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

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

Upload 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

$T(n)=1*2+(n+1)+n+[(1+n)*n/2]+6*[(1+n)*n/2]+3*[(1+n)*n/2]+2*[(1+n)*n/2]+2=6n^2+8n+3$

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	2^0	n	8n	8n
1	2^1	n/2	8(n/2^1)=8n/2	8n
2	2^2	n/4	8(n/2^2)=8n/4	8n
Lg(n-1)	n/2	2	16	8n
lgn	n	1	8	8n

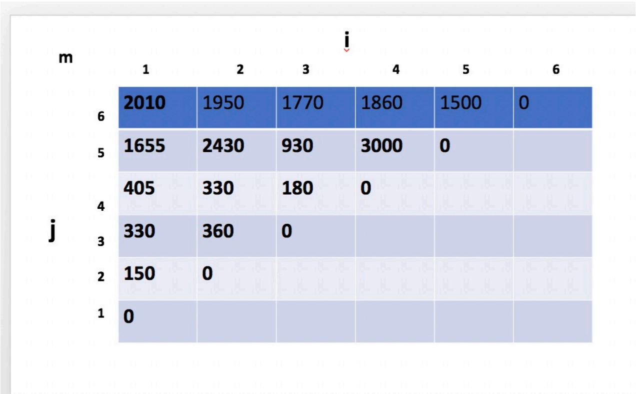
T(n)=8n*(lgn+1)=8nlgn+8n

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
sum = max = 0
for i = p to q
        sum = 0
        for j = i to q
                sum = sum + A[j]
                if sum > max then
                        max = sum
                        Result[1] = i
                        Result[2] = J
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.

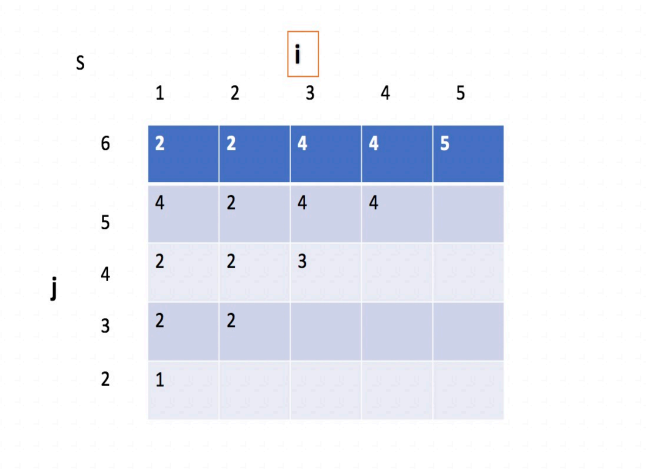
A1 = 5*10 A2=10*3 A3=3*12 A4=12*5 A5=5*50 A6=50*6



m

i

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



S

i

j	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				

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):
    c[i,j] = 0
    if Sij = ∅
        return c[i,j]
    else
        for k = i+1 to j-1
        if f[i]<= s[k] and f[k]<= s[j]
            q = RAS(s,f,i,k) + RAS(s,f,k,j) + 1
            if c[i,j] < q
                c[i,j] = q
    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 A_k be a max size subset of mutably compatible activities in S_k and let a_i be the activity in A_k with the last start time , if $a_i = a_m$ done , if if $a_i \neq a_m$,let set $A_{k'} = A_k - \{ a_i \} \cup \{a_m\}$ be A_k but substituting a_m for $a_{i,}$ $A_{k'}$ are disjoint, which follow because the activities in A_k are disjoint, a_i is the last activity in A_k to start and $s_m \geq s_i$, since $| A_{k'} |=| A_k |$, we conclude $A_{k'}$ is the maxmium subset of mutually compatible activities of s_k and include a_m

(2) proof the resulting solution has optimal substructure

Assume S_{ij} is the set of activities that start after activity a_i finished and that finish before activity a_j starts. Assume A_{ij} is the maximum set of mutually compatible activities in S_{ij}, and A_{ij} includes activity a_k, so we can divide the S_{ij} into two subproblems, S_{ik} ,and S_{kj}. In order to prove the resulting solution has optimal substructure ,we have to prove optimal solution Aij include optimal solutions to S_{ik} and S_{kj}.

Let A Let A_{ik} = A_{ij} intersect S_{ik} , A_{kj} = A_{ij} intersect S_{kj}, then, A_{ij} = A_{ik} union {a_k} union A_{kj}. Thus, $|A_{ij}| = |A_{ik}|+|A_{kj}| + 1$. If we could find a subset $A_{ik'}$ of mutually compatible activities in S_{ik} where $|A_{ik'}|>|A_{ik}|$, then $A_{ik'}$ is part of solution for S_{ij}. Then we have $|A_{ik'}| + |A_{kj}| + 1 > |A_{ik}| + |A_{kj}| + 1 = |A_{ij}|$. However, A_{ij} is an optimal solution. $A_{ik'}$ cannot exist. Thus, A_{ij} include the optimal solutions A_{ik} and A_{kj} to the two subproblems for S_{ik} and S_{kj}

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 t_{ijk}, 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,l,f)

```
$f_1[1] = e_1+a_{1,1}$
$f_2[1] = e_2+a_{2,1}$
$f_3[1] = e_3+a_{3,1}$
for i = 2 to n
    k =1
    if $f_1[i-1] +a_{1,i} \leq f_2[i-1]+t_{i-1,2,1}+a_{1,i}$
        $f_1[i-1] +a_{1,i} \leq f_3[i-1]+ t_{i-1,3,1}+a_{1,i}$
            then $f_1[i]= f_1[i-1] +a_{1,j}$
            $l_1[i]=1$
        elseif $f_2[i-1]+t_{i-1,2,1}+a_{1,i} \leq f_3[i-1]+ t_{i-1,3,1}+a_{1,i}$
            then $f_1[i]= f_2[i-1]+t_{i-1,2,1}+a_{1,i}$
            $l_1[i]=2$
        elseif  $f_1[i]= f_3[i-1]+ t_{i-1,3,1}+a_{1,i}$
            $l_1[i]=3$

    k=2
    if $f_2[i-1] +a_{2,i} \leq f_1[i-1]+t_{i-1,1,2}+a_{2,i}$
        $f_2[i-1] +a_{2,i} \leq f_3[i-1]+ t_{i-1,3,2}+a_{2,i}$
            then $f_2[i]= f_2[i-1] +a_{2,i}$
            $l_2[i]=2$
        elseif $f_3[i-1]+t_{i-1,3,2}+a_{2,i} \leq f_1[i-1]+ t_{i-1,1,2}+a_{2,i}$
            then $f_2[i]= f_3[i-1]+t_{i-1,3,2}+a_{2,i}$
            $l_2[i]=3$
        elseif  $f_1[i]= f_1[i-1]+ t_{i-1,1,2}+a_{2,i}$
            $l_2[i]=1$
    k=3
    if $f_3[i-1] +a_{3,i} \leq f_1[i-1]+t_{i-1,1,3}+a_{3,i}$
        $f_3[i-1] +a_{3,i} \leq f_2[i-1]+ t_{i-1,2,3}+a_{3,i}$
            then $f_3[i]= f_3[i-1] +a_{3,i}$
            $l_3[i]=3$
        elseif $f_1[i-1]+t_{i-1,1,3}+a_{3,i} \leq f_2[i-1]+ t_{i-1,2,3}+a_{3,i}$
            then $f_3[i]= f_1[i-1]+t_{i-1,1,3}+a_{3,i}$
            $l_3[i]=1$
        elseif  $f_3[i]= f_2[i-1]+ t_{i-1,2,3}+a_{3,i}$
            $l_1[i]=2$
```

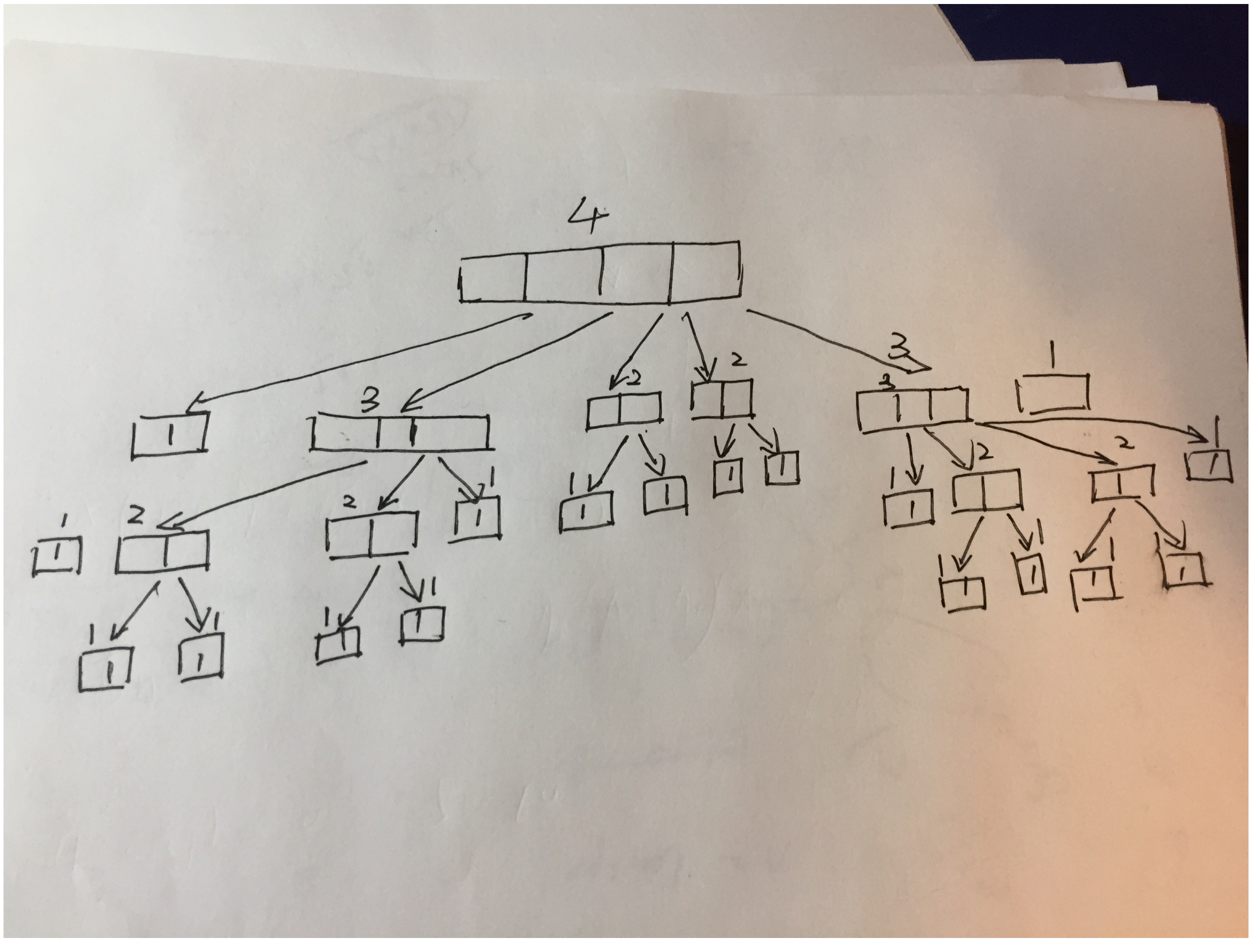
8. Write a memorized recursive algorithm RECURSIVE-MEMOIZED-LCS-LENGTH(X,Y) to compute the length of the LCS of X and Y based on equations (15.9), p. 393.

```
LCS-MEMOIZED-LENGTH(X,Y, i, j)
    m = X.length
    n  = Y.length
    let c[0..m,0…n]be new table
    for i = 0 to m
        for j = 0 to n
            c[i,j] = -∞
    RECURSIVE-LCS- LOOKUP(x,y,m,n,c)
    return c
```

```
RECURSIVE-LCS -LOOKUP(X,Y,i,j,c)
    if c[i,j] > 0
        return c[i,j]
    if i =0 or j = 0
        then c[i,j] = 0
    elseif   $X_i$=$Y_i$
        c[i,j]= RLL(X,Y,i-1, j -1,c) +1
    elseif    $X_i$ != $Y_i$
        c[I,j]=max(RLL(X,Y,i-1,j,c),RLL(X,Y,i,j-1,c))
    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, $p_n$)
```



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)
    Let r[1….n]be a new array
    for i = 1 to n
        r[i] = $-\infty$
    return   CUT-ROD-MEMOIZED-Aux(p,n,r)
```

```
CUT-ROD-MEMOIZED-Aux(p,n,r)
    If r[n] $\geq$ 0
        return r[n]
    if n==1
        q = p
    else q = $-\infty$
        for i = 1 to n
            q = max(q, CUT-ROD-MEMOIZED-Aux(p,i,r)+ CUT-ROD-MEMOIZED-Aux(p,n-i,r) )
    r[n] = q
    return q
```

this is less efficient than the Memoized-Cut –Rod

Bottom – up algorithm

```
Bottom – Up- Cut-Rod(p,n)
    Let r [0…n]be a new array
    r[0]=0
    for j =1 to n
        q = $-\infty$
        for i=1 to j-1
            q = max(q, r[i]+r[j-i])
        q=max($p_j$,q)
```

```
            r[j]=q
    return r[n]
```

this is same efficient as the usual one ,both of which complexity is $\Theta(n^2)$
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]