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: Huy Quang Lai

Resources. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. *Introduction to Algorithms*. The MIT Press.

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: _____

Make sure that you describe all solutions in your own words. Typesetting in LaTeX is required. Read chapters 15 and 16 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.

Solution. Let $A = 5 \times 10$ matrix, $B = 10 \times 3$ matrix, $C = 3 \times 12$ matrix, $D = 12 \times 5$ matrix, $E = 5 \times 50$ matrix, $F = 50 \times 6$ matrix
The most optimal parenthesization of the matrix chain is

$$(AB)\,((CD)(EF))$$

With this order, the three largest dimensions of 10, 12, and 50 are absorbed before they can make any other matrix multiplication take more operations. With this approach, we get the optimal 2010 operations.
AB is a 5×3 matrix that requires $5 \times 10 \times 3 = 150$ operations to compute.
CD is a 3×5 matrix that requires $3 \times 12 \times 5 = 180$ operations to compute.
EF is a 5×6 matrix that requires $5 \times 50 \times 6 = 1500$ operations to compute.
From here, there is a choice to group $(AB)(CD)$ or $(CD)(EF)$
$(AB)(CD)$ is a 5×5 matrix that requires an additional $5 \times 3 \times 5 = 75$ operations to compute.
$(CD)(EF)$ is a 3×6 matrix that requires an additional $3 \times 5 \times 6 = 90$ operations to compute.
After this grouping a final multiplication is needed.
$(AB)((CD)(EF))$ is a 5×6 matrix that requires an additional $5 \times 3 \times 6 = 90$ operations to compute.
$((AB)(CD))(EF)$ is a 5×6 matrix that requires an additional $5 \times 5 \times 6 = 150$ operations to compute.
Although more locally optimal to group $(AB)(CD)$, the grouping of $(CD)(EF)$ is more globally optimal.

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 $\Omega(2^n)$.

Solution. Base Case:
Let $n = 2$

$$\begin{aligned} P(2) &= \sum_{k=1}^{2-1} P(k)P(2-k) \\ &= P(1)P(1) \\ &= 1 \end{aligned}$$

$\because 1 \geq 2^{2-2} = 2^0 = 1$
$P(n) \geq 2^{n-2}$ holds for $n = 2$
Inductive Case:
Assume that $P(n) \geq 2^{n-2}$

$$\begin{aligned} P(n) &= \sum_{k=1}^{n-1} P(k)P(n-k) \\ &= P(1)P(n-1) + \sum_{k=2}^{n-2} P(k)P(n-k) + P(n-1)P(1) \\ &\geq 2^{n-3} + \sum_{k=2}^{n-2} \left(2^{k-2}2^{n-k-2}\right) + 2^{n-3} \\ &= 2^{n-2} + \sum_{k=2}^{n-2} 2^{n-4} \\ &= 2^{n-2} + 2^{n-4} \sum_{k=2}^{n-2} 1 \\ &= 2^{n-2} + (n-3)\left(2^{n-4}\right) \\ &= (n+1)2^{n-4} \end{aligned}$$

$\therefore P(n) \in \Omega(2^n)$
□

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

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

Solution. From the textbook

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

□

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

Solution.

```
def longSeq(seq):
    n = len(seq)
    L = [1 for i in range(n)]
    for i in range(1, n):
        for j in range(0, i):
            if seq[j] < seq[i]:
                L[i] = max(L[j] + 1, L[i])

    max_length = max(L)
    current_length = max_length
    subseq = []

    for i in range(n - 1, -1, -1):
        if L[i] == current_length:
            current_length -= 1
            subseq.append(seq[i])
    subseq.reverse()
    return subseq
```

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.

Solution. First, sort the set of n points $\{x_1, x_2, \ldots, x_n\}$ to get the set $Y = \{y_1, y_2, \cdots, y_n\}$ such that $y_1 \leq y_2 \leq \cdots \leq y_n$.
Next, linearly scan over Y. For every $y_i \in Y$, place the closed interval $[y_i, y_i + 1]$ in the optimal solution set. Call this set S.
Remove all the points in Y that are covered by $[y_i, y_i + 1]$.
Repeat this process for every point y_i until Y is empty.
S should be the optimal solution.
Suppose that there exists an optimal solution S^* such that y_1 is covered by $[x', x' + 1] \in S^*$ where $x' < 1$.
Since y_1 is the left most element of the given set, there is no other point lying in $[x', y_1)$.
Therefore, replacing $[x', x' + 1]$ in S^* with $[y_1, y_1 + 1]$ will result in another optimal solution.
Since this idea can be applied to every point on Y, S is the optimal solution.
The runtime of the algorithm is $O(n \log n)$ due to the initial sort. The linear traversal of the set is $O(n)$. With this the total runtime of the algorithm is

$$O(n \log n)$$