Problem 1*. Let $A$ be an array of $n$ integers. Define a function $f(x)$ — where $x \geq 0$ is an integer — as follows:

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ \max_{i=1}^{x} (A[i] + f(x - i)) & \text{otherwise} \end{cases}$$

Consider the following algorithm for calculating $f(x)$:

algorithm $f(x)$
1. if $x = 0$ then return 0
2. $max = -\infty$
3. for $i = 1$ to $x$
4. \hspace{1em} $v = A[i] + f(x - i)$
5. \hspace{1em} if $v > max$ then $max = v$
6. return $max$

Prove: the above algorithm takes $\Omega(2^n)$ time to calculate $f(n)$.

Solution. Let $g(x)$ denote the time of the algorithm in calculating $f(x)$. We know:

$$
g(0) \geq 1 \\
g(1) \geq 1 \\
g(n) \geq \sum_{i=0}^{n-1} g(i)
$$

We will show by induction that $g(n) \geq 2^{n-1}$ for $n \geq 1$. First, this is obviously correct when $n = 1$. Next, we will prove the claim on $n = k$ for any $k \geq 2$, assuming that it is correct for all $n \leq k - 1$.

$$
g(n) \geq \sum_{i=0}^{n-1} g(i) \\
\geq 1 + \sum_{i=1}^{n-1} g(i) \\
\geq 1 + \sum_{i=1}^{n-1} 2^{i-1} \\
\geq 2^{n-1}.
$$

Problem 2 (The Piggyback Technique). Recall that, for the rot cutting problem, we derived the function $opt(n)$ — the optimal revenue from cutting up a rod of length $n$ — as follows:

$$opt(0) = 0$$
\[ \text{opt}(n) = \min_{i=1}^{n} P[i] + \text{opt}(n - i) \]  

For \( n \geq 1 \), define \( \text{bestSub}(n) = k \) if the maximization in (1) is obtained at \( i = k \). Answer the following questions:

- Explain how to compute \( \text{bestSub}(t) \) for all \( t \in [1, n] \) in \( O(n^2) \) time.

- Assume that \( \text{bestSub}(t) \) has been computed for all \( t \in [1, n] \). Explain how to output an optimal way to cut the rod in \( O(n) \) time.

**Solution.** First, compute \( \text{opt}(t) \) for all \( t \in [1, n] \) in \( O(n^2) \) time. Then, for each \( t \in [1, n] \), spend \( O(t) \) time to find the value \( k \in [1, t] \) that maximizes \( P[k] + \text{opt}(t - k) \); after that, set \( \text{bestSub}(t) = k \). The total cost of doing so for all \( t \in [1, n] \) is \( \sum_{t=1}^{n} O(t) = O(n^2) \).

We can now produce an optimal way to cut the rod as follows:

1. \( \ell \leftarrow n \)
2. while \( \ell > 0 \) do
   3. output “produce a segment of length \( \text{bestSub}(\ell) \)”
   4. \( \ell \leftarrow \ell - \text{bestSub}(\ell) \)

It is easy to see that the running time is \( O(n) \).

**Problem 3*. Let \( A \) be an array of \( n \) integers. Define function \( f(a, b) \) — where \( a \in [1, n] \) and \( b \in [1, n] \) — as follows:

\[
f(a, b) = \begin{cases} 
0 & \text{if } a \geq b \\
(\sum_{c=a}^{b} A[c]) + \min_{c=a+1}^{b-1} \{f(a, c) + f(c, b)\} & \text{otherwise}
\end{cases}
\]

Design an algorithm to calculate \( f(1, n) \) in \( O(n^3) \) time.

**Solution.** We will launch \( n \) rounds. In the \( i \)-th round \((i \in [1, n - 1])\), we calculate all the \( f(a, b) \) satisfying \( 1 \leq a \leq b \leq n \) and \( b = a + i \). The strategy ensures that when \( f(a, b) \) is computed, \( f(a, c) \) and \( f(c, b) \) are ready for all \( c \in [a, b] \). Hence, the computation of \( f(a, b) \) takes \( O(n) \) time. The total running time is \( O(n^3) \) because there are \( O(n^2) \) values to compute.

**Problem 4.** In Lecture Notes 8, our algorithm for computing \( f(n, m) \) has space complexity \( O(nm) \), i.e., it uses \( O(nm) \) memory cells. Reduce the space complexity to \( O(n + m) \).

**Solution.** The lecture notes mentioned that we can list the subproblems in the “row-major” order. Specifically, row \( i \in [0, n] \) contains all the subproblems \( f(i, 0), f(i, 1), \ldots, f(i, m) \); and we process the rows in ascending order of \( i \). Storing all the rows consumes \( O(nm) \) space. Noticing that only row \( i = 1 \) is needed to compute row \( i \geq 1 \). Therefore, at any moment, it suffices to store only two rows, which requires only \( O(m) \) cells.

Remark: the space consumption is \( O(n + m) \) (not \( O(m) \)) because you still need to store the input strings \( x \) and \( y \).

**Problem 5*. Let \( G = (V, E) \) be a directed acyclic graph (DAG). For each vertex \( u \in V \), let \( \text{IN}(u) \) be the set of in-neighbors of \( u \) (recall that a vertex \( v \) is an in-neighbor of \( u \) if \( E \) has an edge from \( v \) to \( u \)). Define function \( f : V \rightarrow \mathbb{N} \) as follows:

\[
f(u) = \begin{cases} 
0 & \text{if } \text{IN}(u) = \emptyset \\
1 + \min_{v \in \text{IN}(u)} f(v) & \text{otherwise}
\end{cases}
\]
Design an algorithm to calculate $f(u)$ of every $u \in V$. Your algorithm should run in $O(|V| + |E|)$ time. You can assume that the vertices in $V$ are represented as integers $1, 2, ..., |V|$.

**Solution.** Compute a topological order of $G$ in $O(|V| + |E|)$ time. Then, compute the $f(u)$ values of all vertices $u \in V$ according to the vertex ordering in the topological order.

Remark: Recall that a topological order of $G$ is an ordering of the vertices in $V$ where each vertex $u \in V$ is positioned after every vertex $v \in \text{IN}(u)$. A topological order can be obtained using depth first search in $O(|V| + |E|)$ time, which was discussed in CSCI2100. See Prof. Tao’s homepage (http://www.cse.cuhk.edu.hk/~taoyf/) for the course homepage of CSCI2100.

**Problem 6**. Let $G = (V, E)$ be a directed acyclic graph (DAG). Design an algorithm to find the length of the longest path in $G$ (recall that the length of a path is the number of edges in the path). Your algorithm should run in $O(|V| + |E|)$ time. You can assume that the vertices in $V$ are represented as integers $1, 2, ..., |V|$.

**Solution.** Define function $f : V \to \mathbb{N}$ as follows:

$$f(u) = \begin{cases} 
0 & \text{if } \text{IN}(u) = \emptyset \\
1 + \max_{v \in \text{IN}(u)} f(v) & \text{otherwise} 
\end{cases}$$

The length of the longest path equals $\max_{u \in V} f(u)$. Similar to Problem 5, we can compute $f(u)$ for all $u \in V$ in $O(|V| + |E|)$ time.