Problem 1. Let A be an array of n integers. Define

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

Consider the algorithm for computing $f(x)$:

```
procedure F(x)
    if x = 0 then
        return 0
    end if
    max ← −∞
    for i ← 1 to x do
        v ← A[i] + F(x − i)
        if v > max then
            max ← v
        end if
    end for
    return max
end procedure
```

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

Proof.

$$f(0) = 1, f(1) = 1, f(n) \geq \sum_{i=1}^{n} f(n-i) = \sum_{i=0}^{n-1} f(i).$$

Proof by induction. Let $P(k)$ be the predicate "$f(k) \geq 2^{n-1}$."

Base Case. The statement is true for $n = 1$, as $f(1) = 1 \geq 2^{1-1} = 1$.

Inductive Step. Suppose the statement is true for any $P(n)$ with $n < k$. Then, for $k = n + 1$,

$$f(k) \geq \sum_{i=0}^{n-1} f(i) \geq 1 + \sum_{i=1}^{n-1} 2^{i-1} = 1 + \frac{1 \times (1 - 2^{n-1})}{1 - 2} = 2^{n-1}.$$

Hence, the statement holds true for all $n \geq 1$.

For any $n \geq n_0 = 1$, there exists $k = 1/2 > 0$, such that $f(n) \geq k \cdot 2^n$. Hence, the algorithm takes $\Omega(2^n)$ time to compute $f(n)$. □

Problem 2. Reconsider the rot cutting problem where we cut the rod into segments of integer lengths corresponding to different prices. The optimal revenue from cutting up a rod of length n can be derived using the function $opt(n)$, where

$$opt(n) = \begin{cases} 0, & n = 0, \\ max_{i=1}^{n} P[i] + opt(n-i), & \text{otherwise.} \end{cases}$$

For $n \geq 1$, define $bestSub(n) = k$ if the maximization is obtained at $i = k$. Describe how to compute $bestSub(t)$ for all $t \in [1, n]$ in $O(n^2)$ time, and how to output an optimal way to cut the rod in $O(n)$ time after computing $bestSub(t)$.

Solution. First, the subproblems are $opt(0), opt(1), \dots, opt(n)$. With the essence of dynamic programming, we resolve subproblems with the order $opt(0), opt(1), opt(2), \dots, opt(n)$. Clearly, computing $opt(i)$ given $opt(0), opt(1), \dots, opt(i-1)$ requires $O(i)$ time only. Hence, we can compute $opt(t)$ for all $t \in [1, n]$ in $O(n^2)$ time.

Then, for each $t \in [1, n]$, we spend $O(i)$ time to find $k \in [1, n]$ that maximizes $P[k] + opt(n-k)$, which is $bestSub(t)$. Therefore, $bestSub(t)$ for all $t \in [1, n]$ can also be computed in $O(n^2)$ time.

The piggyback technique can be used to compute an optimal solution to the rot cutting problem: Since $bestSub(n)$ indicates the best way for cutting up a rod of length n by first obtaining a segment with length $bestSub(n)$, we only need to consider the optimal way to cut a rod with length $n - bestSub(n)$ now.

```
procedure OptimalCut(n)
    if n > 0 then
        output "produce a segment of length bestSub(n)"
        OptimalCut(n − bestSub(n))
    end if
end procedure
```

Problem 3. Let A be an array of n integers. Define

$$f(a, b) = \begin{cases} 0, & a \geq b, \\ \left(\sum_{i=a}^{b} A[i]\right) + \min_{i=a+1}^{b-1}\{f(a, i) + f(i, b)\}, & \text{otherwise.} \end{cases}$$

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

Solution. Each $f(a, b)$ can only depend on $f(a, i)$ and $f(i, b)$ with $a < i < b$. We order the subproblems such that all the $f(a, b)$ satisfying $1 \leq a \leq b \leq n$ and $b = a + i$ is ahead of all the $f(a, b)$ satisfying $1 \leq a \leq b \leq n$ and $b = a + i + 1$. We can also order all the $f(a, a + i)$ such that smaller value of a's are calculated first.

Since we have obtained all $f(a, i)$ and $f(i, b)$ for computing $f(a, b)$, we only need to spend an extra $O(b - a)$ time to compute $f(a, b)$.

The time complexity of this computation strategy becomes

$$\sum_{k=0}^{n-1} \sum_{i=1}^{n-k} O(i + k - i) = \sum_{k=0}^{n-1} (n - k) O(k) = \sum_{k=1}^{n} O(n^2) = O(n^3).$$

Problem 4. **(Rolling Array)** Reduce the space complexity for computing the length of the Longest Common Subsequence for two strings x and y with length n and m, from space complexity of $O(nm)$ to $O(n+m)$.

Solution. Recall that the recursive formula of computing the length of LCS is

$$f(a,b) = \begin{cases} 0, & \text{if } a = 0 \text{ or } b = 0, \\ 1 + f(a-1, b-1), & \text{if } a, b > 0 \text{ and } x[a] = y[b], \\ \max\{f(a, b-1), f(a-1, b)\}, & \text{otherwise.} \end{cases}$$

The computation for subproblems can be arranged in "row-major" order. Specifically, row $i \in [0, n]$ contains all the subproblems $f(i,0), f(i,1), \dots f(i,m)$, while processing the rows in ascending order of i. Noticing that only row $i-1$ is needed to compute row $i \geq 1$. Therefore, it suffices to store only two rows (a temporary storage array and a result array), which requires only $O(m)$ cells. After computing row i, we can move it to the temporary storage array and use it to compute row $i+1$ on the result array.

Note that the storage complexity is $O(n+m)$ instead of $O(m)$ since we have to store the strings as well.

Problem 5. **(Shortest Path in a DAG)** Let $G = (V, E)$ be a directed acyclic graph (DAG). For each vertex $u \in V$, let $\mathrm{IN}(u)$ be the set of in-neighbours of u (a vertex v is an in-neighbour of u if E has an edge from v to u.) Define $f : V \mapsto \mathbb{N}$, where

$$f(u) = \begin{cases} 0, & \mathrm{IN}(u) = \emptyset, \\ 1 + \min_{v \in \mathrm{IN}(u)} f(v), & \text{otherwise.} \end{cases}$$

Design an algorithm to compute $f(u)$ for every $u \in V$ in $O(|V| + |E|)$ time. Vertices in V are assumed to be represented using integers $1, 2, \dots, |V|$.

Solution. Compute a topological order of G in $O(|V| + |E|)$ time. Then, compute $f(u)$ of every $u \in V$, following the topological order. Since in a topological order, every vertex $u \in V$ is positioned after every vertex $v \in \mathrm{IN}(u)$, we can ensure that every $f(v)$ has been computed before compuing $f(u)$. Therefore, all $f(u)$ can be computed in $O(|V| + |E|)$ time.

Problem 6. **(Longest Path in a DAG)** Let $G = (V, E)$ be a DAG. Design an algorithm to find the length of the longest path in G in $O(|V| + |E|)$ time. Recall that the length of a path is the number of edges in the path. You can assume that the vertices in V are represented as integers $1, 2, \ldots, |V|$.

Solution. **Lemma.** A path with the maximum length must have a vertex with in-degree (number of edges coming into the vertex) zero as its starting point.

Proof. Let an optimal path be $v_1 - v_2 - \ldots - v_k$, and the in-degree of v_1 is non-zero. Hence, $\mathrm{IN}(v_1) \neq \emptyset$. Let $u \in \mathrm{IN}(v_1)$. Then we are able to construct a longer path in G by considering $u - v_1 - v_2 - \ldots - v_n$, which is a contradiction.

Define $f(u)$ as the length of the longest path in G starting from any point with in-degree zero and ends with u. In addition, each u must come from an in-neighbour of u. We have

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

We can calculate $f(u)$ in $O(|V| + |E|)$ time.

The longest path must end with a point $u \in V$ and thus we can obtain the longest path with an extra $O(|V|)$ time.

Problem 7. Define

$$f(n) = \begin{cases} 0, & n = 0, \\ 1, & n = 1, \\ f(n-1) + f(n-2), & \text{otherwise} \end{cases},$$

where n is a non-negative integer. Give an algorithm to calculate $f(n)$ in $O(n)$ time, with the assumption that $f(n)$ fits in a word.

Solution. The subproblems are $f(0), f(1), f(2), \ldots, f(n)$. We adapt the idea of dynamic programming and arrange the subproblems with the computation order $f(0), f(1), \ldots, f(n)$. Thus, when we are required to compute $f(n)$ where $n \geq 2$, the underlying subproblems $f(n-1)$ and $f(n-2)$ would have been ready and stored in an array of size n.

Therefore, the complexity of computing $f(n)$ is $\sum_{i=1}^{n} O(1) = O(n)$.

Problem 8. Let A be an array of n integers. Consider the following recursive function

$$f(i,j) = \begin{cases} 0, & i = j \\ A[i] \cdot A[j] + \min_{k=i+1}^{j-1} f(i,k) + f(k,j), & i \neq j. \end{cases},$$

where $1 \leq i \leq j \leq n$.
Design an algorithm to compute $f(1,n)$ in $O(n^3)$ time.

Solution. Each $f(a,b)$ can only depend on $f(a,i)$ and $f(i,b)$ with $a < i < b$. We order the subproblems such that all the $f(a,b)$ satisfying $1 \leq a \leq b \leq n$ and $b = a + i$ is ahead of all the $f(a,b)$ satisfying $1 \leq a \leq b \leq n$ and $b = a + i + 1$. We can also order all the $f(a, a+i)$ such that smaller value of a's are calculated first.

Since we have obtained all $f(a,i)$ and $f(i,b)$ for computing $f(a,b)$, we only need to spend an extra $O(b-a)$ time to compute $f(a,b)$.

The time complexity of this computation strategy becomes

$$\sum_{k=0}^{n-1} \sum_{i=1}^{n-k} O(i + k - i) = \sum_{k=0}^{n-1} (n-k) O(k) = \sum_{k=1}^{n} O(n^2) = O(n^3).$$

To be clearer, the algorithm below shows the aforesaid method of calculating $f(1,n)$.

```
Set F[i,i] = 0 for all i = 1,2,...,n
for k ← 1 to n do
    for l ← 1 to n do
        r ← l + k, F[l,r] ← +∞
        for i ← l + 1 to r − 1 do
            F[l,r] ← min{F[l,r], A[l] · A[r] + F[l,i] + F[i,r]}
        end for
    end for
end for
```

Problem 9. Establish a recursive function $f(i,j)$ to compute the Longest Common Subsequence for two substrings $x[1:i], y[1:j]$. Compute all $f(i,j)$'s for $x = \texttt{ABC}$ and $y = \texttt{BDCA}$.

Solution. Recall that

$$f(a,b) = \begin{cases} 0, & \text{if } a = 0 \text{ or } b = 0, \\ 1 + f(a-1, b-1), & \text{if } a, b > 0 \text{ and } x[a] = y[b], \\ \max\{f(a, b-1), f(a-1, b)\}, & \text{otherwise.} \end{cases}$$

$f(i,j)$	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	1
2	0	1	1	1	1
3	0	1	1	2	2

$best(i,j)$	0	1	2	3	4
0	-	-	-	-	-
1	-	(1,0)	(1,1)	(1,2)	(0,3)
2	-	(1,0)	(2,1)	(2,2)	(2,3)
3	-	(2,1)	(3,1)	(2,2)	(3,3)

The length of LCS(x,y) is 2, while the LCS is `BC`.

Problem 10. Consider the rot cutting problem again. Suppose $n = 5$ and the price array P is $[2, 6, 7, 9, 10]$. What is the maximum revenue achievable? What is the optimal way of rot cutting?

Solution. We first compute the table of $opt(t)$ and $bestSub(t)$ for $t \in [1, n]$.

t	1	2	3	4	5
$opt(t)$	2	6	8	12	14

t	1	2	3	4	5
$bestSub(t)$	1	2	1	2	1

Therefore, the maximum revenue is 14 by cutting the rod into segments of $1, 2, 2$.

Problem 11. Consider a modification of the rod-cutting problem in which, in addition to a price $P[i]$ for each length $i \in [1, n]$, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the segments minus the total cost of making the cuts. Give a dynamic-programming algorithm to solve this modified problem in $O(n^2)$ time.

Solution. In the modified recursive formula, we take note that we can either keep the rod uncut, or spend the cost c for obtaining a segment of length i.

$$opt(n) = \begin{cases} 0, & n = 0 \\ \max\{P[n], \max_{i=1}^{n-1}(P[i] + opt(n-i)) - c\}, & \text{otherwise.} \end{cases}$$

We can still resolve the subproblems following the order: $opt(0), opt(1), opt(2), \ldots, opt(n)$. Clearly, computing $opt(i)$ given $opt(0), opt(1), \ldots, opt(i-1)$ requires $O(i)$ time only. Hence, we can compute $opt(n)$ in $O(n^2)$ time.