Max Profit with k Transactions

Example: You are given an array of integers representing the prices of a single stock on various days (each index in the array represents a different day). You are also given an integer, k, which represents the number of transactions you are allowed to make. One transaction consists of buying the stock on a given day and selling it on another, later day. Write a function that returns the maximum profit that you can make buying and selling the stock given k transactions. Note that you can only hold 1 share of the stock at a time. In other words, you can not buy more than 1 share of the stock on any given day. You also can not buy a share of the stock if you are still holding another share.

Sample input: `[5, 11, 3, 50, 60, 90], 2`

Sample output: `93 (Buy: 5, Sell: 11; Buy 3, Sell :90)`

Solution. We first present a solution that is $O(nk)$ in both time and space complexity. Here, n is the number of days in the input and k is the given number of transactions.

```
# O(nk) time and space.
def maxProfitWithKTransactions(prices, k):

   # Handles the case of empty list of prices.
    if not len(prices):
      return 0

   # Fills a 2D list of size k x len(prices) with 0's.
   # This 2D list will contain the max profits possible for a particular
      number of transactions and price.
   profits = [[0 for p in prices] for t in range(k + 1)]

   # Dynamic programming method of using the previous possible max profits
      to calculate future max profits.
   for t in range (1, k + 1):
      maxThusFar = float("-inf")
      for p in range (1, len(prices)):
         maxThusFar = max(maxThusFar, profits[t - 1][p - 1] - prices[p - 1])
         profits[t][p] = max(profits[t][p - 1], maxThusFar + prices[p])

   # Returns last element of the 2D list.
   return profits[-1][-1]
```

We now present a solution that is $O(nk)$ in time complexity but only $O(n)$ in space complexity. Again, n is the number of days in the input and k is the given number of transactions.

```
# O(nk) time, O(n) space.
def maxProfitWithKTransactions(prices, k):

   # Handles the case of empty list of prices.
    if not len(prices):
      return 0

   # Space complexity is O(n) in this case.
   # We only need to initialize two rows.
   evenProfits = [0 for p in prices]
   oddProfits = [0 for p in prices]

   # Swaps the rows depending on whether the transaction number is even or
       odd.
   # Same overall DP algorithm as before.
   for t in range(1, k + 1):
      maxThusFar = float("-inf")
      if t % 2 == 1:
         currentProfits = oddProfits
         previousProfits = evenProfits
      else:
         currentProfits = evenProfits
         previousProfits = oddProfits
      for p in range(1, len(prices)):
         maxThusFar = max(maxThusFar, previousProfits[p - 1] - prices[p - 1])
         currentProfits[p] = max(currentProfits[p - 1], maxThusFar +
             prices[p])

   # Returns the last element of the correct profits row.
   return evenProfits[-1] if k % 2 == 0 else oddProfits[-1]
```