Problem Set 5

Due date: Electronic submission of the pdf file of this homework is due on **2/24/2023 before 11:59pm** on canvas.

Name: Ayushri Jain

Resources.

https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/

https://www.techiedelight.com/longest-increasing-subsequence-using-dynamic-programming/

Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, 3rd edition, The MIT Press, 2009 (or 4th edition)

https://en.wikipedia.org/wiki/Longest_increasing_subsequence

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework.

Signature: 

Problem 1 (20 points)**.**

Solution. We will use dynamic programming to find an optimal parenthesization of a matrix-chain product with the given sequence of dimensions $\langle 5, 10, 3, 12, 5, 50, 6\rangle$. To do that, we will first define subproblems, let $M[i, j]$ denote minimum number of scalar multiplications required to calculate product of matrices A_i to A_j. We will initialize all $M[i][j]$ with ∞. Our goal is to find value of $M[1, n]$, n is number of matrices which is 6 in this case, so we want to find $M[1][6]$. We know that when $i = j$, we have a single matrix which requires 0 scalar multiplications. So,

$$M[1][1] = 0, M[2][2] = 0, M[3][3] = 0, M[4][4] = 0, M[5][5] = 0, M[6][6] = 0$$

If two matrices $A(p \times q)$ and $B(q \times r)$ are multiplied, cost of multiplication $= p * q * r$. For i, j where $i < j$, we should consider all possible ways to split the matrix chain between A_i and A_j. Let that optimal split happen at k, then our recurrence relation is

$$M[i, j] = min(M[i][j], M[i, k] + M[k + 1, j] + d_{i-1} * d_k * d_j)$$

where $A_i, A_j, A_k = \langle d_{i-1}, d_k, d_j\rangle$. We will keep track of this k value in another array such that $S[i][j]$ will denote the optimal split point k for A_i and A_j.
Now we will fill values in M diagonally.
For $i = 1$ and $j = 2$

$$k = 1 : M[1][2] = min(M[1][2], M[1][1]+M[2][2]+5*10*3) = min(\infty, 150) = 150$$

$$M[1][2] = 150, S[1][2] = 1$$

For $i = 2$ and $j = 3$

$$k = 2 : M[2][3] = min(M[2][3], M[2][2]+M[3][3]+10*3*12) = min(\infty, 360) = 360$$

$$M[2][3] = 360, S[2][3] = 2$$

For $i = 3$ and $j = 4$

$$k = 3 : M[3][4] = min(M[3][4], M[3][3]+M[4][4]+3*12*5) = min(\infty, 180) = 180$$

$$M[3][4] = 180, S[3][4] = 3$$

For $i = 4$ and $j = 5$

$$k = 4 : M[4][5] = min(M[4][5], M[4][4]+M[5][5]+12*5*50) = min(\infty, 3000) = 3000$$

$$M[4][5] = 3000, S[4][5] = 4$$

For $i = 5$ and $j = 6$

$$k = 5 : M[5][6] = min(M[5][6], M[5][5]+M[6][6]+5*50*6) = min(\infty, 1500) = 1500$$

$$M[5][6] = 1500, S[5][6] = 5$$

For $i = 1$ and $j = 3$

$k = 1 : M[1][3] = min(M[1][3], M[1][1]+M[2][3]+5*10*12) = min(\infty, 960) = 960$

$$M[1][3] = 960, S[1][3] = 1$$

$k = 2 : M[1][3] = min(M[1][3], M[1][2]+M[3][3]+5*3*12) = min(960, 330) = 330$

$$M[1][3] = 330, S[1][3] = 2$$

For $i = 2$ and $j = 4$

$k = 2 : M[2][4] = min(M[2][4], M[2][2]+M[3][4]+10*3*5) = min(\infty, 330) = 330$

$$M[2][4] = 330, S[2][4] = 2$$

$k = 3 : M[2][4] = min(M[2][4], M[2][3]+M[4][4]+10*12*5) = min(330, 960) = 330$

For $i = 3$ and $j = 5$

$k = 3 : M[3][5] = min(M[3][5], M[3][3]+M[4][5]+3*12*50) = min(\infty, 4800) = 4800$

$$M[3][5] = 4800, S[3][5] = 3$$

$k = 4 : M[3][5] = min(M[3][5], M[3][4]+M[5][5]+3*5*50) = min(4800, 930) = 930$

$$M[3][5] = 930, S[3][5] = 4$$

For $i = 4$ and $j = 6$

$k = 4 : M[4][6] = min(M[4][6], M[4][4]+M[5][6]+12*5*6) = min(\infty, 1860) = 1860$

$$M[4][6] = 1860, S[4][6] = 4$$

$k = 5 : M[4][6] = min(M[4][6], M[4][5]+M[6][6]+12*50*6) = min(1860, 6600) = 1860$

For $i = 1$ and $j = 4$

$k = 1 : M[1][4] = min(M[1][4], M[1][1]+M[2][4]+5*10*5) = min(\infty, 580) = 580$

$$M[1][4] = 580, S[1][4] = 1$$

$k = 2 : M[1][4] = min(M[1][4], M[1][2]+M[3][4]+5*3*5) = min(580, 405) = 405$

$$M[1][4] = 405, S[1][4] = 2$$

$k = 3 : M[1][4] = min(M[1][4], M[1][3]+M[4][4]+5*12*5) = min(405, 630) = 405$

For $i = 2$ and $j = 5$

$k = 2 : M[2][5] = min(M[2][5], M[2][2]+M[3][5]+10*3*50) = min(\infty, 2430) = 2430$

$$M[2][5] = 2430, S[2][5] = 2$$

$k = 3 : M[2][5] = min(M[2][5], M[2][3]+M[4][5]+10*12*50) = min(2430, 9360) = 2430$

$k = 4 : M[2][5] = min(M[2][5], M[2][4]+M[5][5]+10*5*50) = min(2430, 2830) = 2430$

For $i = 3$ and $j = 6$

$k = 3 : M[3][6] = min(M[3][6], M[3][3]+M[4][6]+3*12*6) = min(\infty, 2076) = 2076$

$$M[3][6] = 2076, S[3][6] = 3$$

$k = 4 : M[3][6] = min(M[3][6], M[3][4]+M[5][6]+3*5*6) = min(2076, 1770) = 1770$

$$M[3][6] = 1770, S[3][6] = 4$$

$k = 5 : M[3][6] = min(M[3][6], M[3][5]+M[6][6]+3*50*6) = min(1770, 1830) = 1770$

For $i = 1$ and $j = 5$

$k = 1 : M[1][5] = min(M[1][5], M[1][1]+M[2][5]+5*10*50) = min(\infty, 4930) = 4930$

$$M[1][5] = 4930, S[1][5] = 1$$

$k = 2 : M[1][5] = min(M[1][5], M[1][2]+M[3][5]+5*3*50) = min(4930, 1830) = 1830$

$$M[1][5] = 1830, S[1][5] = 2$$

$k = 3 : M[1][5] = min(M[1][5], M[1][3]+M[4][5]+5*12*50) = min(1830, 6330) = 1830$

$k = 4 : M[1][5] = min(M[1][5], M[1][4]+M[5][5]+5*5*50) = min(1830, 1655) = 1655$

$$M[1][5] = 1655, S[1][5] = 4$$

For $i = 2$ and $j = 6$

$k = 2 : M[2][6] = min(M[2][6], M[2][2]+M[3][6]+10*3*6) = min(\infty, 1950) = 1950$

$$M[2][6] = 1950, S[2][6] = 2$$

$k = 3 : M[2][6] = min(M[2][6], M[2][3]+M[4][6]+10*12*6) = min(1950, 2940) = 1950$

$k = 4 : M[2][6] = min(M[2][6], M[2][4]+M[5][6]+10*5*6) = min(1950, 2130) = 1950$

$k = 5 : M[2][6] = min(M[2][6], M[2][5]+M[6][6]+10*50*6) = min(1950, 5430) = 1950$

For $i = 1$ and $j = 6$

$k = 1 : M[1][6] = min(M[1][6], M[1][1]+M[2][6]+5*10*6) = min(\infty, 2250) = 2250$

$$M[1][6] = 2250, S[1][6] = 1$$

$k = 2 : M[1][6] = min(M[1][6], M[1][2]+M[3][6]+5*3*6) = min(2250, 2010) = 2010$

$$M[1][6] = 2010, S[1][6] = 2$$

$k = 3 : M[1][6] = min(M[1][6], M[1][3]+M[4][6]+5*12*6) = min(2010, 2550) = 2010$

$k = 4 : M[1][6] = min(M[1][6], M[1][4]+M[5][6]+5*5*6) = min(2010, 2055) = 2010$

$k = 5 : M[1][6] = min(M[1][6], M[1][5]+M[6][6]+5*50*6) = min(2010, 3155) = 2010$

Finally, M is :

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

Minimum number of multiplications is 2010.
S is :

```
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
```

From S, we can trace find the actual paranthesization by recursively putting brackets around subexpression from i to $bracket[i][j]$ and then recursively put brackets around subexpression from $bracket[i][j] + 1$ to j and when $i = j$, we will print matrix name. Here is what happens -

```
function call : What happens?
print(1,6) : prints (
           : calls print(1,s[1][6]) = print(1,2)
print(1,2) : prints (
           : calls print(1,s[1][2]) = print(1,1)
print(1,1) : prints A1
           : return to print(1,2)
back to print(1,2) : calls print(s[1][2]+1,2) = print(2,2)
print(2,2) : prints A2
           : return to print(1,2)
back again to print(1,2) : prints )
back to print(1,6) : calls print(s[1][6]+1,6) = print(3,6)
print(3,6) : prints (
           : calls print(3,s[3][6]) = print(3,4)
print(3,4) : prints (
           : calls print(3,s[3][4]) = print(3,3)
print(3,3) : print A3
           : return to print(3,4)
back to print(3,4) : calls print(s[3][4]+1,4) = print(4,4)
print(4,4) : print A4
           : return to print(3,4)
back again to print(3,4) : prints )
back to print(3,6) : calls print(s[3][6]+1,6) = print(5,6)
print(5,6) : prints (
           : calls print(5,s[5][6]) = print(5,5)
print(5,5) : print A5
```

```
          : return to print(5,6)
back to print(5,6) : calls print(s[5][6]+1,6) = print(6,6)
print(6,6) : print A6
          : return to print(5,6)
back again to print(5,6) : prints )
back again to print(3,6) : prints )
back again to print(1,6) : prints )
```

According to above, our optimal parenthesization is : $((A_1A_2)((A_3A_4)(A_5A_6)))$. In terms of dimensions, answer is :

$$((5 \times 10)(10 \times 3))(((3 \times 12)(12 \times 5))((5 \times 50)(50 \times 6)))$$

Problem 2 (20 points)**.**

Solution. We need to prove that there exist some positive constants c and n_0 such that $P(n) \geq c2^n$ for all $n \geq n_0$ to show that $P(n)$ is $\Omega(2^n)$. We can do this using mathematical induction. First we will see our base case. When $n = 1$, then $P(1) = 1$. Substituting in $c2^n = c2^1 = 2c$ for any positive constant c. So we can choose $c \leq 1$ and $n_0 = 1$ to satisfy $P(1) \geq c2^1$, e.g. c can be $1/2$ for $n = 1$.
As next step, we will assume that $P(k) \geq c2^k$ for all $1 \leq k \leq n-1$. Now, we need to show that $P(n) \geq c2^n$.
Given recurrence relation for $P(n)$:

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$$

Let's assume that $P(n) \geq c2^n$ for some $n = k$. Now we will substitute our assumption in recurrence relation (inductive hypothesis), i.e., we have $P(k) \geq c2^k$ for $1 \leq k \leq n-1$. Therefore, we can write: $P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$ as

$$P(n) \geq \sum_{k=1}^{n-1} c2^k \cdot c2^{n-k} = c^2 \sum_{k=1}^{n-1} 2^n = c^2(n-1)2^n$$

Finally, we should show that $c^2(n-1)2^n \geq c2^n$ for all $n \geq 2$

$$c^2(n-1)2^n \geq c2^n$$

Dividing both sides by 2^n (which is positive), we get:

$$c^2(n-1) \geq c$$

Dividing both sides by c (which is positive), we get:

$$c(n-1) \geq 1$$

This inequality holds for any $c > 0$ as long as $n \geq 2$ because $c(n-1)$ is an increasing function of n for any fixed $c > 0$, so the inequality is most likely

to hold for the smallest value of n that we consider. When $n = 2$, we have $c(2-1) = c \geq 1$, so the inequality holds for any $c \leq 1/(n-1) = 1$. Therefore, we can choose c to be any positive constant that is smaller than or equal to $1/(n-1)$, and the inequality $c^2(n-1)2^n \geq c2^n$ will still hold for all $n \geq 2$. As a conclusion, if we choose c to be any positive constant that is smaller than or equal to $1/(n-1)$, and choose $n_0 = 2$, then by induction, we have shown that $P(n) \geq c2^n$ for all $n \geq n_0$. Hence, $P(n) = \Omega(2^n)$.

Problem 3 (20 points)**.**

Solution. In the MATRIX-CHAIN-ORDER algorithm (given in CLRS book), when we are calculating a table entry $m[i][j]$, we use the values of two other table entries inside the innermost loop, i.e. $m[i][k]$ and $m[k+1, j]$. Therefore, the innermost loop uses 2 references and it runs from $k = i$ to $k = j - 1$. The loop outside it runs from $i = 1$ to $i = n - l + 1$ and the outermost loop runs from $l = 2$ to $l = n$. If we combine all this, we can count the total number of times (sum of $R(i, j)$) that we reference a different entry than the one we are computing.

$$\begin{aligned}
\sum_{l=2}^{n}\sum_{i=1}^{n-l+1}\sum_{k=i}^{i+l-2} 2 &= \sum_{l=2}^{n}\sum_{i=1}^{n-l+1} 2(l-1) : using_normal_summation[i+l-2-i+1 = l-1] \\
&= \sum_{l=2}^{n} 2(l-1)(n-l+1) : using_normal_summation[n-l+1-1+1 = n-l+1] \\
&= \sum_{l=1}^{n-1} 2l(n-l) : replace[l]with[l-1] \\
&= 2n\sum_{l=1}^{n-1} l - 2\sum_{l=1}^{n-1} l^2 : distributing_summation \\
&= n^2(n-1) - 2\cdot\frac{(n-1)n(2n-1)}{6} : applying_sum_of_n_terms_and_n^2terms_formula \\
&= n^3 - n^2 - \frac{2n^3 - 3n^2 + n}{3} \\
&= \frac{n^3 - n}{3}.
\end{aligned}$$

Hence, $\sum_{i=1}^{n}\sum_{j=i}^{n} R(i, j) = \frac{n^3-n}{3}$.

Problem 4 (20 points)**.**

Solution. We can use dynamic programming to find the longest monotonically increasing subsequence of a sequence of n numbers. Let sequence of n numbers be denoted by seq and let l be an array such that for an index i, $L[i]$ stores the longest increasing subsequence of the sublist $seq[0 \ldots i]$ that ends with $seq[i]$. Our base case is when $i = 0$, $L[0]$ denotes the longest increasing subsequence ending at $seq[0]$. Now, we will iterate through our sequence using i and for

each element at index j in sublist $seq[0...i]$, we will find the longest increasing subsequence that ends with $seq[j]$ and $seq[j]$ is less than the current element $seq[i]$. We will keep appending the favorable elements in $L[i]$. Finally, using another loop, we will find the index j of L where the longest monotonically increasing subsequence is present. So our answer will be $L[j]$. For simplicity, we will consider our lists start with index 0. Below is the algorithm -

```
Algorithm longest_monotonically_increasing_subsequence(seq):
    if there are no elements in sequence, we return empty list
    n = length(seq)
    declare L : L[i] stores longest increasing subsequence of sublist seq[0...i]
     that ends with seq[i]
    L[0] = list(seq[0])
    # iterate over sequence from second element
    for i from 1 to n:
        for j from 0 to i-1:
            if seq[j] < seq[i] and length(L[j]) > length(L[i]):
                L[i] = L[j]

        # add seq[i] in L[i]
        L[i].append(seq[i])

    # find the index of L which has max length
    j = 0
    for i in range(length(seq)):
        if length(L[j]) < length(L[i]):
            j = i
    return L[j]
```

Since there are two nested loops, time complexity is $O(n^2)$.

Problem 5 (20 points)**.**

Solution. We can use a greedy algorithm to find the smallest set of unit-length closed intervals that contains all of the given n points:

$$\{x_1, x_2, \ldots, x_n\}$$

First we can sort the points in increasing order : $O(n \log n)$. Let our answer set be called as intervals and we initialize it as empty. We can then iterate over our input sorted points using a loop variable i. If the last interval in intervals does not cover the point x_i, we will add a new interval $[x_i, x_i + 1]$ to intervals. Otherwise if the last interval in intervals already covers the point x_i, we do nothing. Finally, we return the set intervals. In general, we will maintain a set of intervals that cover all the points, and we will add a new interval to our set whenever a point is not covered by any of the existing intervals. We will use unit-length intervals to minimize the total length of the intervals required.

```
Algorithm : smallest_set_of_closed_intervals(points)
n = length(points)
intervals = {} # empty set
sort points
for i from 1 to n:
     if last interval do not contain points[i] :
        add {points[i], points[i]+1} to intervals
return intervals
```

To prove the correctness of this algorithm, we need to show that the set of intervals it produces actually covers all the points, and that it is the smallest such set. To begin with, we can show that the set of intervals produced by the algorithm covers all the points. This is true because we add a new interval whenever a point is not covered by any of the existing intervals, and each new interval has length 1, so it will cover the point and no other point in the sequence. Now, we want to show that the set of intervals produced by the algorithm is the smallest set that covers all the points. Suppose that there exists a smaller set of intervals that covers all the points. Then, there must be at least one point that is covered by some interval in the smaller set, but not covered by any interval in the set produced by our algorithm. Let x_i be the leftmost such point, and let $[a, b]$ be the interval in the smaller set that covers x_i. Since x_i is not covered by any interval in the set produced by our algorithm, the last interval in intervals must end before x_i, i.e., it must be of the form $[c, c + 1]$ where c $< x_i$. But then we could replace the interval $[a, b]$ with the shorter interval $[x_i, x_i + 1]$, which also covers x_i and all subsequent points. This contradicts the assumption that the smaller set of intervals was the smallest set that covers all the points. Therefore, the set of intervals produced by our algorithm is indeed the smallest set that covers all the points. The time complexity of the algorithm is dominated by the sorting step, which takes $O(n \log n)$ time. The rest of the algorithm takes $O(n)$ time, since we only need to iterate over the sorted points once and perform constant-time operations at each step. The space complexity is also $O(n)$, since we need to store the list of intervals, which has at most n elements.