**Problem Set 5**

**Due date:** Electronic submission of the pdf file of this homework is due on **2/23/2024 before 11:59pm** on canvas.

**Name: Jawahar Sai Nathani**

**Resources.** (All people, books, articles, web pages, etc. that have been consulted when producing your answers to this homework)

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:** Jawahar Sai Nathani

Make sure that you describe all solutions in your own words. Typesetting in LaTeX is required. Read chapters 14 and 15 in our textbook.

**Problem 1** (20 points)**.** Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $\langle 5, 10, 3, 12, 5, 50, 6\rangle$. Explain how you found the solution. [Hint: You might find it worthwhile to implement the dynamic programming algorithm, and print the relevant tables to aid in your explanations. However, you should make sure that you are able to solve it by hand as well.]

**Solution.** Let the matrices and their dimensions be $A_1 = 5 \times 10$, $A_2 = 10 \times 3$, $A_3 = 3 \times 12$, $A_4 = 12 \times 5$, $A_5 = 5 \times 50$, $A_6 = 50 \times 6$.

Table 1: Array for matrix-chain product

| | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|---|---|
| $A_1$ | 0 | 150 | 330 | 405 | 1655 | 2010 |
| $A_2$ | 0 | 0 | 360 | 330 | 2430 | 1950 |
| $A_3$ | 0 | 0 | 0 | 180 | 930 | 1770 |
| $A_4$ | 0 | 0 | 0 | 0 | 3000 | 1860 |
| $A_5$ | 0 | 0 | 0 | 0 | 0 | 1500 |
| $A_6$ | 0 | 0 | 0 | 0 | 0 | 0 |

The optimal parenthesization for matrix-chain product takes 2010 scalar multiplications and the order of multiplication is

$$(A_1A_2)((A_3A_4)(A_5A_6))$$

- In the above order we multiply all the large matrix dimensions 10, 12 and 50 only once.
- The base matrix multiplications take respective no of scalar multiplications to compute - $A_1A_2 = 150$, $A_3A_4 = 180$ and $A_5A_6 = 1500$.
- The multiplication $(A_1A_2)(A_3A_4)$ requires $5 \times 3 \times 5 = 75$ scalar multiplications and $(A_3A_4)(A_5A_6)$ requires $3 \times 5 \times 6 = 90$ scalar multiplications.
- $((A_1A_2)(A_3A_4))(A_5A_6)$ requires $75+5\times5\times6 = 225$ and $(A_1A_2)((A_3A_4)(A_5A_6))$ requires $90 + 5 \times 3 \times 6 = 180$. Hence second matrix multiplication order is optimal.
- Therefore, the total number of scalar multiplications to compute $(A_1A_2)((A_3A_4)(A_5A_6))$ are $150 + 180 + 1500 + 180 = 2010$.

**Problem 2** (20 points)**.** Use a proof by induction to show that the solution to the recurrence

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2. \end{cases}$$

is $\Omega(2^n)$.

**Why do we care?** This is a simple lower bound on the number of parenthesizations of a chain of $n$ matrices. It will remind you about the true meaning of $\Omega(2^n)$. The result serves as a reminder why a brute-force solution is not attractive.

**Solution.** Base Case :
Let n = 1
For the base case n = 1, P(1) = 1 and $1 \geq c\, 2^1$ for all $c \leq 1/2$. Hence $P(n) = \Omega(2^n)$, This satisfies the claim.
Inductive case :
Need to prove $P(t+1) \geq c'\ 2^{t+1}$ for $t > 1$

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

According to inductive hypothesis lets assume $P(k) \geq c\, 2^k$ and $P(t+1-k) \geq c\, 2^{t+1-k}$

$$\geq \sum_{k=1}^{t} c.2^k.c.2^{t+1-k}$$

$$= \sum_{k=1}^{t} c^2.2^{(k+t+1-k)}$$

$$= c^2.2^{t+1} \sum_{k=1}^{t} 1$$

$$= c^2.2^{t+1}\ t$$

As $t > 1$, it can be written as

$$\geq c^2.2^{t+1}$$

$$= c'\ 2^{t+1}$$

As $P(t+1) \geq c'\ 2^{t+1}$ we can say that $P(n) = \Omega(2^n)$ satisfies for all values of n according to proof by induction. Hence Proved.

**Problem 3** (20 points)**.** Let $R(i, j)$ be the number of times that table entry $m[i, j]$ is referenced while computing other table entries in a call of MATRIX-CHAIN-ORDER. Show that the total number of references for the entire table is

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

[Hint: Re-read the definition of $R(i, j)$ a couple of times.]

**Why do we care?** This is a key statistics for the run-time of the dynamic-programming solution. The manipulation of such sums is an essential skill that you need to train.

**Solution.** We can compute the summation of R(i, j) by calculating the number of times each matrix element is visited while calculating other matrix entries. According to the textbook, the summation can be written as

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j) &= \sum_{l=2}^{n} 2(n-l+1)(l-1) \\
&= 2\sum_{l=1}^{n-1} l(n-l) \\
&= 2\sum_{l=1}^{n-1} nl - 2\sum_{l=1}^{n-1} l^2 \\
&= 2n\frac{n(n-1)}{2} - 2\frac{n(n-1)(2n-1)}{6} \\
&= n^2(n-1) - \frac{n(n-1)(2n-1)}{3} \\
&= \frac{n(n-1)(3n-2n+1)}{3} \\
&= \frac{n(n-1)(n+1)}{3} \\
&= \frac{n^3-n}{3}
\end{aligned}$$

**Problem 4** (20 points)**.** Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

**Why do we care?** You should learn how to apply the algorithms from the lecture. This is a good opportunity to hone your problem solving skills. Make sure that you solve it yourself without any help!

**Solution.** Below algorithm

```
def LongIncSeq(arr) {
    n = len(arr)
    seqLen = [1]*n

    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j++)
            if arr[i] >= arr[j] and seqLen[i] < seqLen[j] + 1
                seqLen[i] = seqLen[j]+1


    longestSeqLen = max(seqLen)
    curLen = longestSeqLen
    longSubSeq = []

    for (int i = n-1; i > -1; i--)
        if seqLen[i] == curLen:
            curLen -= 1
            longSubSeq.append(arr[i])
    longSubSeq.reverse()
    return longSubSeq
}
```

**Problem 5** (20 points)**.** Describe an efficient algorithm that, given a set

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

of $n$ points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

**Why do we care?** This is nice opportunity to design a simple greedy algorithm. Arguing the correctness of a greedy algorithm is essential, and this is not too difficult to do in this case.

**Solution.** To determine the smallest set of unit-length closed intervals let's use a greedy algorithm.

- Since the initial set of points can be in any order, first sort the points into ascending order. Let the sorted set points be

$$Y = \{y_1, y_2, \ldots, y_n\}$$

- Iterate through the array Y, and for for the current indexed point add a unit-length interval $[y_i, y_i + 1]$. Suppose for $y_1$ we will add an interval $[y_1, y_1 + 1]$.
- Now skip all the points of Y which are in the interval $[y_i, y_i + 1]$, as all those points are covered in the interval.
- From the next uncovered point $y_j$, repeat the above 2 steps until all the points in Y are covered.
- To compute the above algorithm it takes $\Theta(n \log n)$. Array of points can be sorted in $\Theta(n \log n)$ and adding the unit-length intervals takes $\Theta(n)$ time complexity.

**Proof of correctness :**
Since we are sorting the array and adding unit-length intervals based on leftmost uncovered point, we ensure that all the points from the set are covered. As the array is sorted and we are adding the intervals optimally, it makes the whole algorithm optimal.

Discussions on canvas are always encouraged, especially to clarify concepts that were introduced in the lecture. However, discussions of homework problems on canvas should not contain spoilers. It is okay to ask for clarifications concerning homework questions if needed. Make sure that you write the solutions in your own words.

**Checklist:**
✓ Did you add your name?
✓ Did you disclose all resources that you have used?
(This includes all people, books, websites, etc. that you have consulted)
✓ Did you sign that you followed the Aggie honor code?
✓ Did you solve all problems?
✓ Did you submit the pdf file of your homework?