Assignment 4

Ankur Kumar(14109) Tushar Vatsal(14766)

Buying and selling shares multiple times

1 Notations

→ Days are numbered $i = 1, 2, ..., n$
→ There is a price $p(i)$ per share of the stock on i^{th} day.
→ Given a variable k, a $k - shot\ strategy$ is a collection of m pair of days $(b_1, s_1), ..., (b_m, s_m)$ where $0 \leq m \leq k$ and $1 \leq b_1 < s_1 < b_2 < s_2 < .... < b_m < s_m \leq n$
Return of $k - shot\ strategy$

$$Profit(k) = 1000 \sum_{i=1}^{m} (p(s_i) - p(b_i))$$

Aim : To maximize this profit for a given k
→ $Rec\_profit(i, l)$ =Maximum profit using $l - shot\ strategy$ from i^{th} day to n^{th}(last) day.
Aim : To find $Rec\_profit(1, k)$

2 Recursive Formulation of $Rec\_profit(i, l)$

$Rec\_profit(i, l) = max\{Rec\_profit(i+1, l), max_{i<j\leq n}(Rec\_profit(j+1, l-1)+p(j)-p(i))\}$

Base Case:
$Rec\_profit(n, l) = 0 \ \forall \ l \in \{0, 1, 2, ..., k\}$
$Rec\_profit(n + 1, l) = 0 \ \forall \ l \in \{0, 1, 2, ..., k\}$
$Rec\_profit(i, 0) = 0 \ \forall \ i \in \{1, 2, 3, ..., n\}$

3 Proof of Recursive Formulation

$Rec\_profit(i, l) = max\{Rec\_profit(i+1, l), max_{i<j\leq n}(Rec\_profit(j+1, l-1)+p(j)-p(i))\}$

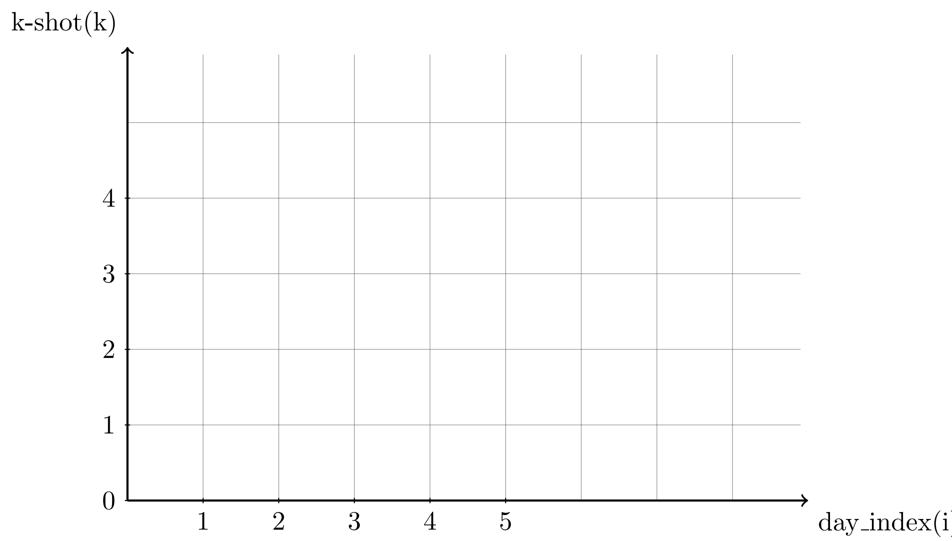
$Rec\_profit(i, l)$ = Maximum profit from i^{th} day to n^{th} day using $l - shot$

strategy.
There are two possible cases :
Case1 : Suppose no transaction occurs at i^{th} day then maximum profit from i^{th} day to n^{th} day using $l - shot\ strategy$ is same as maximum profit from $i+1^{th}$ day to n^{th} day using $l - shot\ strategy$. This means $Rec\_profit(i,l) = Rec\_profit(i+1,l)$
Case2 : Suppose we buy share on i^{th} day then I have to sell this share on j^{th} day for some j such that $i < j \leq n$. For selling this share bought on i^{th} day, we have $n - i$ possible days. Suppose I sell this on some day $j > i$, then maximum possible profit is sum of $(p(j) - p(i))$(profit due to this transaction) and maximum possible profit from $j + 1^{th}$ day to n^{th} day using $l - 1$ $shot\ strategy$. This means maximum possible profit if we sell this on j^{th} day for some $j > i = Rec\_profit(j+1, l-1) + p(j) - p(i)$. Thus maximum possible profit if we sell this share brought on i^{th} day on any day $j > i = max_{i<j\leq n}(Rec\_profit(j+1, l-1) + p(j) - p(i))$
Since we have to maximize the term $Rec\_profit(i,l)$ either by doing any transaction at i^{th} day or not doing it, I will take

$$Rec\_profit(i,l) = max\{Rec\_profit(i+1,l), max_{i<j\leq n}(Rec\_profit(j+1,l-1)+p(j)-p(i))\}$$

4 Implementation using Dynamic Programming

Build a table of $k+1$ rows and $n+1$ columns where k and n have their usual meanings



Let us start calling $Rec\_profit(i,l) = T(i,l)$ from now just to simplify our life.

$$T(i,l) = max\{T(i+1,l), max_{i<j\leq n}(T(j+1,l-1) + p(j) - p(i))\}$$

To calculate any entry $T(i,l)$, we should know $T(i+1,l)$ and all the entries to the right of the block $T(i+1,l-1)$ in the grid shown above.
First we will assign all the entries of n^{th} and $n+1^{th}$ column 0 and all the entries of 0^{th} row 0 according to the base case stated above. And then we will calculate the entries column-wise starting from second-last column from bottom to top. This will ensure while calculating any entry $T(i,l)$ we will know beforehand all other entries used for calculating it.

Space Complexity : $O(n(k+1)) = O(nk+n) = O(nk)$
Time Complexity :
Assigning last column : $O(k+1) = O(k)$
Assigning last row : $O(n)$
As we know that

$$T(i,l) = max\{T(i+1,l), max_{i<j\leq n}(T(j+1,l-1) + p(j) - p(i))\}$$

Thus time required to calculate any block $T(i,l) = c(1) + c(n-i) = c(n-i+1)$, where c is a constant.
Thus total time to calculate all the remaining entries of the table :

$$\sum_{i=1}^{n-1}\sum_{l=1}^{k} c(n-i+1) = ck\sum_{i=1}^{n-1}(n-i+1) = O(n^2k)$$

Thus the entire table can be computed in $O(n^2k) + O(n) + O(k) = O(n^2k)$ time. This is a polynomial time algorithm in n and k.

5 Towards $O(nk)$ time complexity

However we can improve this to $O(nk)$ time complexity. Take a look over the term $T(i,l)$=

$$max\{T(i+1,l), max_{i<j\leq n}(T(j+1,l-1) + p(j) - p(i))\}$$

If this term can be calculated in $O(1)$ time then we are done.
Let's say $max_{i<j\leq n}(T(j+1,l-1) + p(j) - p(i)) = P(i,l)$
$\Rightarrow P(i,l) = (max_{i<j\leq n}(T(j+1,l-1) + p(j)) - p(i)$
Say $max_{i<j\leq n}(T(j+1,l-1) + p(j) = P^{'}(i,l)$
Now $P(i,l) = P^{'}(i,l) - p(i)$
To find $P(i,l)$, we need $P^{'}(i,l)$ and $p(i)$
While calculating i^{th} column of the table we will keep an array M of size k such that $M[l] = P^{'}(i,l)$
Initalize all the elements of $M = 0$
After initializing n^{th} column and 0^{th} row of the grid start calculating the grid T from $n-1^{th}$ to the last column according to these rule :

$T(i,l) = max\{T(i+1,l), M[l] - p(i)\}$. After calculating $T(i,l)$ update $M[l] = max\{M[l], T(i+1, l-1) + p(i)\}$
Thus we are able to calculate every entry in $O(1)$ time and thus total time complexity goes to $O(nk)$