ABIGAIL RECHKIN
ALEX YAVNILOVITCH

HOMEWORK 6

INPUT Array A[1,....,n] of natural numbers
Integer K for the initial dollar value
Integers b - transaction fee BDC → USD
d - transaction fee USD → BDC
Integer n - the size of A

Output The maximum amount of USD attained by day n, via a sequence of exchanges.

```
1  PROCEDURE    MAXDOLLA$(A[1,...,n], K, b, d, n)
2        maxBDC ← 0
3        maxUSD ← K
4        for i ← 1 to n do:
5              currBDC ← convertBDC(maxUSD, d, A[i])
6              currUSD ← convertUSD(maxBDC, b, A[i])
7              maxBDC ← MAX(maxBDC, currBDC)
8              maxUSD ← MAX(maxUSD, currUSD)
9        return maxUSD
```

Input maxUSD: value in USD to be converted
d - exchange fee USD → BDC
rate - exchange rate for i
Output Result of conversion in BDC

```
1  PROCEDURE  convertBDC
2        return  (maxUSD-d)/rate
```

Input maxBDC: value in BDC to be converted
b - exchange fee BDC → USD
rate - exchange rate for i

```
1  PROCEDURE  convertUSD
2        return  (maxBDC-b)*rate
```

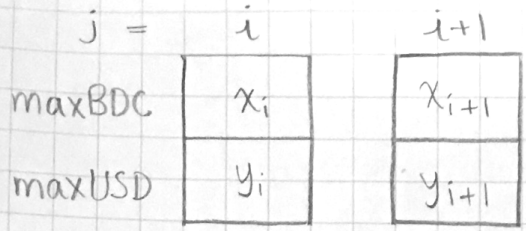
KEY IDEA

- Given: an array A of exchange rates from USD to BDC, transaction fees b, d, and the initial dollar amount,
- GOAL: A maximized amount in USD on day n, to be attained through a series of exchanges.

Our algorithm iterates through the given array. For each day, the procedure test out two cases:
(1) Performs the conversion (for both USD and BDC)
(2) Retains the saved maximum value (for USD, BDC), meaning, no conversion is performed.
The two cases are compared, and our algorithm updates the saved maximum values for USD and BDC. We continue this process for the entire array, until reaching the nth day.

SUBPROBLEMS

We are given the array $A[1,\ldots,n]$ of n exchange rates. We define a subproblem A' as $A'[1,\ldots,j]$, where $1 \leq j \leq n$. We are finding the maximum values of USD and BDC for each subproblem, and eventually, the entire array.
Our initial condition consists of an empty array, where our max USD = K (initial dollar value), and our max BDC = 0, as we have not had a chance to perform a conversion.
Next, our algorithm begin the first iteration, where $j = 1$. We perform both conversion operations, and compare the results to our max USD, and max BDC. We update the maximum values, and proceed to subproblem $A'[1,\ldots,j+1]$
We continue to expand our subproblem, using the bottom-up approach - j increases with each iteration, until $j = n$, meaning we have arrived at the full array.



DP TABLE

For each subproblem, we determine maxBDC and maxUSD, using the following comparison:

$$x_{i+1} = \text{MAX}\left(x_i\ ,\ (y_i - d)/A[i+1]^{*}\right) \qquad {}^{*}j = i+1$$

$$y_{i+1} = \text{MAX}\left(y_i\ ,\ (x_i - b) * A[i+1]\right)$$

Using the above relation between the max values in $j=i$ and $j=i+1$, we observe that the DP table for i is overwritten with the updated max values in $i+1$. Thus, since the size of our DP table is independent of n (size of A), we conclude that the algorithm has space complexity of $O(1)$.

As for TIME COMPLEXITY, the algorithm iterates through the array once, and covers n subproblems. This results in $O(n)$ for the for loop on line 4. Furthermore, the algorithm only performs arithmetic computations locally, resulting in $O(1)$ for each subproblem. Finally, our total time complexity may be expressed as $O(1 \cdot n) = O(n)$

▨