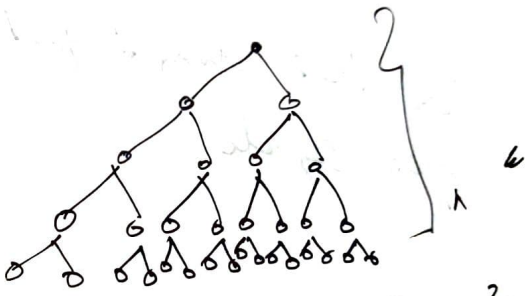
Algorithms:

Virgil M. A. N

26-Dec-2019

Our lectures this week are centered on the idea of problem solving models like maximum flow and shortest path, where a new problem can be formulated as an instance of one of those problems and then solved with a classic and efficient algorithm. To complete the course we describe the unsolved problem from theoretical computer science that is centered on the concept of algorithm efficiency and guides us in the search for efficient solutions to difficult problems

Reductions:



$2^2 \times 2^2$

1000 Gr 224 cards

99%. of working network CPUs

So%. have working CPUs $2^{10} \times 2^{10}$

$1000 \times \frac{99}{100}$ (10)

990 - tri 1000 500

10

1024
1024
4096
1008
0000
1024
1038176

$2^3 \times 2^2$

8×4

1038176 × 16

623056
038176

16615816

1) Longest path and longest cycle. Consider the following two problems

• Longest path: Given an undirected graph G and two distinct vertices s and t, find a simple path (no repeated vertices) between s and t with the most edges

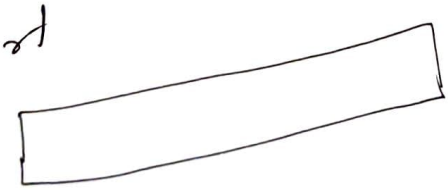
• Longest cycle: Given an undirected graph G, find a simple cycle (no repeated vertices or edges except the first and last vertex) with the most edges

Show that longest path linear time reduces to longest cycle

Find distinct elements in an array such that sum of them is zero.

$A = [i, j, k, l, m, n, o, p]$

MergeSort



Rod cutting problem:

i) Can cut up a rod of length n in 2^{n-1} different ways

ii) If we required the pieces to be cut in order of non decreasing size, there would be fewer ways to consider. for $n=4$, we could consider only 5 ways The number of ways is called the partition function. It is approximately equal to $e^{\pi\sqrt{2n/3}} / 4n\sqrt{3}$. This quantity is less than 2^{n-1} but still greater than any polynomial in n

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

$r_1 = 1$ from solution $1 = 1$ (no cuts)
$r_2 = 5$ from solution $2 = 2$ (no cuts)
$r_3 = 8$ from solution $3 = 3$ (no cuts)
$r_4 = 10$ from solution $4 = 2 + 2$
$r_5 = 13$ from solution $5 = 2 + 3$
$r_6 = 17$ from solution $6 = 6$ (no cuts)
$r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$
$r_8 = 22$ from solution $8 = 2 + 6$
$r_9 = 25$ from solution $9 = 3 + 6$
$r_{10} = 30$ from solution $10 = 10$ (no cuts)

Generally $r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$

$\simeq r_n = \max_{1 \le i \le n}(p_i + r_{n-1})$

Recursive top down approach

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

Error recursive call there is a "for" loop

Suppose $n = 6$

$i=1$: max(8, 1 + CUT-ROD(p, 5))
$i=2$ max(9, 5 + CUT-ROD(p, 4))
$i=3$ max(9, 8 + CUT-ROD(p, 3))
$i=4$ max(9, 9 + CUT-ROD(p, 2))
$i=5$ max(9, 10 + CUT-ROD(p, 1))
$i=6$ max(9, 17 + CUT-ROD(p, 0))

Thus print 17 ← Correct

CUT-ROD will double the run time even if we increase n by 1. Why is CUT-ROD so inefficient?

CUT-ROD calls itself recursively for every element in the for loop

for ( ) for ( )

for ( )

for (i=1 to n)

Recursion ( ) ——→ for ( ) for ( ) for ( )

in the Recursion

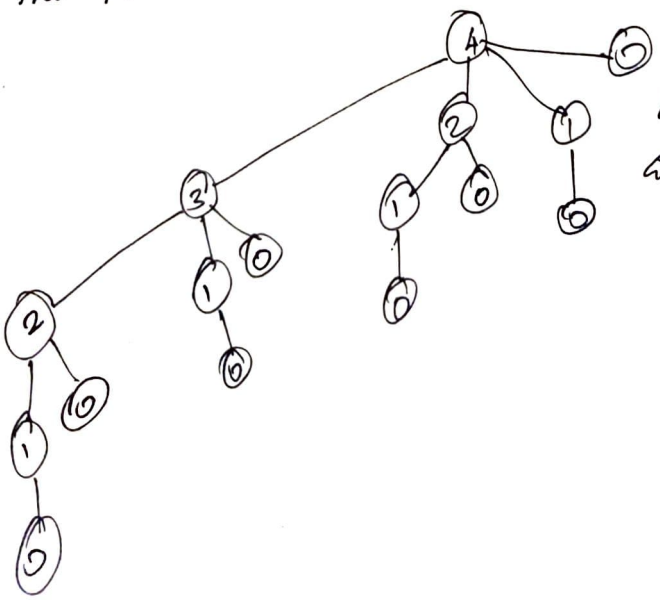
for

the

It explodes because every recursion will make a for loop

Running Time of CUT-ROD Analysis

T(n) denote the total number of calls made to CUT-ROD when called with its second parameter equal to n. This expression equals the number of nodes in a subtree whose root is labeled n in the recursion tree.



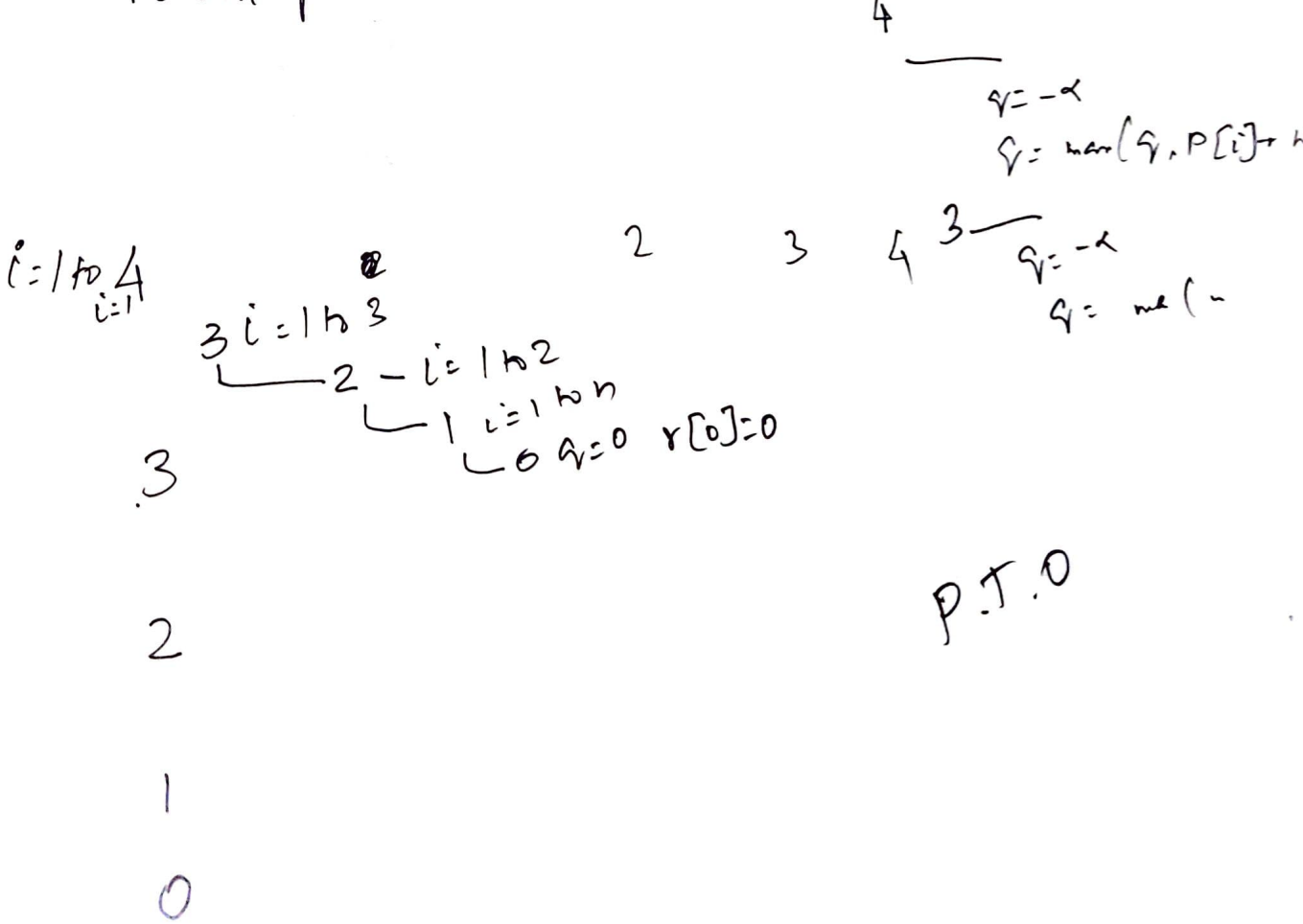
The recursion tree showing recursive calls resulting from a call CUT-ROD(p, n) for n=4

Top down approach

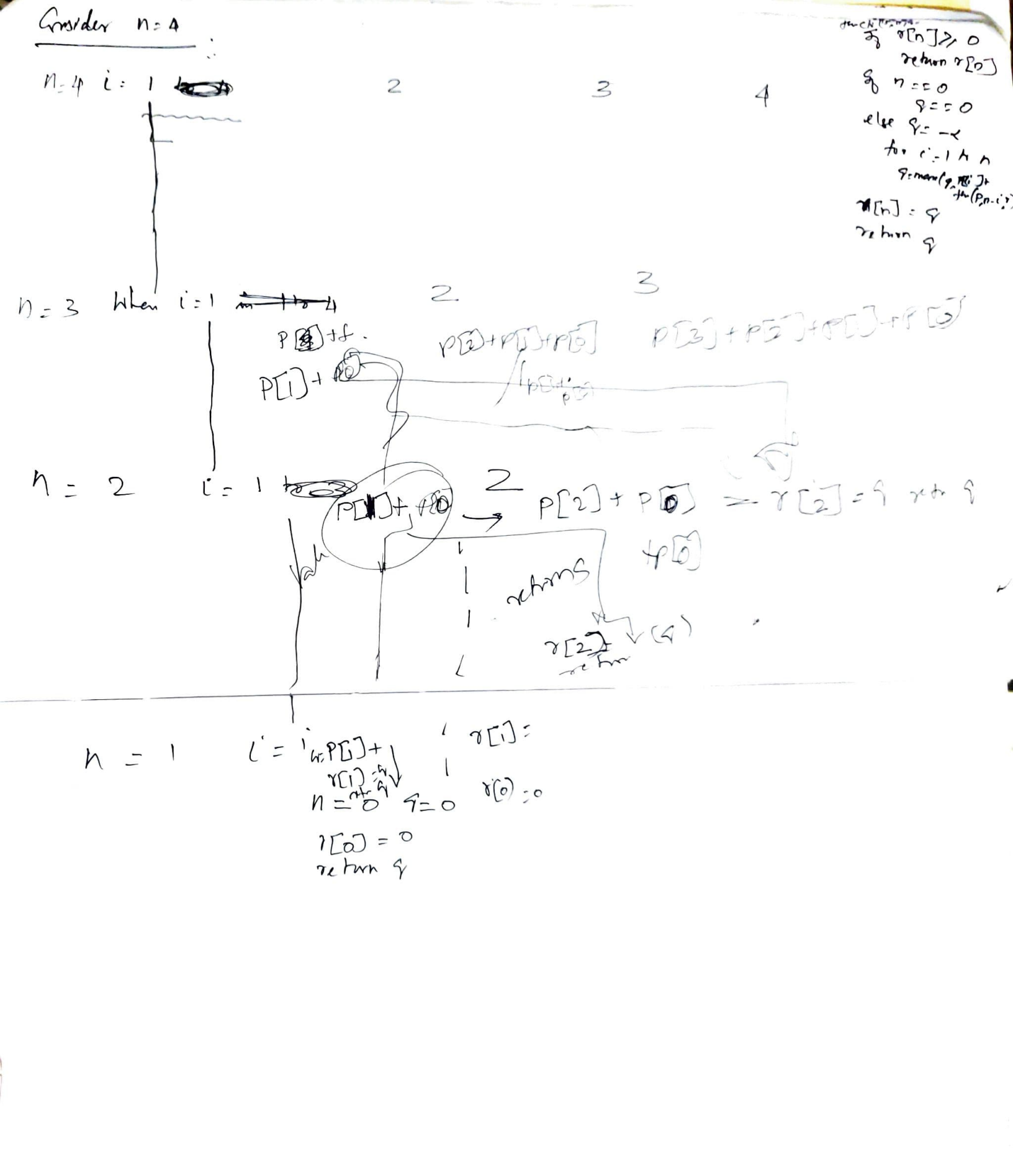
```
Memoized-Cut-Rod(p, n):
    let r be an array [0..n]
    for i = 1 to n
        r[n] = -∞
    Memoized-Cut-Rod-Aux(p, n, r)

Memoized-Cut-Rod-Aux(p, n, r):
    If r[n] ≥ 0
        return r[n]

    If n == 0
        q = 0
    else q = -∞
        for i = 1 to n
            q = max(q, p[i] + Memoized-Cut-Rod-Aux(p, n-i, r
    r[n] = q
    return q
```



Consider n = 4


n=4 i: 1 2 3 4
if r[n] >= 0
return r[n]
if n == 0
q = 0
else q = -∞
for i = 1 to n
q = max(q, p[i] + r(p, n-i))
r[n] = q
return q
n = 3 When i: 1
p[3] + r[0]
p[2] + p[1] + p[0]
p[3] + p[2] + p[1] + p[0]
p[1] +
n = 2 i: 1
p[1] + p[0]
2
p[2] + p[0] → r[2] = q return q
returns
p[0]
r[2] (q)
return
n = 1 i: 1 p[1] +
r[1] = q
n = 0 q = 0
r[0] = 0
r[0] = 0
return q
r[1] =

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

```
EXTENDED_BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be arrays
r[0] = 0
s[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        if q < p[i] + r[j - i]
            q = p[i] + r[j - i]
            s[j] = i
    r[j] = q
return r and s
```

Exercises : Pg : 364

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

$T(n) = 2^n$

The initial 1 is for the call at the root, and the term $T(j)$ counts the number of calls (including recursive calls) due to the call CUT-ROD$(p, n-i)$ where $j = n - i$

1 c. 3

5.4

Show that equation 15.4 follows from equation 15.3 and the initial condition $J(0) = 1$

ToDo: Complete the exercise

Dynamic programming

Matrix chain multiplication:

We are given a sequence $\langle A_1, A_2 \dots A_n \rangle$ of n matrices to be multiplied $A_1 A_2 \dots A_n$ — (2)

We can evaluate the expression using the standard algorithm for multiplying pairs of matrices as a subroutine once we have parenthesized it to resolve all ambiguities in how the matrices are multiplied together. Matrix multiplication is associative and so all parenthesizations yield the same product. A product of matrices is fully parenthesized if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses. For example if the chain of matrices is $\langle A_1, A_2, A_3, A_4 \rangle$, then we can fully parenthesize the product $A_1 A_2 A_3 A_4$ in five distinct ways:

$(A_1(A_2(A_3 A_4)))$.
$(A_1((A_2 A_3)A_4))$.
$((A_1 A_2)(A_3 A_4))$,
$(((A_1 A_2)A_3)A_4)$.

How we can parenthesize a chain of matrices can have a dramatic impact on the cost of evaluating the product. Consider first the cost of multiplying two matrices.

```
MATRIX-MULTIPLY(A, B)
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows x B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{p \times q} \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}_{q \times r} \qquad C = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$$

for i = 1 to 2 (rows)
  for j = 1 to 2 (cols)
    $c_{ij} = 0 \rightarrow c_{11}$
    for k = 1 to 2 (a_{ik} b_{kj})
      $c_{11} = c_{11} + a_{11} \cdot b_{11}$
      $0 + 1 \times 5$ k=1
      $c_{11} = 5$ k=2
      $c_{11} = c_{11} + a_{12} \cdot b_{21}$
      $= 5 + 14$

$c_{12} = 0$
$c_{12} = c_{12} + a_{11} \cdot b_{12}$

i = 1 2
j = 1 2 1 2
k 1 2 1 2 k 1 2 1 2

We can multiply two matrices A and B only if they are compatible

$A_{p \times q}$ $B_{q \times r}$

To illustrate the different costs incurred by different parenthesizations of a matrix product, Consider the problem of a chain $\langle A_1, A_2, A_3 \rangle$ of three matrices. Suppose that the dimensions of the matrices are 10×100, 100×5 and 5×50

If $((A_1 A_2) A_3) \Rightarrow 10 \times 100 \times 100 \times 5 \Rightarrow pqr \Rightarrow 10 * 100 * 5 = 5000$
$p \; q \quad q \; r$

$A_1 A_2 \Rightarrow$ 5000 scalar multiplications
$\therefore$ Compute a 10×5 matrix

$A_1 A_2 \Rightarrow 10 \times 100 \times 100 \times 5 = 10 \times 5$

$(A_1 A_2) A_3 \Rightarrow 10 \times 5 \times 5 \times 50 \Rightarrow 2500$ scalar multiplications
$p \; q \quad q \; r$

Total scalar multiplication $(A_1 A_2) A_3 = 7500$

So we instead multiply according to the parenthesization

B $(A_1 (A_2 A_3))$ ~~we perform $10 \cdot 5 \cdot 50 = 25000$~~

$A_2 A_3 \Rightarrow 100 \times 5 \times 5 \times 50 \Rightarrow pqr = 25000$
$p \; q \quad q \; r$

$= 100 \times 50$

$\downarrow$

$A_1 (A_2 A_3) \Rightarrow 10 \times 100 \times 100 \times 50 \Rightarrow pqr = 50000$
$p \; q \quad q \; r$

Total scalar multiplication is 75000.

Computing product by A is 10 times faster

We state the matrix-chain multiplication problem as follows, Given a chain $\langle A_1, A_2 \ldots A_n \rangle$ of n matrices where for $i = 1,2 \ldots n$ matrix A_i has dimension $P_{i-1} \times P_i$ fully parenthesize the product $A_1 A_2 \ldots A_n$ in a way that minimizes the number of multiplications.

Note that in the matrix-chain multiplication problem, we are not actually multiplying matrices.

Matrix Algo:

```
If A.cols != B.rows
    return "not compatible"
else
    for i = 0 to A.rows
        for j = 0 to B.cols
            C[i][j] = 0
            for k = 0 to A.cols
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
    return C
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad 2 \times 3 \quad P \times Q$$

$$B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad 3 \times 2 \quad Q \times R$$

$$C = \begin{bmatrix} 14 & 32 \\ 22 & 77 \end{bmatrix} \quad 2 \times 2$$

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$$

$$c_{00} = a_{00} \cdot b_{00} + a_{01} \cdot b_{10} + a_{02} \cdot b_{20}$$
$$c_{01} = a_{00} \cdot b_{01} + a_{01} \cdot b_{11} + a_{02} \cdot b_{21}$$
$$c_{10} = a_{10} \cdot b_{00} + a_{11} \cdot b_{10} + a_{12} \cdot b_{20}$$
$$c_{11} = a_{10} \cdot b_{01} + a_{11} \cdot b_{11} + a_{12} \cdot b_{21}$$

# of scalar multiplications: 12 $\Rightarrow$ $P=2$ $Q=3$ $R=2$

$2 \times 3 \times 2 = 12$

We state the matrix-chain multiplication problem as follows, given a chain $\langle A_1, A_2 \ldots A_n \rangle$ of n matrices where for $i = 1,2 \ldots n$ matrix A_i has dimension $P_{i-1} \times P_i$

$(A_1 \quad , \quad A_2 \quad , \quad A_3)$

$10 \times 100 \quad , \quad 100 \times 5 \quad , \quad 5 \times 50$

We state the matrix-chain multiplication problem as follows. Given a chain $\langle A_1, A_2 \ldots A_n \rangle$ where for $i = 1, 2 \ldots n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1, A_2 \ldots A_n$ in a way that minimizes the number of scalar multiplications.

$(A_1(A_2(A_3 A_4)))$

$(A_1((A_2 A_3) A_4))$

$((A_1 A_2)(A_3 A_4))$

$(((A_1 A_2) A_3) A_4)$

Denote the number of alternative parenthesizations of a sequence of n matrices by $P(n)$. When $n = 1$ we have just one matrix and therefore only one way to fully parenthesize the matrix.

When $n \geq 2$, a fully parenthesized matrix product is the product of two fully parenthesized matrix subproducts and the split between the two subproducts may occur between the kth and $(k+1)$st matrices for any $k = 1, 2 \ldots n-1$. Thus we obtain the recurrence.

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) P(n-k) & \text{if } n \geq 2 \end{cases}$$

$P(4) = P(1) P(3) \quad P(2) P(2)$

Dynamic programming:

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution
4. Construct an optimal solution from computed information

Step 1: The structure of an optimal parenthesization

For our first step in the dynamic programming paradigm we find the optimal substructure and then use it to construct an optimal solution for the problem from optimal solutions to subproblems

1) The optimal substructure is as follows. Suppose that to optimally parenthesize $A_i A_{i+1} \dots A_j$, we split the product between A_k & A_{k+1}

Then the way we parenthesize the prefix subchain $A_i A_{i+1} \dots A_k$ within the optimal parenthesization must be an optimal parenthesization of $A_i A_{i+1} \dots A_j$ must be an optimal parenthesization of $A_i A_{i+1} \dots A_k$

Step 2: Recursion We define the cost of an optimal solution recursively in terms of the optimal solutions to subproblems for matrix chain multiplication problem we pick as our subproblem of determining the minimum cost of parenthesizing $A_i A_{i+1} \dots A_j$ for $1 \le i \le j \le n$. Let $m[i,j]$ be the number of scalar multiplications needed to compute the matrix $A_{i..j}$ for the full problem.

$$m[i,j] = m[i,k] + m[k+1,j] + P_{i-1} P_k P_j$$

$A_i = P_{i-1} \times P_i$ $A_{i..k}$ $A_{k+1..j}$

$A_{ii} = P_i \times P_{i+1}$ $P_{i-1} \times P_k$ $P_k \times P_j$

$P_{i-1} \times P_k \times P_j$ the algo from 2-pages ago

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \} & \text{if } i < j \end{cases}$$

$(A_1 (A_2 (A_3 A_4)))$

$(A_1 (A_2 A_{34}))$

$(A_1 A_{234})$

A_{1234}

When multiplying two matrices

$p \times q \quad q \times r$

The running time is # of scalar multiplications

$A_3 A_4 \Rightarrow P_2 \times P_3 \; P_3 \times P_4$

$P_2 P_3 P_4$

$A_2 (A_3 A_4) \Rightarrow P_1 \times P_2 \; P_2 \times P_4$

$P_1 P_2 P_4$

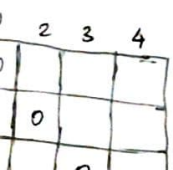
$A_1 (A_2 (A_3 A_4)) \Rightarrow P_1 \times P_4$

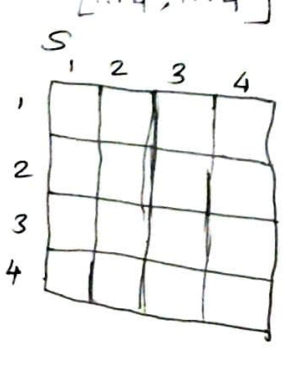
$P_0 \times P_1 \; P_1 \times P_4$

$P_0 P_1 P_4$

$\Rightarrow m[i,j] \Rightarrow m[1..n, 1..n]$

$n = 4 \Rightarrow m[1..4, 1..4]$





$A_{1..}$

MATRIX-CHAIN-ORDER(P)

p.length − 1

$m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables

for $i = 1$ to n

$m[i,i] = 0$

for $l = 2$ to n // l is the chain length

for $i = 1$ to $n - l + 1$

$j = i + l - 1$

$m[i,j] = \infty$

for $k = 1$ to $j - 1$

$q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$

if $q < m[i,j]$

$m[i,j] = q$

$s[i,j] = k$

$n = 3 \quad A_1 A_2 A_3 A_4$

$p_0 \; p_1 \; p_2 \; p_3 \; p_4$

Directly for $A_1 A_2 \Rightarrow p_0 p_1 \times p_1 p_2$

# of scalar multiplications $= p_0 p_1 p_2$

$m[1..n, 1..n] \quad s[1..n-1, 2..n]$

for $l = 2$ to 3

for $i = 1$ to $n - l + 1 \Rightarrow 3 - 2 + 1$

$= 1 \qquad 2$

$j = i + l - 1 \Rightarrow 2$

$i = 1 \quad j = 2$

$m[1,2] = \infty$

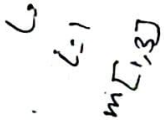
$i = 2 \quad 3$

$m[2,3]$

$i = 1 \quad j = 4$

$i + j - 1$

$l = 4$



$l = 2$ Assume $n = 8$

$i = 1$ to $n - l + 1 \quad j = i + l$

if $i = 1 \Rightarrow 1$ to 7

2 2 to 7

3 3 to 7

4 to 7

5 to 7

6 to 7

7 to 7

$i = 1$ to 7

$j = 2$

$m[1,2] = \infty$

for $k = 1$ to $j - 1$

$1 \quad 1$

$q = m[1,1] + m[2,2] + p_0 p_1 p_2$

if $q < m[i,j]$

$m[i,j] = q$

$s[i,j] = k$

$17 - 5 \qquad 16 - 5$

$1 + 15 - 5 \qquad 1 + 10$

$2 + 15 - 5$

$2 + 10 = 12$

$18 - 4$

$14 + 4 - 4 =$

$14 + 0 = 14$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \} \end{cases}$$

$(A_1 (A_2 (A_3 A_4)))$

$(A_1 (A_2 A_{34}))$

$(A_1 (A_{234}))$

A_{1234}

~~A_i~~

When multiplying two matrices

$p \times q \quad q \times r$

The running time is

# of scalar multiplication = pq

$A_3 A_4 \Rightarrow p_2 \times p_3 \quad p_3 \times p_4$

$p_2 p_3 p_4$

$A_2 (A_3 A_4) \rightarrow p_1 \times p_2 \quad p_2 \times p_4$

$p_1 p_2 p_4$

$A_1 (A_2 (A_3 A_4)) \rightarrow p_1 \times p_4$

$p_0 \times p_1 \quad p_1 \times p_4$

$p_0 p_1 p_4$

$m[i,j] \Rightarrow m[1..n, 1..n]$

$\Rightarrow m[1..4, 1..4]$

$n = 4$

m

	1	2	3	4
1	0			
2		0		
3			0	
4				0

s

	1	2	3	4
1				
2				
3				
4				

CHAIN-ORDER(P)

p.length − 1

m[1..n, 1..n] and s[1..n−1, 2..n] to new tables

i = 1 to n

m[i, i] = 0

L = 2 to n // L is the chain length

for i = 1 to n − L + 1

j = i + L − 1

m[i, j] = ∞

for k = 1 to j − 1

q = m[i, k] + m[k+1, j] + $P_{i-1} P_k P_j$

if q < m[i, j]

m[i, j] = q

s[i, j] = k

---

n = 3 $A_1 A_2 A_3 A_4$

$P_0 P_1 P_2 P_2 P_3 P_3 P_4$

Dimension

For eg $A_1 A_2 \Rightarrow P_0 P_1 \times P_1 P_2$

# of scalar multiplication $= P_0 P_1 P_2$

m: [1..n, 1..n] s: [1..n−1, 2..n]

for L = 2 to 3

for i = 1 to n − L + 1 → 3 − 2 + 1 = 2

i = 1 j = 1 + 2 − 1 ⇒ 2

i = 1 j = 2

m[1, 2] = ∞

i = 2 3

m[2, 3]

---

n = 4 L = 2

i = 1 to 4 − 2 + 1 = 3

L = 2 Assume n = 8

i = 1 to n − L + 1 j = i + L − 1

i = 1 ⇒ 1 to 7

2 2 to 7

3 3 to 4

4 to 7

5 to 7

6 to 7

7 to 7

i = 1 to 7

j = 2

m[1, 2] = ∞

for k = 1 to j − 1

1 to 1

q = m[1, 1] + m[2, 2] + $P_0 P_1 P_2$

if q < m[i, j]

m[i, j] = q

s[i, j] = k

17 − 5 16 − 5

2 + 15 − 5 1 + 15 − 5

2 + 15 = 12 1 + 10 =

18 − 4

14 + 4 − 4 =

14 + 0 = 14

$m[i,j] = \begin{cases} 0 & \text{if } i = j \end{cases}$

(A

MATRIX-CHAIN-ORDER(P)

n = P.length - 1

for l = 2 to n

for i = 1 to n-l+1;

j = i+l-1;

m[i,j] = ∞

for k = i to j-1

q = m[i,k] + m[k+1,j] + $p_{i-1} p_k p_j$

if q < m[i,j]:

m[i,j] = q

s[i,j] = k

$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

2×2 2×3

# of scalar operations

2×2×3 = 12

$A_1 A_2 A_3 A_4 A_5 A_6$

A_1A_2 A_2A_3 A_3A_4 A_4A_5 A_5A_6

$A_1A_2A_3$ ~~$A_2A_4A_5$~~ ~~$A_4A_5A_6$~~

$A_2A_3A_4$

$A_3A_4A_5$ $A_4A_5A_6$

$A_1A_2A_3A_4$ $A_2A_3A_4A_5$

$A_3A_4A_5A_6$

$A_1A_2A_3$ m[1,3] = m[1,1] + m[2,3] + $p_0 p_1 p_3$

$A_1A_2A_3$ m[1,2] [2,3]

if i = 1

$(A_1A_2)A_3A_4A_5A_6$

length = 2

A_1A_2

# of scalar multiplications $c_0 \times c_1 \times c_2$

$c_0 c_1 c_2$

m[1,2] = m[1,1] + m[2,2] + $c_0 \times c_1 \times c_2$

length = 3

$A_1A_2A_3$

i = 1 to n-l+1

n = 7

7-3+1

5

$A_1A_2A_3$

i = 1 i = 2

j = 2 j = 3

~~A_1A_2~~

$A_1A_2A_3A_4A_5A_6A_7A_8$

A_1A_2 A_2A_3 A_3A_4 A_4A_5 A_5A_6 A_6A_7 A_7A_8

1 2 3 4 5 6 7

~~$A_1A_2A_3$ $A_2A_3A_4$ $A_3A_4A_5$ $A_4A_5A_6$~~

$A_1A_2A_3$ $A_2A_3A_4$ $A_3A_4A_5$

$A_4A_5A_6$ $A_5A_6A_7$ $A_6A_7A_8$

$A_1A_2A_3A_4$ $A_2A_3A_4A_5$

$A_3A_4A_5A_6$ $A_4A_5A_6A_7$

$A_5A_6A_7A_8$

$A_1A_2A_3$ m[1,2] [3,3]

n = 7

n.length = 8

length of chain = 2

```
PRINT_OPTIMAL-PARENS(s,i,j)
    if i == j
        print "A_i"
    else
        print "("
        PRINT-OPTIMAL-PARENS(s, i, s[i,j])
        PRINT-OPTIMAL-PARENS(s, s[i,j]+1, j)
        print ")"
```

s[1,6]

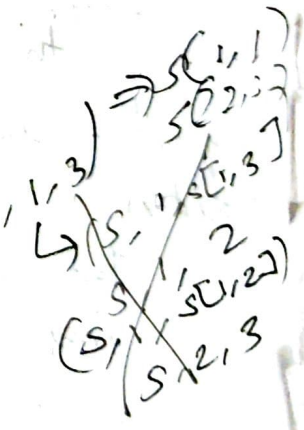
PRINT_OPTIMAL-PARENS(s, 1, 6)

(n=)

P_O_P(s, 1, s[1,6]) → (s, 1, 3)

s(1, 3)

P_O_P(s, 4, 6))



s, 1, 3

s, 1, 1

s, 2, 3 — s 2, 2

s 3, 3

s, 4, 6

Exercises:-

Matrix-Chain-Order find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is

for ⟨5, 10, 3, 12, 5, 50, 6⟩

Solve later

15.3 Elements of dynamic programming

$A_1 A_2 A_3 A_4$

$i=1 \quad j=4 \quad i<j \quad 4-1=3$

1) $(A_1 (A_2 A_3)) A_4$

2) $((A_1 A_2) A_3) A_4$ 3 choices

3) $(A_1 A_2)(A_3 A_4)$

Eg

~~Running a dynamic programming~~

Typically, the running time of a dynamic programming algorithm depends on the product of two factors

the number of subproblems overall

how many choices we look at for each subproblem

Dynamic programming often uses optimal substructure in a bottom up fashion

$A_1 A_2 \ldots$

$\geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1)$ for $n > 1$

≥ 1

Noting that for $i = 1, 2, \ldots n-1$ each term $T(i)$ appears
as $T(k)$ and once as $T(n-k)$ and collecting the
1 is in the summation together with the one
front, we can rewrite the recurrence as

$\geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1)$ for $n > 1$

$n = 5$

$T(1) + T(4) + 1$
$T(2) + T(3) + 1$
$T(3) + T(2) + 1$
$T(4) + T(1) + 1$

$2 \left(\sum_{i=1}^{n-1} T(i)\right) + 4$

$T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n$

shall prove that $T(i) = \Omega(2^n)$ using the substitution
nd. Specifically we shall show that $T(n) \geq 2^{n-1}$
all $n \geq 1$. The basis is easy, since $T(1) \geq 1 = 2^0$
ductively for $n \geq 2$

intain

$= 2^0$

$n-1$
$\sum T(i) + n$
$i=1$

$$T(n) \geq 2 \sum_{i=1}^{n-1} 2^{i-1} + n$$

$$= 2\left(2^{n-1-1}\right) + n$$

$$= 2\left(2^{n-2}\right) + n$$

$$= 2\left(2^{n} - 1\right) + n$$

$$=$$

$$T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n$$

$$T(n) \geq 2 \sum_{i=1}^{n-1} 2^{i-1} + n \qquad \text{if } n = 4,\ n-1 = 3$$

$$2\left(2^{0} + n + 2^{1} + n + 2^{2} + n + 2^{3} + n\right)$$

$$2\left(1 + n + 2 + n + 2^{2} + n + 2^{3} + n\right)$$

$$2\left(4n + 1 + 2 + 4 + 8\right)$$

$$\Rightarrow 2\left(4n + 2^{0} + 2^{1} + 2^{2} + 2^{3}\right)$$

$$= 2 \sum_{i=0}^{n-2} 2^{i} + n \qquad \text{if } n = 4$$

$$2\left(2^{0} + n + 2^{1} + n + 2^{2} + n\right)$$

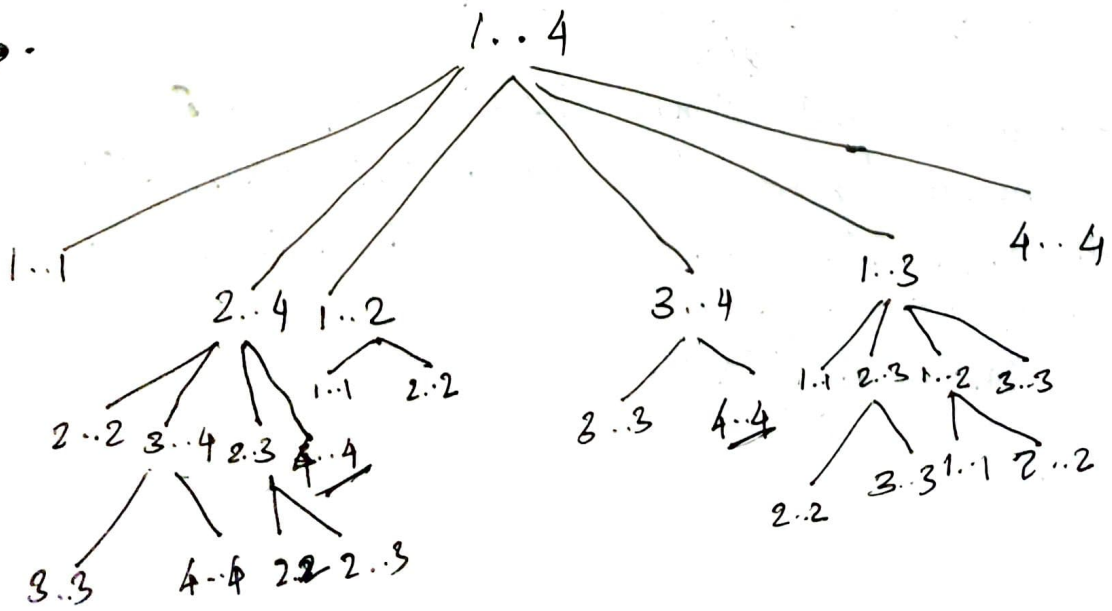
$$2\left(3n + 2^{0} + 2^{1} + 2^{2}\right)$$

We shall prove that $T(n) = \Omega(2^n)$ using the substitution method. Specifically we shall show that $T(n) \geq 2^{n-1}$ for all $n \geq 1$. The basis is easy, since $T(1) \geq 1 = 2^0$. Inductively for $n \geq 2$ we have

$$T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n$$

```
RECURSIVE-MATRIX-CHAIN(P, i, j)
1   if i == j
2       return 0
3   m[i,j] = ∞
4   for k = i to j-1
5       q = RECURSIVE-MATRIX-CHAIN(P, i, k)
            + RECURSIVE-MATRIX-CHAIN(P, k+1, j)
            + P_{i-1} P_k P_j
6       if q < m[i,j]
7           m[i,j] = q
8   return m[i,j]
```

b.



We can show that the time to compute $m[i,n]$ by this recursive procedure is at least exponential in n. Let $T(n)$ denote the time taken by RECURSIVE-MATRIX-CHAIN to compute an optimal parenthesization of n matrices.

Because the execution of lines 1-2 and of lines 6-7 each take at least unit time, as does the multiplication in line 5, inspection of the procedure yields recurrence.

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \text{ for } n > 1$$

Noting for that $i = 1, 2 \ldots n-1$, each term $T(i)$ appears once as $T(k)$ and once as $T(n-k)$, and collecting the $n-1$ 1s in the summation together with the 1 out front, we can rewrite the recurrence as

$$T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n$$

We shall prove that $T(n) = \Omega(2^n)$ using the substitution method. Specifically we shall show that $T(n) \geq 2^{n-1}$ for all $n \geq 1$. The basis is easy, since $T(1) \geq 1 = 2^0$. Inductively, for $n \geq 2$ we have

$$T(n) \geq 2 \sum_{i=1}^{n-1} 2^{i-1} + n$$