Homework 5
Foundations of Algorithms
Divesh Badod

1. a. Here $p_0 = 5, p_1 = 10, p_2 = 3, p_3 = 12, p_4 = 5, p_5 = 50, p_6 = 6$
According to the algorithm for all $i = j$, $m[i, j] = 0$
$m[1,2] = m[1,1] + m[2,2] + p_0 * p_1 * p_2$
$m[1,2] = 0 + 150$
$m[1,2] = 150$
$m[3,4] = m[3,3] + m[4,4] + p_2 * p_3 * p_4$
$m[3,4] = 0 + 180$
$m[3,4] = 180$
$m[4,5] = m[4,4] + m[5,5] + p_3 * p_4 * p_5$
$m[4,5] = 0 + 3000$
$m[4,5] = 3000$
$m[5,6] = m[5,5] + m[6,6] + p_4 * p_5 * p_6$
$m[5,6] = 0 + 1500$
$m[5,6] = 1500$

$m[1,3] = \min\{\{m[1,1] + m[2,3] + p_0 * p_1 * p_3 = 750\}, \{m[1,2] + m[3,3] + p_0 * p_2 * p_3 = 330\}\} = 330$
$m[2,4] = \min\{\{m[2,2] + m[3,4] + p_1 * p_2 * p_4 = 330\}, \{m[2,3] + m[4,4] + p_1 * p_3 * p_4 = 960\}\} = 330$
$m[3,5] = \min\{\{m[3,3] + m[4,5] + p_2 * p_3 * p_5 = 4800\}, \{m[3,4] + m[5,5] + p_2 * p_4 * p_5 = 930\}\} = 930$
$m[4,6] = \min\{\{m[4,4] + m[5,6] + p_3 * p_4 * p_6 = 1860\}, \{m[4,5] + m[6,6] + p_3 * p_5 * p_6 = 6600\}\} = 1860$
$m[1,4] = \min\{\{m[1,1] + m[2,4] + p_0 * p_1 * p_4 = 580\}, \{m[1,2] + m[3,4] + p_0 * p_2 * p_4 = 405\}, \{m[1,3] + m[4,4] + p_0 * p_3 * p_4 = 630\}\} = 405$
$m[2,5] = \min\{\{m[2,2] + m[3,5] + p_1 * p_2 * p_5 = 2430\}, \{m[2,3] + m[4,5] + p_1 * p_3 * p_5 = 9360\}, \{m[2,4] + m[5,5] + p_1 * p_4 * p_5 = 2830\}\} = 2430$
$m[3,6] = \min\{\{m[3,3] + m[4,6] + p_2 * p_3 * p_6 = 2076\}, \{m[3,4] + m[5,6] + p_2 * p_4 * p_6 = 1770\}, \{m[3,5] + m[6,6] + p_2 * p_5 * p_6 = 1830\}\} = 1770$
$m[1,5] = \min\{\{m[1,1] + m[2,5] + p_0 * p_1 * p_5 = 4930\}, \{m[1,2] + m[3,5] + p_0 * p_2 * p_5 = 1830\}, \{m[1,3] + m[1,4] + p_0 * p_3 * p_5 = 6330\}, \{m[1,4] + m[1,5] + p_0 * p_4 * p_5 = 1655\}\} = 1655$
$m[2,6] = \min\{\{m[2,2] + m[3,6] + p_1 * p_2 * p_6 = 1950\}, \{m[2,3] + m[4,6] + p_1 * p_3 * p_6 = 2940\}, \{m[2,4] + m[5,6] + p_1 * p_4 * p_6 = 2130\}, \{m[2,5] + m[6,6] + p_1 * p_5 * p_6 = 5430\}\} = 1950$
$m[1,6] = \min\{\{m[1,1] + m[2,6] + p_0 * p_1 * p_6 = 2250\}, \{m[1,2] + m[3,6] + p_0 * p_2 * p_6 = 2010\}, \{m[1,3] + m[4,6] + p_0 * p_3 * p_6 = 2550\}, \{m[1,4] + m[5,6] + p_0 * p_4 * p_6 = 2055\}, \{m[1,5] + m[6,6] + p_0 * p_5 * p_6 = 3155\}\} = 2010$

So, the m table is

0	150	330	405	1655	2010
0	0	360	330	2430	1950
0	0	0	180	930	1770
0	0	0	0	3000	1860
0	0	0	0	0	1500
0	0	0	0	0	0

We construct s table using the c table,

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

The minimum cost is 2010 and the optimal parenthesization is $(A_1.A_2).\big((A_3.A_4).(A_5.A_6)\big)$

b. Let's consider parenthesizing two matrices, because of that these two matrices will be converted into one matrix after multiplication. As this process goes on, for n number of matrices it will occur n-1 times before there is only single matrix. As a pair of parentheses at least contains one single operator and in similar way n elements must have n-1 operators, hence a full parenthesization of an n-element expression requires exactly n-1 pairs of parentheses.

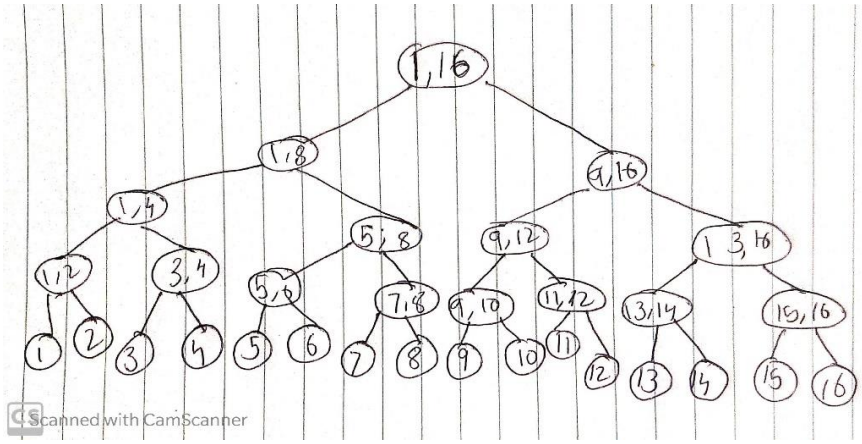
2. a. In Enumeration approach

- All the ways to parenthesize the left half is found
- All the ways to parenthesize right half is found
- Then all possible combinations of left half with the right half is done
- So, the total amount of work for each combinations results in overall running time of $\Omega\left(\frac{4^n}{n^{\frac{3}{2}}}\right)$

In Recursive-Matrix-Chain approach

- One best way to parenthesize the left half is found
- One best way to parenthesize the right half is found
- The is combines the two results obtained from the above two steps
- Thus the amount of work results in overall running time of $O(n3^{n-1})$

Hence the Recursive-Matrix-Chain is more efficient than enumeration.

b. 

Memoization requires the solutions of the sub-problems and reuse them when required. But since an algorithm like merge-sort needs the solutions of the subproblems only once using memoization will only use more space than required and doesn't actually help in speeding up a divide and conquer algorithm.

c. Yes this problem exhibits optimal substructure property

In this case we will search for the maximum value to slice the matrix in rather than minimum value to multiply. The recurrence for the subproblem optimality will be

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

Considering matrix-chain multiplication for $A_i, A_{i+1}, \dots \dots A_j$ with two sub-problems and $j - i$ choices. For a given matrix $A_k$at which we split the product, we have two subproblems of parenthesizing $A_i, A_{i+1}, \dots \dots A_k$ and $A_{k+1}, A_{k+2}, \dots \dots A_j$. We can determine the optimal solutions to these subproblems, we choose from $j - i$ choices for the value of index k where the splitting will take place. Thus the best split point can be obtained by considering the cost of multiplying two sub-chains and the cost of each sub-chain optimally parenthesized.

3. Electronic submission

4. a. Knapsack(n,W)

```
If n ≤ 0
        Return 0
If W < n.w
        Lastitem = -1
Else
        Lastitem = n.v + Knapsack(n-1, W-n.w)
NotLastItem = Knapsack(n-1,W)
Return max(lastItem, NotLastItem)
```

n – no of item
v – cost of the items
w – weight of the items
W – capacity of the knapsack

b. Electronic Submission

c. Electronic Submission

5. a. Pseudo Code:-

```
e(S, M, i, j)
    if words don't fit in i and j
        return M
    else
        return M – j + I – (sum of all the lengths of strings(words) in S)
```

b. Electronic Submission

c. Pseudo Code : -

```
bl(S, M, I, j)
    if M < 0 or e(S,M,i,j) < 0
        return ∞
    else
        return e(S, M, i, j)
```

d. Electronic Submission

e. Pseudo Code: -

```
mb(S, M)
    if e(S, M, 0, j) >= 0
        bl(S, M, 0, ∞)
    else:
        bl(S[ :mb`(S, M, 0)], M, 0, mb`(S, M))+ mb(S[: mb`(S, M, 0)]
```

f. Pseudo Code: -

```
mb`(S, M, i)
    if i > |S| or e(S[: i], M, 0, ∞) < 0
        return i – 1
    else
        mb`(S, M, i+1)
```

g. Electronic Submission

h. Electronic Submission

i. Electronic Submission. The asymptotic runtime of the algorithm is $O(n^2)$

6. First we start a *z* labelling it as '0', following the algorithm we go to *s* and label it as '2' since we travelled that much weight through the edge to reach *s*, then we go to *x* from *z* since there are two paths from *z* and label it as '7'. We now travel from *s* to *y* and label *y* as the total weight travelled which is 9 and from *x* we go to *t* which becomes 7 + (-2) = 5 and label *t* as '5' from *y* we go to *x* again and label it as 9 + (-3) = 6. We get

u = z

V:	S	T	X	Y	Z
Π:	Z	X	Y	S	NIL
D:	2	5	6	9	0

Here the algorithm returns true

Now change the edge *z*-*x* to 4 and changing the source to *s*

Starting with *s* we label it '0' then the two adjacent vertices *t* as '6' and *y* as '7'After that we visit *x* and label it as 7 + (-3) = 4 from *t* we go to *z* and label it as 6 + (-4) = 2 from *x* we can revisit *t* and re-label it as 4 + (-2) = 2 and from *t* we revisit *z* and re-label it as 2 + (-4) = -2. We get

u = s

V:	S	T	X	Y	Z
Π:	NIL	X	Y	S	T
D:	0	2	4	7	-2

Here the algorithm returns false.