**COMP 7270 Assignment 2 10 Problems 200 points No late submissions!**

**Due by 11:59 PM 03/31(Friday)**

U**pload your submission well before this deadline. Even if you are a few minutes late, as a result of which Canvas marks your submission late,** **your assignment may not be accepted**.

Instructions:

1. This is an individual assignment. You should do your own work. **Any evidence of copying will result in a zero grade and additional penalties/actions.**
2. Late submissions **will not** be accepted unless prior permission has been granted or there is a valid and verifiable excuse.
3. **Think carefully; formulate your answers, and then write them out concisely** using English, logic, mathematics and pseudocode (no programming language syntax).
4. Algorithms should be provided in numbered pseudocode steps.
5. **Type your answers in this Word document and submit it. If that is not possible, use a word processor to type your answers as much as possible (you may hand-write/draw equations and figures), turn it into a PDF document and upload**.

All questions carry equal weight

1. Use the Detailed Method to determine the precise T(n) of the following iterative maximum subsequence sum (MSS) algorithm. You must show your work below to get any credit. The algorithm is as below.

MSS Algorithm-1 (A:array[p..q] of integer)

sum, max: integer

1 sum = max = 0

2 for i = p to q

3 sum = 0

4 for j = i to q

5 sum = sum + A[j]

6 if sum > max then

7 max = sum

8 return max

|  |  |  |  |
| --- | --- | --- | --- |
| Line # | Step | Single execution cost | # times executed |
| 1 | sum = max = 0 | 2 | 1 |
| 2 | for i = p to q | 1 | q-p+1+1 = n+1  (assume :n=q-p+1) |
| 3 | sum = 0 | 1 | n =q-p+1 |
| 4 | for j = i to q | 1 | (1+n)\*n/2 |
| 5 | sum = sum + A[j] | 6 | (1+n)\*n/2 |
| 6 | if sum > max then | 3 | (1+n)\*n/2 |
| 7 | max = sum | 2 | (1+n)\*n/2 |
| 8 | return max | 2 | 1 |

If n = q-p+1

T(n)=1\*2+(n+1)+n+[(1+n)\*n/2]+6\*[(1+n)\*n/2]+3\*[(1+n)\*n/2]+2\*[(1+n)\*n/2]+2=6n2+8n+3

**2****.** Develop, state and solve the recurrence relations of the Recursive Divide & Conquer iterative algorithm as follows by answering the following questions.

MSS Algorithm-2 (A:array[p..q] of integer)

1 if p=q then

2 if A[p] > 0 then

3 return A[p]

4 else return 0

5 (left-partial-sum = right-partial-sum) =( max-right = max-left )= left-max-sum = right-max-sum = 0

6 center = floor((p+q)/2)

7 max-left = **Algorithm-2**(A[p..center])

8 max-right **= Algorithm-2**(A[center+1..q])

(9 for i from center downto p do

10 left-partial-sum = left-partial-sum + A[i]

11 if left-partial-sum > left-max-sum then

12) left-max-sum = left-partial-sum

(13 for i from center+1 to q do

14 right- partial-sum = right-partial-sum + A[i]

15 if right- partial-sum > right-max-sum then

16) right-max-sum = right- partial-sum

17 if max-left≤max-right then

18 if max-right≤left-max-sum+right-max-sum then

19 return left-max-sum+right-max-sum

20 else return max-right

else

21 if max-left<left-max-sum+right-max-sum then

22 return left-max-sum+right-max-sum

23 else return max-left

You must state costs in terms of n with numerical coefficients, and not using a complexity order notation, to get credit. You may assume that the for loops on lines 9-12 and 13-16 are executed n/2 times.

Which statements are executed when the input is a base case (provide line #s)? \_\_(1~4)\_\_

What is the total cost of these? \_\_\_\_\_\_\_\_8\_\_\_\_\_\_\_\_\_

Which statements are executed when the input is not a base case (provide line #s)? \_1, \_5～23\_\_

What is the total cost of these?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2T(n/2)+8n+23\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Provide the complete and precise two recurrence relations characterizing the complexity of CountPairs:

T(n) = \_\_\_8\_\_\_\_\_\_\_\_ when n=1

T(n) = \_ 2T(n/2)+8n+23\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ when n>1

Now simplify the recurrence relations by:

1. If your recurrence relation for the non base case input has multiple terms in it besides the term representing the recursive calls, keep only the largest n-term from them and drop the others; if your recurrence relation for the non base case input has only one other term besides the term representing the recursive calls, keep it.

2. Take the largest numerical coefficient of all terms (excluding the term representing the recursive calls) in your two recurrence relations, round it up to the next integer if it is not an integer, and replace the numerical coefficients of all other terms (excluding the term representing the recursive calls) with this coefficient.

Provide the simplified recurrence relations below.

T(n) = \_\_\_\_\_\_\_\_\_8\_\_\_\_\_\_\_ when n=1

T(n) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2T(n/2)+8n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ when n>1

Solve these recurrence relations using the Recursion Tree method, determine and state the T(n) of the algorithm. You must show your work below to get any credit.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Of recursive execution | Input size to each execution | Additional work done by each execution | Total work done at this level |
| 0 | 20 | n | 8n | 8n |
| 1 | 21 | n/2 | 8(n/21)=8n/2 | 8n |
| 2 | 22 | n/4 | 8(n/22)=8n/4 | 8n |
| Lg(n-1) | n/2 | 2 | 16 | 8n |
| lgn | n | 1 | 8 | 8n |

T(n)=8n\*(lgn+1)=8nlgn+8n

**3.** Let Result[1..2] be an array of two integers. Modify steps of the Iterative MSS algorithm as in Question 1 to return the starting and ending indexes of the optimal (maximum) subsequence it found in its last line instead of the maximum sum value. Provide your modified algorithm below. Make only the minimum number of modifications necessary. You will need to add additional steps.

MSS- Modified- Algorithm (A:array[p..q] of integer)

sum, max: integer

let array Result[1,2] be a new array of two integers

1 sum = max = 0

2 for i = p to q

3 sum = 0

4 for j = i to q

5 sum = sum + A[j]

6 if sum > max then

7 max = sum

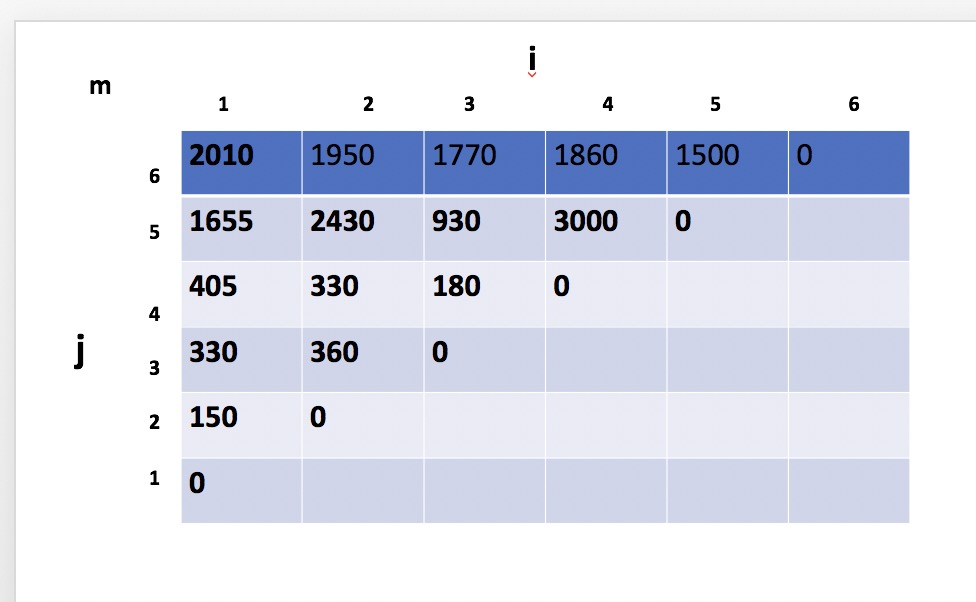
8 Result[1] = i

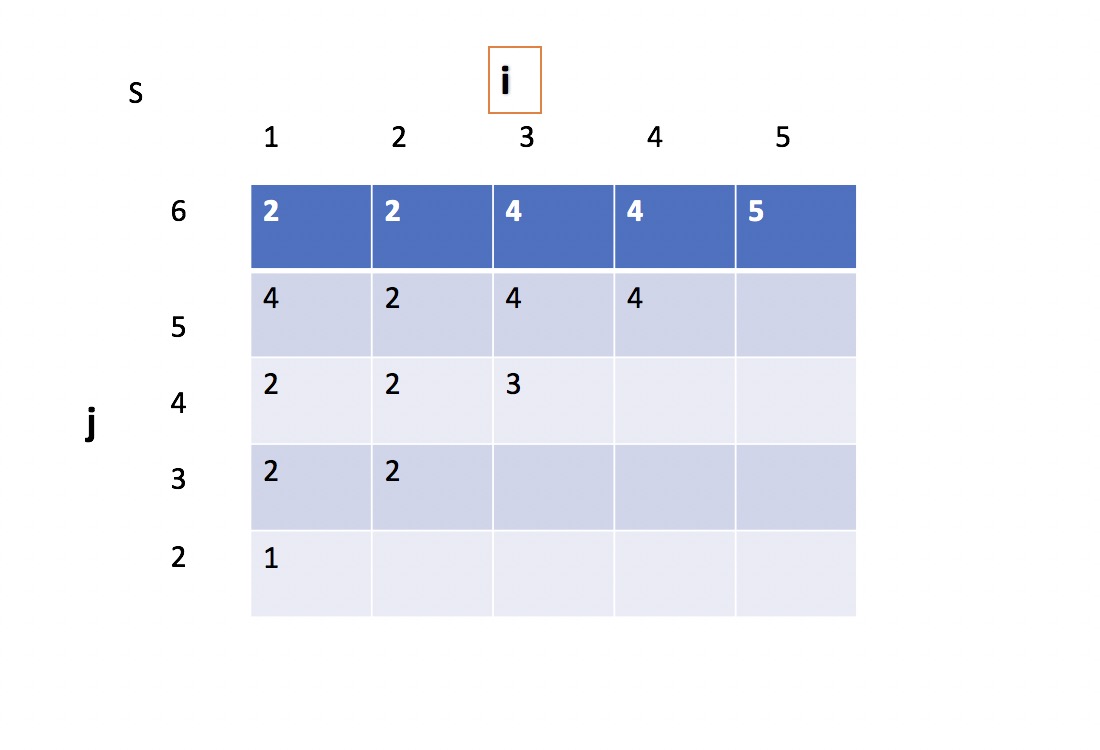
9 Result[2] = J

10 return Result

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

A1 = 5\*10 A2=10\*3 A3=3\*12 A4=12\*5 A5=5\*50 A6=50\*6





the optimal parenthesization is (A1,A2)(A3,A4)(A5,A6)

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

RECURSIVE\_ACTIVITY\_SELECTOR(s,f,i,j):

1 c[i,j] = 0

2 if Sij = ∅

3 return c[i,j]

4 else

5 for k = i+1 to j-1

6 if f[i]<= s[k] and f[k]<= s[j]

7 q = RAS(s,f,i,k) + RAS(s,f,k,j) + 1

8 if c[i,j] < q

9 c[i,j] = q

10 return c[i,j]

**6.** Problem 16.1-2 in the text. To prove that the stated approach yields an optimal solution, we have to prove two things: (1) that the choice being made is the greedy choice (this proof will be along the lines of the proof of Theorem 16.1 in the text; but do not copy that proof; your proof will be different, yet similar in structure), and (2) that the resulting solution has optimal substructure.

(1)Let Ak  be a max size subset of mutably compatible activities in Sk and let ai be the activity in Ak with the last start time , if ai = am done , if if ai ≠ am ,let set Ak′ = Ak – { ai }∪{am} be Ak  but substituting am for ai, Ak′ are disjoint, which follow because the activities in Ak are disjoint, ai is the last activity in Ak  to start and sm≥si , since | Ak′ |=| Ak  |, we conclude Ak′  is the maxmium subset of mutually compatible activities of sk and include am

(2) proof the resulting solution has optimal substructure

Assume Sij is the set of activities that start after activity ai finished and that finish before activity aj starts. Assume Aij is the maximum set of mutually compatible activities in Sij, and Aij includes activity ak, so we can divide the Sij into two subproblems, Sik ,and Skj. In order to prove the resulting solution has optimal substructure ,we have to prove optimal solution Aij include optimal solutions to Sik and Skj.

Let A Let Aik = Aij intersect Sik , Akj = Aij intersect Skj, then, Aij = Aik union {ak} union Akj. Thus, |Aij| = |Aik|+|Akj| + 1. If we could find a subset Aik’ of mutually compatible activities in Sik where |Aik’|>|Aik|, then Aik’ is part of solution for Sij. Then we have |Aik’| + |Akj| + 1 > |Aik| + |Akj| + 1 = |Aij|. However, Aij is an optimal solution. Aik’ cannot exist. Thus, Aij include the optimal solutions Aik and Akj to the two subproblems for Sik and Skj

**7****.** Modify the FASTEST-WAY algorithm for two-line Assembly Line Scheduling to suit a factory with three assembly lines with n stations. Use a similar notation for matrices a, e, x, f and l. Matrix t is to be interpreted as follows: it is a 3-dimensional matrix with entry tijk, 1≤i≤(n-1), 1≤j, k≤3 is the transfer cost of the product to move it, after work on it at station i is finished on line j, to line k so that work on it at station (i+1) will be done on line k.

FASTEST-WAY(a,t,e,x,I,f)

1 f1[1] = e1+a1,1

2 f2[1] = e2+a2,1

3 f3[1] = e3+a3,1

4 for i = 2 to n

k =1

5 if f1[i-1] +a1,i ≤ f2[i-1]+ti-1,2,1+a1,i

6 f1[i-1] +a1,i ≤ f3[i-1]+ ti-1,3,1+a1,i

7 then f1[i]= f1[i-1] +a1,j

8 l1[i]=1

9 elseif f2[i-1]+ti-1,2,1+a1,i ≤ f3[i-1]+ ti-1,3,1+a1,i

10 then f1[i]= f2[i-1]+ti-1,2,1+a1,i

11 l1[i]=2

12 elseif f1[i]= f3[i-1]+ ti-1,3,1+a1,i

13l1[i]=3

k=2

14 if f2[i-1] +a2,i ≤ f1[i-1]+ti-1,1,2+a2,i

15 f2[i-1] +a2,i ≤ f3[i-1]+ ti-1,3,2+a2,i

16 then f2[i]= f2[i-1] +a2,i

17l2[i]=2

18 elseif f3[i-1]+ti-1,3,2+a2,i ≤ f1[i-1]+ ti-1,1,2+a2,i

19then f2[i]= f3[i-1]+ti-1,3,2+a2,i

20l2[i]=3

21 elseif f1[i]= f1[i-1]+ ti-1,1,2+a2,i

22l2[i]=1

k=3

23 if f3[i-1] +a3,i ≤ f1[i-1]+ti-1,1,3+a3,i

24 f3[i-1] +a3,i ≤ f2[i-1]+ ti-1,2,3+a3,i

25 then f3[i]= f3[i-1] +a3,i

26l3[i]=3

27 elseif f1[i-1]+ti-1,1,3+a3,i ≤ f2[i-1]+ ti-1,2,3+a3,i

28 then f3[i]= f1[i-1]+ti-1,1,3+a3,i

29 l3[i]=1

30 elseif f3[i]= f2[i-1]+ ti-1,2,3+a3,i

31l1[i]=2

**8.** 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.

LCS-MEMOIZED-LENGTH(X,Y, i, j)

1 m = X.length

2 n = Y.length

3 let c[0..m,0…n]be new table

4 for i = 0 to m

5 for j = 0 to n

6 c[i,j] = -∞

7 RECURSIVE-LCS- LOOKUP(x,y,m,n,c)

8 return c

RECURSIVE-LCS -LOOKUP(X,Y,i,j,c)

1 if c[i,j] > 0

2 return c[i,j]

3 if i =0 or j = 0

4 then c[i,j] = 0

5 elseif Xi=Yi

6 c[i,j]= RLL(X,Y,i-1, j -1,c) +1

7 elseif Xi != Yi

8 c[I,j]=max(RLL(X,Y,i-1,j,c),RLL(X,Y,i,j-1,c))

9 return c[i,j]

**9.** Do the Questions 1 & 2 on the Thinking Assignment on slide 22,

7270-10-DPandGreedyAlgorithmDesign Part I.pptx The specific input for which you would draw a recursion Tree should be a rod of length 4 inches.

CUT-ROD(p[1…n],n)

If n==1

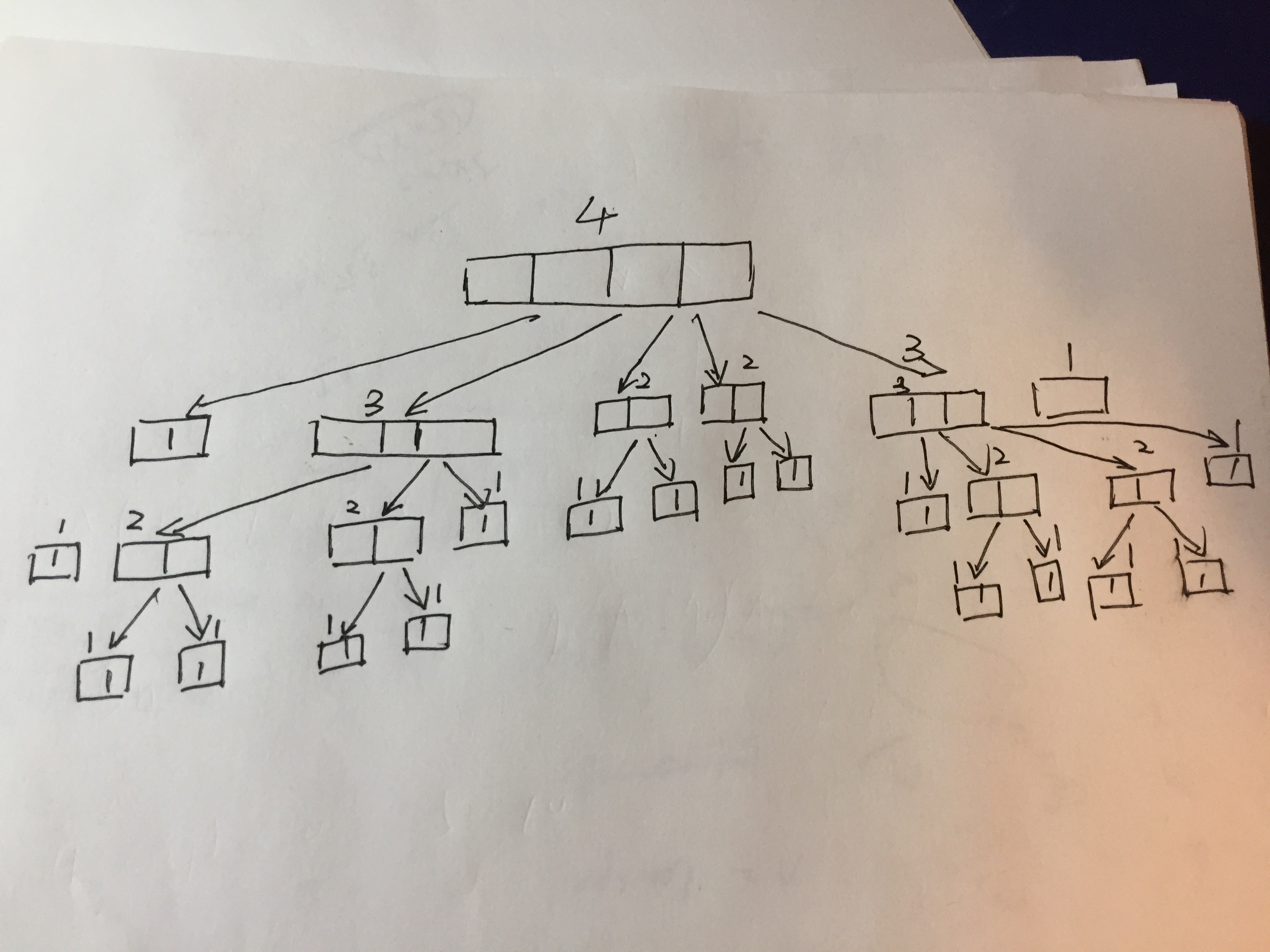
Return p[1]

q = -∞

for i = 1 to n-1

q =max[q , CUT-ROD(p,i)+ CUT-ROD(p,n-i)]

return max(q, pn)



**10.** Do the Questions 3 & 4 on the Thinking Assignment on slide 22,

7270-10-DPandGreedyAlgorithmDesign Part I.pptx As part of your answer for Q.4, you must explain the lookup table – what it’s dimensions are and the order in which its cells will be filled by the algorithm.

CUT-ROD-MEMOIZED(p,n)

1 Let r[1….n]be a new array

2 for i = 1 to n

3 r[i] = -∞

4 return CUT-ROD-MEMOIZED-Aux(p,n,r)

CUT-ROD-MEMOIZED-Aux(p,n,r)

1 If r[n] ≥ 0

2 return r[n]

3 if n==1

4 q = p

5 else q = -∞

6 for i = 1 to n

7 q = max(q, CUT-ROD-MEMOIZED-Aux(p,i,r)+ CUT-ROD-MEMOIZED-Aux(p,n-i,r) )

8 r[n] = q

9 return q

this is less efficient than the Memoized-Cut –Rod

Bottom – up algorithm

Bottom – Up- Cut-Rod(p,n)

1 Let r [0…n]be a new array

2 r[0]=0

3 for j =1 to n

4 q = -∞

5 for i=1 to j-1

6 q = max(q, r[i]+r[j-i])

7 q=max(pj,q)

8 r[j]=q

9 return r[n]

this is same efficient as the usual one ,both of which complexity is Θ(n2)

because i set up a array to store the value ,so the dimension of lookup table should be 1\*n, with the order r[1],r[2],r[3]…..r[n]