1. A simple greedy algorithm that scans the array $A$ from left to right is optimal. Starting from $l_0 = 0$, find the rightmost point $l_1$ such that $\sum_{j=0}^{l_1} A_j \leq W$. Keep repeating this step and finding more $l_i$'s till the end of $A$ is reached.

We can prove that the above algorithm is optimal by contradiction. Let the partition found by the greedy algorithm be $(l_0, \ldots, l_k)$. Suppose there exists a different valid partition $(l'_0, \ldots, l'_k)$, where $k' < k$. Then there must exist an $i < k'$ such that $l'_{i+1} > l_{i+1}$. Pick the smallest such $i$. Then, since the numbers are nonnegative, it must be true that $\sum_{j=l'_{i+1}}^{l'+1} A[j] > W$, which violates the validity of the partition.

2. Define the cost function $C(m, l)$ as the minimum cost of partitioning the first $m$ elements into $l$ pieces, where the cost is defined as in the question. We thus have to find $C(n, k)$. Then the following recursive characterization is the crux of the dynamic programming solution:

$$C(m, l) = \begin{cases} 
\min_i \max_l \{C(i, l-1), \sum_{j=i+1}^{m} A[j]\} & \text{if } m > 1 \text{ and } l > 1, \\
\sum_{j=1}^{m} A[j] & \text{if } l = 1, \\
A[1] & \text{if } m = 1.
\end{cases}$$

The subarray sums can be calculated in $O(1)$ time if we precompute a $Sum$ array, where $Sum[i]$ stores the sum of the first $i$ elements. It is easy to see that $\sum_{j=l+1}^{m} A[j] = Sum[r] - Sum[l]$.

The cost function can be calculated in a bottom-up fashion, starting with $C(n, 1)$, for all $1 \leq m \leq n$. Calculating $C(m, l+1)$ just requires knowledge of all the $C(m, l)$'s, so we do not need more than $O(n)$ storage for the algorithm. Keeping track of the optimal partition along the way is not difficult. Finally note that the running time is $O(kn^2)$.

3. Let us look at Strassen’s algorithm, which offers an improvement in running time over the naive time of $O(n^3)$. The algorithm uses a divide and conquer strategy and calculates sub-matrix multiplications efficiently. For simplicity, let $n = 2^m$. If we partition $A$, $B$ and $C$ into equal sized block matrices,

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \ast \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

then it is clear that

\[
\begin{align*}
C_{11} &= A_{11} \ast B_{11} + A_{12} \ast B_{21} \\
C_{12} &= A_{11} \ast B_{12} + A_{12} \ast B_{22} \\
C_{21} &= A_{21} \ast B_{11} + A_{22} \ast B_{21} \\
C_{22} &= A_{21} \ast B_{12} + A_{22} \ast B_{22}.
\end{align*}
\]
However, if we define new matrices as follows

\[
\begin{align*}
M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \times B_{11} \\
M_3 &= A_{11} \times (B_{12} - B_{22}) \\
M_4 &= A_{22} \times (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \times B_{22} \\
M_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}),
\end{align*}
\]

then one can verify that

\[
\begin{align*}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_4 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6.
\end{align*}
\]

This has reduced the number of multiplications down to 7, and each multiplication can be recursively calculated using the same trick. We thus get the recurrence relation

\[T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)\]

which results in a running time of \(O(n^{\log_2 7}) = O(n^{2.807})\).

4. This is the well-known Knapsack problem [1]. We will give a short summary of what is given in the book. Let us define \(K(w, j)\) as the maximum value achievable using a weight limit \(w\) and the first \(j\) items in the list. We are interested in \(K(W, n)\). The following recursive definition is key:

\[
K(w, j) = \max\{K(w - w_j, j - 1) + v_j, K(w, j - 1)\}.
\]

In words, to calculate \(K(w, j)\), we just have to decide whether to take the element \(j\) or not. The above definition is not complete since one has to take care of boundary cases, but those are easy to see. The time and space required by the algorithm is both \(O(nW)\).

References