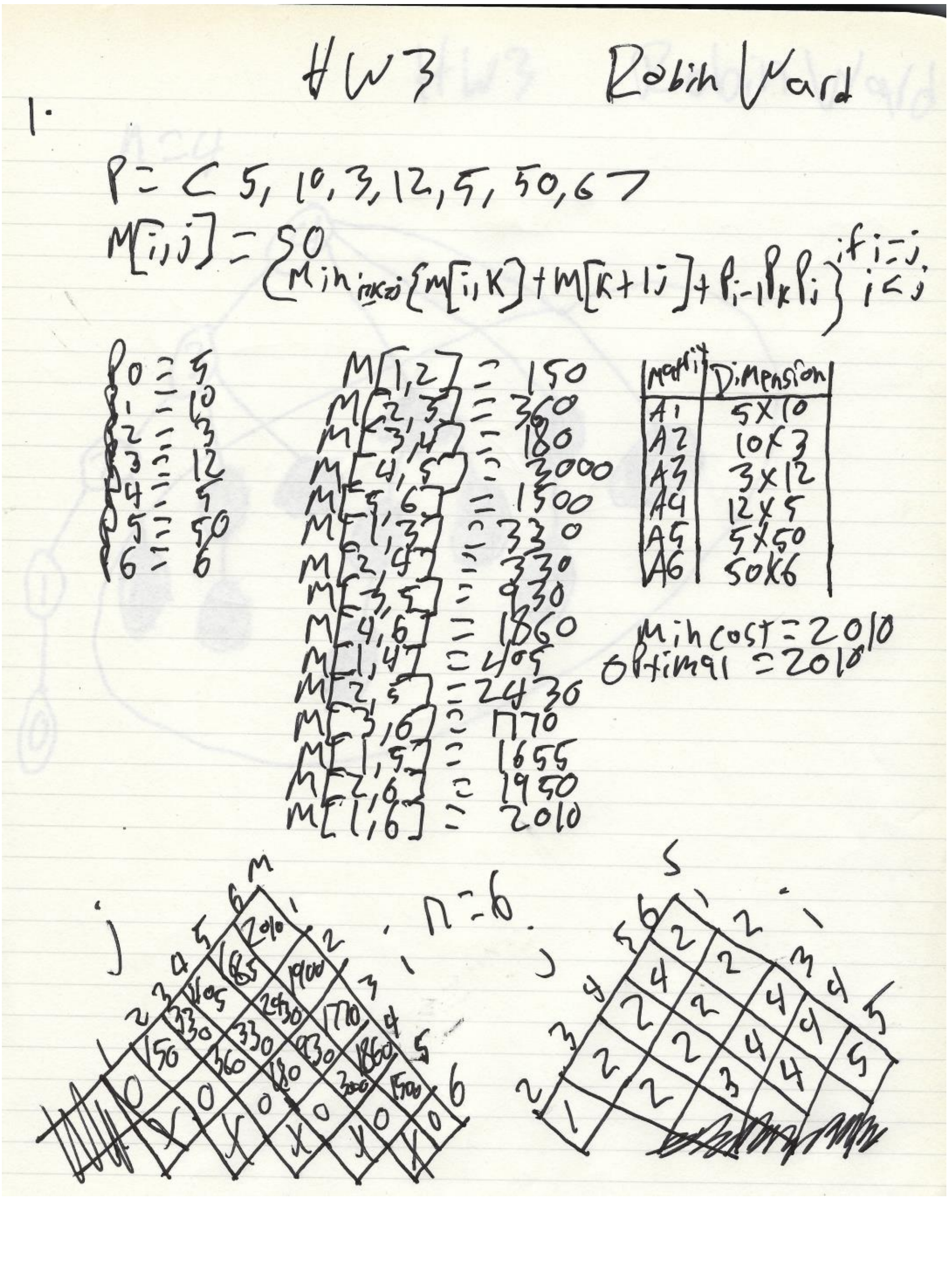
1. Problem 15.2-1 in the text. Show the s and m matrices (like Figure 15.5) and then provide the optimal parenthesization.



HW3 Robin Ward

1.

$P = \langle 5, 10, 3, 12, 5, 50, 6 \rangle$

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

$P_0 = 5$
$P_1 = 10$
$P_2 = 3$
$P_3 = 12$
$P_4 = 5$
$P_5 = 50$
$P_6 = 6$

$M[1,2] = 150$
$M[2,3] = 360$
$M[3,4] = 180$
$M[4,5] = 3000$
$M[5,6] = 1500$
$M[1,3] = 330$
$M[2,4] = 330$
$M[3,5] = 930$
$M[4,6] = 1860$
$M[1,4] = 405$
$M[2,5] = 2430$
$M[3,6] = 1770$
$M[1,5] = 1655$
$M[2,6] = 1950$
$M[1,6] = 2010$

Matrix	Dimension
A1	5x10
A2	10x3
A3	3x12
A4	12x5
A5	5x50
A6	50x6

Min cost = 2010
Optimal = 2010

m

j \ i	1	2	3	4	5	6
6	2010	1900	1770	1860	1500	0
5	1685	2430	930	3000	0	
4	405	330	180	0		
3	330	360	0			
2	150	0				
1	0					

s

j \ i	1	2	3	4	5
6	2	2	4	4	5
5	4	2	4	4	
4	2	2	3		
3	2	2			
2	1				

2. Convert the recursive characterization of equations (16.2) in text into a recursive algorithm and provide the algorithm below.

```
function computeC(i,j):
max = 0
if($S_{i,j}$ = 0):
            return 0
foreach $a_k$ in $S_{i,j}$ do:
            if max < computeC(I,k) + compute(k,j) +1:
                        max = computeC(I,k) + computeC(k,j) + 1
return max
```

3. Write a memorized recursive algorithm RECURSIVE-MEMOIZED-LCS-LENGTH(X,Y) to compute the length of the LCS of X and Y based on equations (15.9), p. 393. You can do this later after 04/09's class.

```
RECURSIVE-MEMOIZED-LCS-LENGTH(X, Y, i, j)
i = len.X
j = len.Y
c = new array table
if c[i,j] > -1
            return c[i,j]
if c[i,j] is null
            return c[i,j]
if i = 0 or j = 0
return c[i, j] = 0
if x[i - 1] = y[j - 1]
return c[X, Y, i - 1, j – 1] + 1
return c[max(X, Y, i - 1, j)], c[X, Y, i, j - 1)]
```

Please read 7270-09-DP Part I.pdf, pp 29 for the following two questions, which is also placed here:

- Another way of characterizing the structure of the optimal solution to this problem recursively is to say: we can first cut the rod into two pieces of length i and n-i inches each, and then recursively calculate the optimal cuts for each of those pieces; if we do that for each possible value of i from 1 to n-1 and calculate the total revenue r_i+r_{n-i} for each of those possible cuts, and then take the maximum of those and p_n (the revenue if the rod is sold without cutting), that will give us the optimal revenue r_n.
- This is formulated as equation 15.1 (p.362). Understand this equation.

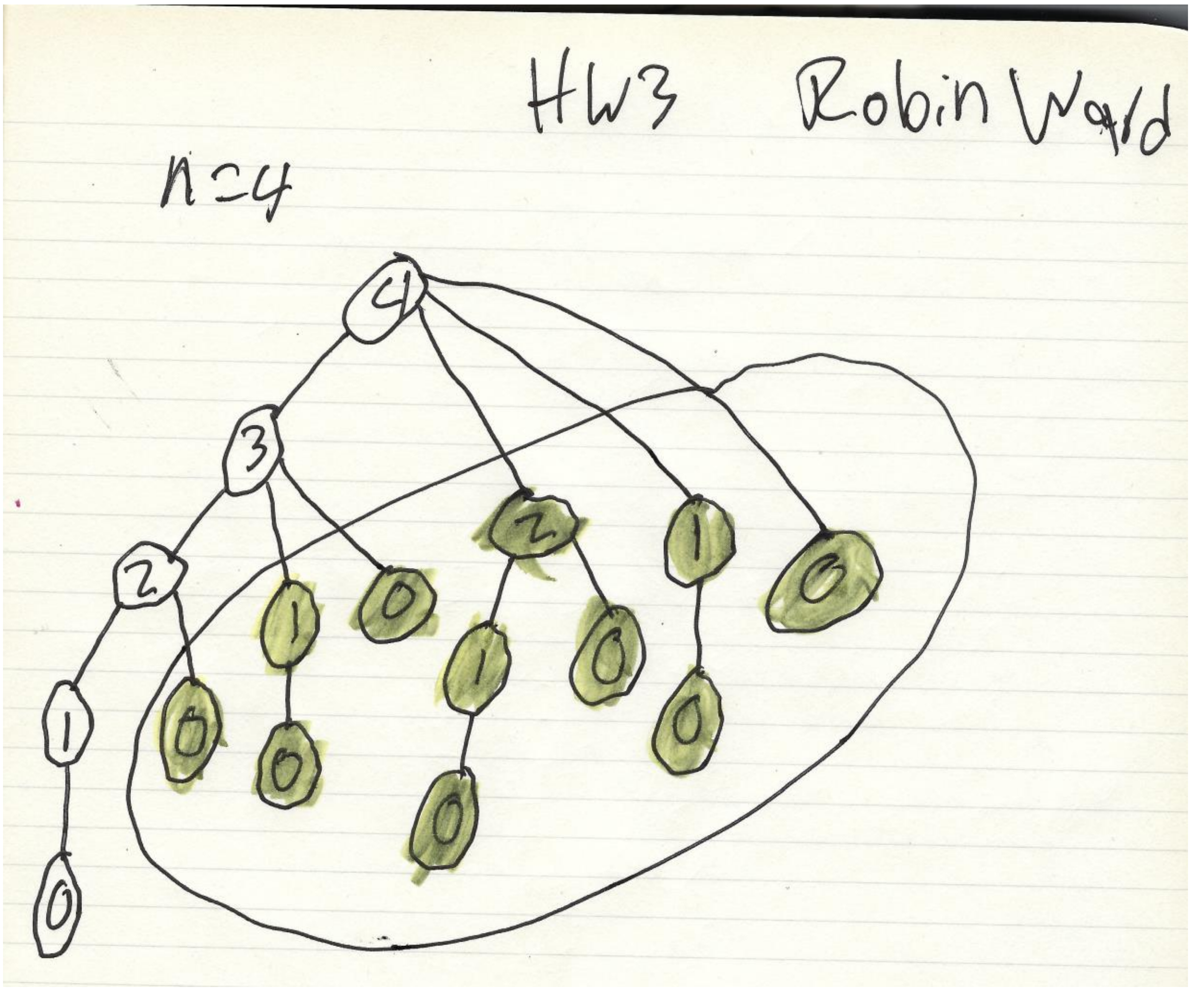
1. Develop a corresponding recursive algorithm.
2. Show how this algorithm duplicates work by drawing a recursion tree for a specific input and pointing out duplicate recursive executions.
3. Modify the recursive algorithm to make it more efficient using memoization. Is this more/less/as efficient as Memoized-Cut-Rod?
4. Develop an algorithm to compute the optimal solution in a bottom up fashion using table lookup. Is it more/less/as efficient as Bottom-Up-Cut-Rod?

4. Do the Questions 1 & 2. The specific input for which you would draw a recursion Tree should be a rod of length 4 inches.

For question 1, this was located in the book at page 363. This is the method that would take an input array and return the max revenue for each cut of a rod length of n.

```
CUT-ROD(p,n)
If n == 0
        return 0
q = -∞
for i = 1 to n
        q = max(q,p[i] + CUT-ROD(p,n – i))
Return q
```

for question 2, I was able to refer to page 364 for part of the answer. In this example we are given a value of n = 4. For subproblems, it will call itself over and over again with the same parameter values. I highlighted the duplicate work.



5. Do the Questions 3 & 4. As part of your answer. For Q4, you must explain the lookup table – what it's dimensions are and the order in which its cells will be filled by the algorithm.

For question 3, I was able to refer to the pseudocode on page 365. The MEMOIZED-CUT-ROD-AUX procedure is simply the memorized version of MEMOIZED-CUT-ROD. This will save off the value in line 8, which is r[n]. the memorization is much more efficient because we are saving off the value versus recurring over the same value.

```
MEMOIZED-CUT-ROD(p,n)
Let r[0..n] be a new array
For i = 0 to n
        r[i] = -∞
return 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
```

for question 4, the most optimal solution would be the BOTTOM-UP-CUT-ROD, which like the TOP, would run in On^2 time.

```
BOTTOM-UP-CUT-ROD(p,n)
R[0]=0
For j=1 to n
        q=-∞
        for i=1 to j
                q=max(q,p[i] + r[j-1])
        r[j]=q
return r[n]
```