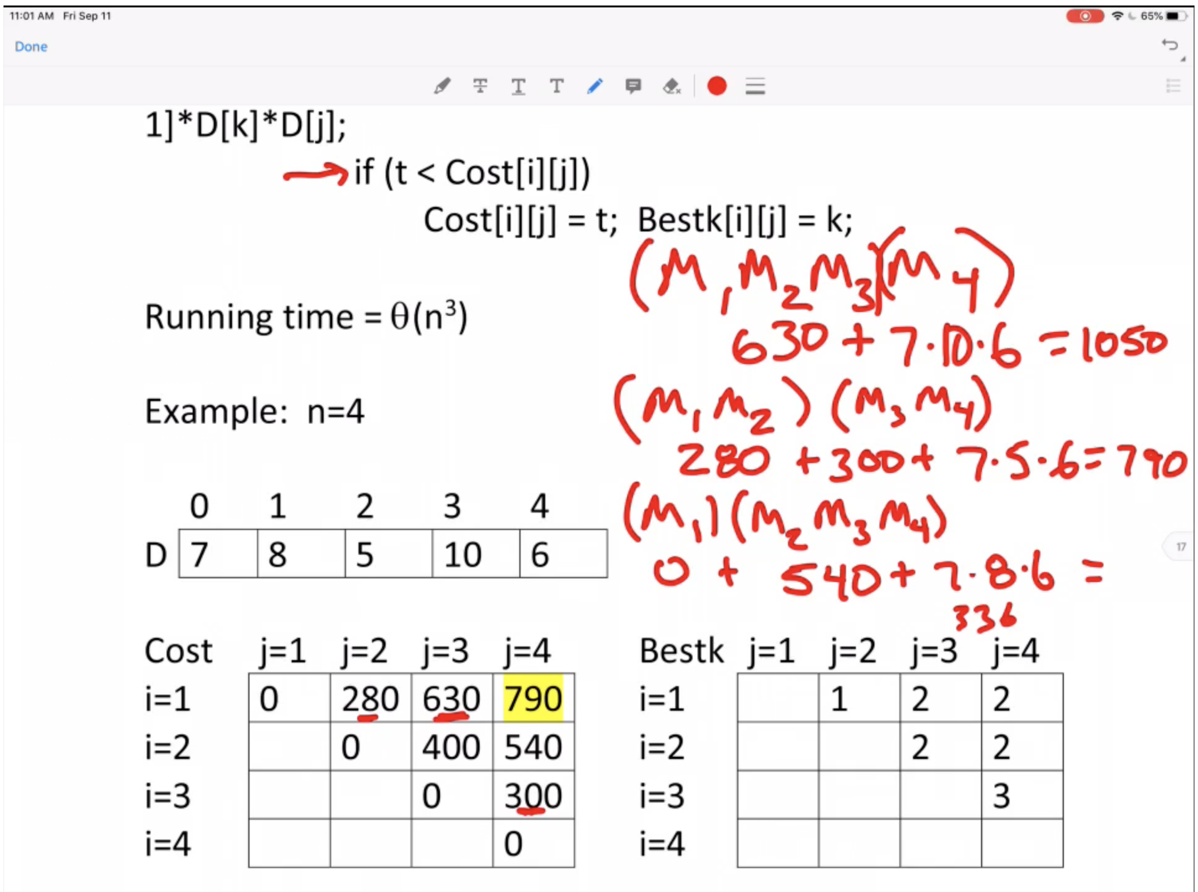
for k=i to j - 1

▼ What's the runtime for this dynamic programming algorithm?

$\theta(n^3)$

n here is the number of matrices we're actually multiplying out.

▼ Example runthrough



▼ How do we get the parenthization from the table?

See reference screenshot here.

As we create the table and add stuff in there, we go through and add the **best known k split for the subproblem we're on**. The final result here gives us the k split that gives us the cheapest result.

We store the best k whenever we set a min in a similar table, which lets us go back through and see where each split was.

k represents the best split!