CS/ECE 374 Spring 2017
Homework 6 Problem 1

Lanxiao Bai (lbai5)
Renheng Ruan (rruan2)
Ho Yin Au (hoyinau2)

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 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.

Solution: Consider a location L_i on Red Street, McKing can either open a restaurant or not, depends on which way it can reach the maximum of profits while the rest is already the optimum solution.

As a result, if we suppose their is a routine NEXT(i) that gives the minimum next i that $L_{\text{NEXT}(i)}$ can open a restaurant if a restaurant is opened at L_i, the max profit MAXPROFITS is as following:

$$\text{MAXPROFIT}(i) = \begin{cases} 0 & \text{if } i > n \text{ or } k \leq 0 \\ \max \begin{cases} p_i + \text{MAXPROFIT}(\text{NEXT}(i), k-1) \\ \text{MAXPROFIT}(i+1, k) \end{cases} & \text{if } i \leq n \text{ and } k > 0 \end{cases}$$

Although we use two parameters to the function, 1-D array of only with length of n is enough to memoize the result, since we can use k to count down.

As a result, MAXPROFITS(i) only depends on MAXPROFITS($i+1$), so we can design a iterative algorithm to fill in the 1-D array as the following.

```
GETMAXPROFIT(L[1..n], p[1..n], k):
  MAXPROFIT[n] = p[n]
  maximum ← MAXPROFIT[n]
  for i ← n − 1 to 1
      if k < 0
          return maximum
      profit1 = MAXPROFIT[NEXT(i)] + p[i]
      profit2 = MAXPROFIT[i + 1]
      if profit1 > profit2
          k ← k − 1
      MAXPROFIT[i] ← max{profit1, profit2}
      maximum ← MAXPROFIT[i]
  return maximum
```

Obviously, this algorithm takes $O(n)$ space to execute, since the data is memoized by a 1-D array. And since the result can be given with a loop of at most the length of n, the time complexity depends on the time complexity of NEXT(i), which can take $O(\log n)$ to execute since m_i is sorted and, as a result, binary search can be used. Hence, the overall time complexity is $O(n \log n)$. ■