CS/ECE 374 FALL 2018	Zhe Zhang (zzhan157@illinois.edu)
Homework 6 Problem 1	Ray Ying (xinruiy2@illinois.edu)
	Anqi Yao (anqiyao2@illinois.edu)

1. The McKing chain wants to open several restaurants along Red street in Shampoo-Banana. The possible locations are at $L_1, L_2, \ldots, L_n$ where L_i is at distance m_i meters from the start of Red street. Assume that the street is a straight line and the locations are in increasing order of distance from the starting point (thus $0 \leq m_1 < m_2 < \ldots < m_n$). McKing has collected some data indicating that opening a restaurant at location L_i will yield a profit of p_i independent of where the other restaurants are located. However, the city of Shampoo-Banana has a zoning law which requires that any two McKing locations should be D or more meters apart. *In addition McKing does not want to open more than k restaurants due to budget constraints.* Describe an efficient algorithm that McKing can use to figure out the maximum profit it can obtain by opening restaurants while satisfying the city's zoning law and the constraint of opening at most k restaurants. Your algorithm should use only $O(n)$ space and you should not assume that k is a constant.

Solution: (Reference Lab 8)
To simplify boundary cases, let's add a location L_{n+1} where L_{n+1} is at distance $m_{n+1} = \infty$.

For any index i, let MaxProfit(i,k) denotes the maximum profit McKing can obtain by opening restaurants on locations $L_i, L_{i+1}, \ldots, L_n$ while satisfying the city's zoning law and the constraint of opening at most k restaurants.

For any index i, let next(i) denotes the smallest index j such that $m_j > m_i$ + D. The possible locations are at $L_1, L_2, \ldots, L_n$. For each possible location L_i, there are two cases: (1) we open one restaurant and open the rest $k-1$ restaurants on other possible locations $L_{i+1}, \ldots, L_n$. (2) we do not open one restaurant and open k restaurants on other possible locations $L_{i+1}, \ldots, L_n$. We will follow the case which gives us the maximum profit. The function MaxProfit(i,k) statisfies the following recurrence:

$$\text{MaxProfit}(i,k) = \begin{cases} 0, & \text{if } i > n \text{ or } k = 0. \\ max\{p_i + \text{MaxProfit}(\text{next}(i), k-1), \text{MaxProfit}(i+1,k)\}, & \text{otherwise.} \end{cases} \tag{1}$$

We can memorize this function into a two-dimensional array MaxProfit[1...n+1,0...k]. Because the array $m_1, m_2, \ldots, m_n$ is sorted, we can compute next(i) for any index i in O(logn) time using binary search. There are O(nk) distinct subproblems so that the total running time is O(nklogn). Space for the aray is O(kn). However, for each subproblem in column k, only column k and column k-1 matter so that instead of memorizing k columns, it is sufficient to memorize only two columns. Then the space for the array is O(n).

■