1.

1. T(n)=3T(n/2)+n

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

Since we have > This satisfies case 1 of master theorem. if f(n) =O( )for ε>0 then T(n) =Θ( )

So according to case 1 The complexity of this recursive will be  **Θ( )**

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

Here we have a= 3 and b=2 and f(n) = . This satisfies case 2 of the master theorem if f(n) =Θ( ) then T(n) =Θ( lg n)

So according to case2 The complexity of this equation is **()**

1. T(n)=3T(n/2)+

Here we have a =3 and b=2. f(n) = . We have <

This satisfies case 3 of the masters theorem.  
if f(n) =Ω() 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)= c since the conditions for case 3 is satisfied.

The Complexity of this equation is **Θ( )**  for 3/8 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 , if i = j,

m[i,j]= , 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

(1,6)

(1,2) (3,6)

(1,1) (2,2) (3,4) (5,6)

(3,3) (4,4) (5,5) (6,6)

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

1. The equation given in 16.2 is as follows

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. is the set of activities that begin after i and end before start of j.

Activity\_Selector(i,j)

1) int k

2) =[] //empty array

3) for k = i+1 to j-1

4) If(A[1,k]>=i and A[2,k]<=j)

5) .add(k) // adding the activities which start after i and end before j

6) If is empty

7) return 0

8) else

9) Max=0

10) for k in

11) Max = max(Max, Activity\_Selector(i,k)+Activity\_Selector(k,j)+1)

12) return Max

3.

The equation is as follows

c[i,j] =

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

RECURSIVE-MEMOIZED-LCS-LENGTH(X,Y)

1. c[0…m,0…n] , m= length of X , n= length of Y , int i, int j
2. for i = 0 to m
3. for j = 0 to n
4. c[i,j] = inf
5. Return LCS\_Table(X,Y,m,n)

LCS\_Table(X,Y,i,j)

1. if c[i,j] < inf
2. return c[i,j]
3. if i== 0 or j == 0

4) c[i, j] = 0

5) else if X[i] == Y[j]

6) c[i, j] = LCS\_Table(X,Y,i-1, j - 1) + 1

7) else

8) c[i, j] = max(LCS\_Table(X, Y, i - 1, j), LCS\_Table(X, Y, i, j - 1))

9) return c[i,j]

4.

The equation 15.1 is given as

The Recursive algorithm is given as

Recursive\_Cut\_Rod(p:array[1…n],n)

1. if n == 1

2. return p[1]

3. q = -inf

4. for i= 1 to n-1

5. q = max(q, Recursive\_Cut\_Rod(p,i) + Recursive\_Cut\_Rod(p, n-i))

6. return max(q,p[n])

The recursion tree for a rod of length 4 is

(4)

(1) (3) (2) (2) (3) (1)

1. (2) (2) (1) (1) (1) (1) (1) (1) (2) (2) (1)

1. (1) (1) (1) (1) (1) (1) (1)

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]

1 . let r[1..n] be new arrays

2. for i= 1 to n

3. r[i] = - INF

4. return Recursive\_Memoized\_Cut\_Rod\_Aux(p, n, r)

Recursive\_Memoized\_Cut\_Rod\_Aux(p, n, r) // p is an array p[1…n]

1. if r[n] ≥ 0

2. return r[n]

3. if n == 1

4. q = p[1]

5. else q = - INF

6. for i = 1 to n-1

7. q = max(q, Recursive\_ Memoized\_Cut\_Rod\_Aux(p, i,r) + Recursive\_Memoized\_Cut\_Rod\_Aux(p,n-i,r))

8. q = max(q,p[n])

9. r[n] = q

10. 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)

1. let r[1...n] be a new array

2. r[1] = p[1]

3. for j = 2 to n

4. q = - INF

5. for i = 1 to j - 1

6. q = max(q, r[i] + r[j – i])

7. q= max(q,p[j])

8. r[j] = q

9. 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.