1. Describe and analyze an algorithm to compute the maximum total reward that the Antarctican SLUG race organizers could be forced to pay, given the array M as input.

Solution: Let $MaxR(i,j)$ be the maximum possible reward if only the snails numbered i through j are allowed to find mates. We need to compute $MaxR(1,n)$. This function obeys the following recurrence:

$$MaxR(i,j) = \begin{cases} 0 & \text{if } j \le i \\ \max \left\{ \begin{array}{c} MaxR(i+1,j) \\ \max\limits_{i<k\le j} \left(\begin{array}{c} M[i,k] \\ +\,MaxR(i+1,k-1) \\ +\,MaxR(k+1,j) \end{array} \right) \end{array} \right\} & \text{otherwise} \end{cases}$$

If there is at most one relevant snail, no reward is possible. Otherwise, we recursively consider all ways of pairing up snail i. If snail i never finds a mate, the maximum reward is $MaxR(i+1,j)$. If snail i meets snail k, the organizers immediately pay $M[i,k]$, and the two slime trails split the remaining snails into two independent subproblems: snails $i+1$ through $k-1$, and snails $k+1$ through j.

We can memoize this function into a two-dimensional array $MaxR[1..n, 0..n]$. Since each entry $MaxR[i,j]$ depends only on entries $MaxR[i',j']$ with $i' > i$, we can fill the array row by row from the bottom up (decreasing i), filling each row in arbitrary order. The resulting algorithm runs in ***$O(n^3)$ time***.

```
MaxReward(M[1..n, 1..n]):
  for i ← n down to 1
      MaxR[i, i−1] ← 0
      MaxR[i, i] ← 0
      for j ← i+1 to n        ⟨⟨or whatever⟩⟩
          MaxR[i, j] ← MaxR[i+1, j]
          for k ← i+1 to j
              tmp ← M[i, k] + MaxR[i+1, k−1] + MaxR[k+1, j]
              if MaxR[i, j] < tmp
                  MaxR[i, j] ← tmp
  return MaxR[1, n]
```

■

Rubric: 10 points: standard dynamic programming rubric

2. Suppose you are given a NFA $M = (\{0, 1\}, Q, s, A, \delta)$ without ε-transitions and a binary string $w \in \{0, 1\}^*$. Describe and analyze an algorithm to determine whether M accepts w. Report the running time of your algorithm as a function of k (the number of states in M) and n (the length of the input string w).

Solution: For any state p and any index i, let *Accepts?*(p, i) indicate whether the NFA would accept the suffix $w[i\,..\,n]$ if it started in state p, or equivalently, if the set $\delta^*(p, w[i\,..\,n])$ contains an accepting state. We need to compute *Accepts?*$(1, 1)$.

This function obeys the following recurrence:

$$\mathit{Accepts?}(p,i) = \begin{cases} \textsc{True} & \text{if } i > n \text{ and } p \in A \\ \textsc{False} & \text{if } i > n \text{ and } p \notin A \\ \displaystyle\bigvee_{q=1}^{k} \big(q \in \delta(p, w[i]) \land \mathit{Accepts?}(q, i+1)\big) & \text{otherwise} \end{cases}$$

Rewriting this recurrence in terms of our input representation gives us

$$\mathit{Accepts?}(p,i) = \begin{cases} \mathit{Acc}[p] & \text{if } i > n \\ \displaystyle\bigvee_{q=1}^{k} \big(\mathit{inDelta}[p, w[i], q] \land \mathit{Accepts?}(q, i+1)\big) & \text{otherwise} \end{cases}$$

We can memoize this function into a two-dimensional array *Accepts?*$[1\,..\,k, 1\,..\,n]$, which we can fill column by column from right to left, filling each column in arbitrary order, in ***O(k²n) time***.

```
NFA-Accepts(k, Acc[1..k], inDelta[1..k, 0..1, 1..k]):
    for p ← 1 to k
        Accepts?[p, n+1] ← Acc[p]
    for i ← n down to 1
        for p ← 1 to k
            Accepts?[p, i] ← False
            for q ← 1 to k
                if inDelta[p, w[i], q] and Accepts?[q, i+1]
                    Accepts?[p, i] ← True
    return Accepts?[1, 1]
```

■

Rubric: 10 points: standard dynamic programming rubric