1.

a) T(n)=3T(n/2)+n

Here a =3 and b=2 and f(n) = n

Since we have $n^{\log_b a} = n^{\log_2 3} > n^1$ This satisfies case 1 of master theorem. if f(n) =O($n^{r-\epsilon}$)for ε>0 then T(n) =Θ(n^r)

So according to case 1 The complexity of this recursive will be **Θ($\boldsymbol{n^{\log_2 3}}$)**

b) T(n)=3T(n/2)+n(log of 3 to the base 2)

Here we have a= 3 and b=2 and f(n) = $n^{\log_2 3}$. This satisfies case 2 of the master theorem if f(n) =Θ(n^r) then T(n) =Θ(n^rlg n)

So according to case2 The complexity of this equation is **Θ($\boldsymbol{n^{\log_2 3} lgn}$)**

c) T(n)=3T(n/2)+n^3

Here we have a =3 and b=2. f(n) = n^3. We have $n^{\log_b a} = n^{\log_2 3} < n^3$

This satisfies case 3 of the masters theorem.

if f(n) =Ω($n^{r+\varepsilon}$) for ε>0 and if af(n/ b) ≤cf(n) for c<1 then T(n) =Θ(f(n))

Here we have af(n/ b)= 3 f(n/2) ≤ cf(n)= cn^3 since the conditions for case 3 is satisfied.

The Complexity of this equation is **Θ($\boldsymbol{n^3}$)** for 3/8 $\leq$ c $<$ 1 given matrix-chain <5, 10, 3, 12, 5, 50, 6> .The matrices has the following dimensions.

A1 = 5 x 10

A2 = 10 x 3

A3 = 3 x 12

A4 = 12 x 5

A5 = 5 x 50

A6 = 50 x 6

Consider P0 = 5, P1 = 10, P2 = 3, P3 = 12, P4 = 5, P5 = 50, P6 = 6

The recursive equation is given as

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

therefore, m[1, 1] = 0, m[2, 2] = 0, m[3, 3] = 0, m[4, 4] = 0, m[5, 5] = 0, m[6, 6] = 0

m[1, 2] = m[1, 1] + m[2, 2] + (P0 × P1 × P2) = 0 + 0 + (5 × 10 × 3) = 150

m[2, 3] = m[2, 2] + m[3, 3] + (P1 × P2 × P3) = 0 + 0 + (10 × 3 × 12) = 360

m[3, 4] = m[3, 3] + m[4, 4] + (P2 × P3 × P4) = 0 + 0 + (3 × 12 × 5) = 180

m[4, 5] = m[4, 4] + m[5, 5] + (P3 × P4 × P5) = 0 + 0 + (12 × 5 × 50) = 3000

m[5, 6] = m[5, 5] + m[6, 6] + (P4 × P5 × P6) = 0 + 0 + (5 × 50 × 6) = 1500

m[1,3] = min{{m[1,1] + m[2,3] + p0p1p3},{m[1,2] + m[3,3] + p0p2p3} = 330

m[2, 4] = min{{m[2, 2] + m[3, 4] + p1p2p4 },{ m[2, 3] + m[4, 4] + p1p3p4 }} = 330
m[3, 5] = min{{m[3, 3] + m[4, 5] + p2p3p5},{m[3, 4] + m[5, 5] + p2p4p5}} = 930
m[4, 6] = min{{m[4, 4] + m[5, 6] + p3p4p6},{m[4, 5] + m[6, 6] + p3p5p6}} = 1860

m[1, 4] = min{{m[1, 1] + m[2, 4] + p0p1p4},{ m[1, 2] + m[3, 4] + p0p2p4}, {m[1, 3] + m[4, 4] + p0p3p4}} = 405
m[2, 5] = min{{m[2, 2] + m[3, 5] + p1p2p5},{m[2, 3] + m[4, 5] + p1p3p5},{m[2, 4] + m[5, 5] + p1p4p5}} = 2430
m[3, 6] = min{{m[3, 3] + m[4, 6] + p2p3p6},{m[3, 4] + m[5, 6] + p2p4p6},{ m[3, 5] + m[6, 6] + p2p5p6}} = 1770

m[1, 5] = min{{m[1, 1] + m[2, 5] + p0p1p5},{m[1, 2] + m[3, 5] + p0p2p5},{m[1, 3] + m[4, 5] + p0p3p5},{m[1, 4] + m[5, 5] + p0p4p5}} = 1665
m[2, 6] = min{{m[2 ,2] + m[3, 6] + p1p2p6},{m[2, 3] + m[4, 6] + p1p3p6},{m[2, 4] + m[5, 6] + p1p4p6},{m[2, 5] + m[6, 6] + p1p5p6}} = 1950

m[1, 6] = min{{m[1, 1] + m[2, 6] + p0p1p6},{m[1, 2] + m[3, 6] + p0p2p6 },{m[1, 3] + m[4, 6] + p0p3p6 },{ m[1, 4] + m[5, 6] + p0p4p6},{m[1, 5] + m[6, 6] + p0p5p6}} = 2010

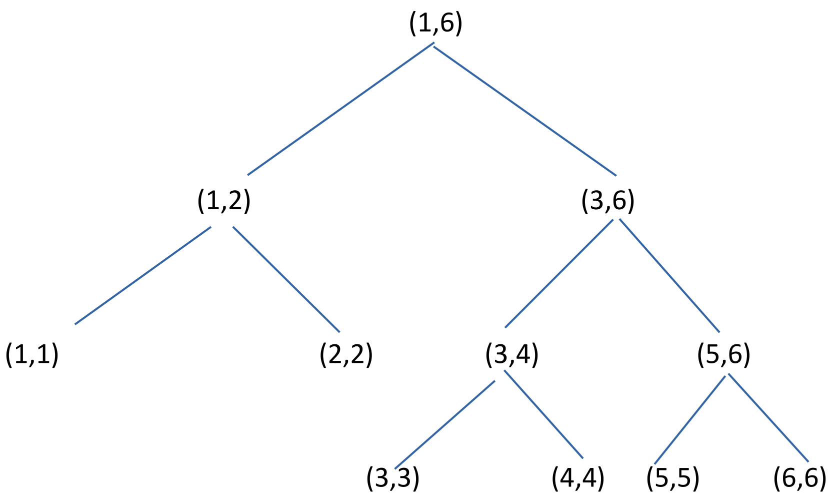
m table is

m	1	2	3	4	5	6
1	0	150	330	405	1655	2010
2		0	360	330	2430	1950
3			0	180	930	1770
4				0	3000	1860
5					0	1500
6						0

s table is as follows

s	1	2	3	4	5	6
1	0	1	2	2	4	2
2		0	2	2	2	2
3			0	3	4	4
4				0	4	4
5					0	5
6						0

Tree for optical parenthesization



From the tree above we get the final multiplication sequence as (A1A2)(A3A4)(A5A6)

2. The equation given in 16.2 is as follows

$$c[i,j] = \begin{cases} 0 & if\ s_{ij} = 0 \\ \max\limits_{a_k \in s_{ij}} \{c[i,k] + c[k,j] + 1\} & if\ s_{ij} \neq 0 \end{cases}$$

Let us consider a 2D-array A[1..2,1…m] where the first index let us know whether its start time or end time and the second index tells us about the activity number. S_{ij} is the set of activities that begin after i and end before start of j.

```
Activity_Selector(i,j)
    int k
    S_ij =[]                //empty array
    for k = i+1 to j-1
        If(A[1,k]>=i   and  A[2,k]<=j)
            S_ij.add(k)   // adding the activities which start after i and end before j
    If S_ij  is empty
        return 0
    else
        Max=0
        for k  in  S_ij
            Max =   max(Max, Activity_Selector(i,k)+Activity_Selector(k,j)+1)
    return Max
```

3.

The equation is as follows

$$c[i,j] = \begin{cases} 0 & , if\ i = 0\ or\ j = 0 \\ c[i-1, j-1] + 1 & , if\ i,j > 0\ and\ x_i = y_j \\ max(c[i, j-1], c[i-1, j]) & , if\ i,j > 0\ and\ x_i \neq y_j \end{cases}$$

Let c[0…m,0…n] be a 2D Array representing the LCS of X and Y.

```
RECURSIVE-MEMOIZED-LCS-LENGTH(X,Y)
    c[0…m,0…n] , m= length of X , n= length of Y , int i, int j
    for  i = 0 to m
      for  j = 0 to n
           c[i,j] = inf
     Return LCS_Table(X,Y,m,n)
```

```
LCS_Table(X,Y,i,j)
    if   c[i,j] < inf
        return c[i,j]
    if i== 0 or j == 0
            c[i, j] = 0
  else     if X[i] == Y[j]
                c[i, j] =  LCS_Table(X,Y,i-1, j - 1) + 1
            else
                c[i, j] = max(LCS_Table(X, Y, i - 1, j), LCS_Table(X, Y, i, j - 1))
     return c[i,j]
```

4.

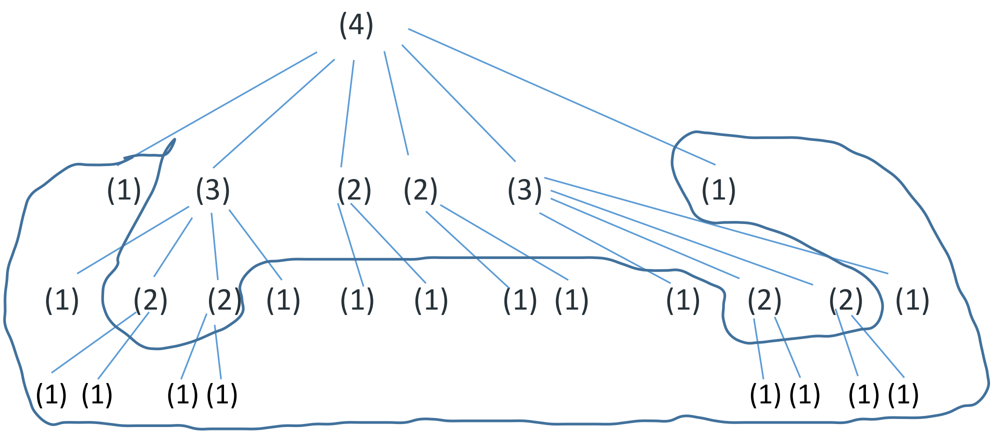
The equation 15.1 is given as

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

The Recursive algorithm is given as

```
Recursive_Cut_Rod(p:array[1…n],n)
    if n == 1
            return p[1]
   q = -inf
    for i= 1 to n-1
            q = max(q, Recursive_Cut_Rod(p,i) +  Recursive_Cut_Rod(p, n-i))
    return max(q,p[n])
```

The recursion tree for a rod of length 4 is



From the tree we can see how duplication work is done as (1) is computed 18 times.

5.

```
Recursive_Memoized_Cut_Rod (p,n)          // p is an array p[1…n]
    let r[1..n]  be new arrays
    for i= 1 to n
         r[i] = - INF
    return  Recursive_Memoized_Cut_Rod_Aux(p, n, r)

Recursive_Memoized_Cut_Rod_Aux(p, n, r)       // p is an array p[1…n]
    if r[n] ≥ 0
         return r[n]
    if n == 1
         q = p[1]
    else q = - INF
           for i = 1 to n-1
                  q = max(q, Recursive_ Memoized_Cut_Rod_Aux(p,  i,r) +
                                  Recursive_Memoized_Cut_Rod_Aux(p,n-i,r))
           q = max(q,p[n])
    r[n] = q
    return q
```

In terms of complexity Memoized-Cut-Rod   and Recursive_Memoized_Cut_Rod has the same complexity.

```
New_Bottom_Up_Cut_Rod(p, n)
    let r[1...n] be a new array
    r[1] = p[1]
    for j = 2 to n
       q = - INF
       for i = 1 to j - 1
              q = max(q, r[i] + r[j – i])
       q= max(q,p[j])
       r[j] = q
    return r[n]
```

The look up table r of dimension 1x n provides the information of r[i] = optimal solution for an i inch rod. It is filled in the order r[1],r[2],………r[n]

In terms of complexity both Bottom_Up_Cut_Rod and New_Bottom_Up_Cut_Rod has the same complexity.