Algorithm 1 A $\Theta(m * n^2)$ time brute force algorithm for solving Problem1

```
function BRUTE-FORCE(m, n, prices[m][n])
    $maxProfit \leftarrow 0$
    $stock \leftarrow 0$
    $buyDay \leftarrow 0$
    $sellDay \leftarrow 0$
    for k=1 to m do
        for i=1 to n do
            for j=i+1 to n do
                $diff \leftarrow prices[k][j] - prices[k][i];$
                if $diff > maxProfit$ then
                    $buyDay \leftarrow i$
                    $sellDay \leftarrow j$
                    $stock \leftarrow k$
                    $maxProfit \leftarrow diff$
                end if
            end for
        end for
    end for
    print $stock, buyDay, sellDay$
end function
```

Algorithm 2 A $\Theta(m * n)$ time greedy algorithm for solving Problem1

```
function GREEDY(m, n, prices[m][n])
    $globalMaxProfit \leftarrow 0$
    $stock \leftarrow 0$
    $buyDay \leftarrow 0$
    $sellDay \leftarrow 0$
    for j=1 to m do
        $minPrice \leftarrow \infty$
        $maxProfit \leftarrow -\infty$
        $currBuyDay \leftarrow 0$
        $currSellDay \leftarrow 0$
        for i=1 to n do
            if $prices[j][i] < minPrice$ then
                $minPrice \leftarrow prices[j][i]$
                $currBuyDay \leftarrow i$
            else if $prices[j][i] - minPrice > maxProfit$ then
                $currSellDay \leftarrow i$
                $maxProfit \leftarrow prices[j][i] - minPrice$
            end if
        end for
        if $globalMaxProfit < maxProfit$ then
            $globalMaxProfit \leftarrow maxProfit$
            $buyDay \leftarrow currBuyDay$
            $sellDay \leftarrow currSellDay$
            $stock \leftarrow j$
        end if
    end for
    print $stock, buyDay, sellDay$
end function
```

Algorithm 3 A $\Theta(m * n)$ time dynamic programming algorithm for solving Problem1

```
function DYNAMIC-PROGRAMMING(m, n, prices[m][n])
    for i=0 to m do
        $M[i][1] \leftarrow 0$
    end for
    for i=0 to n do
        $M[0][i] \leftarrow 0$
    end for
    for i=1 to m do
        $P[i][1] \leftarrow 1$
        for j=2 to n do
            if $prices[i][P[i][j-1]] < prices[i][j-1]$ then
                $P[i][j] \leftarrow P[i][j-1]$
            else
                $P[i][j] \leftarrow j-1$
            end if
        end for
    end for
    for i=1 to m do
        for j=2 to n do
            $M[i][j] \leftarrow max\{prices[i][j] - prices[i][P[i][j]], M[i-1][j], M[i][j-1]\}$
        end for
    end for
    FIND-SOLUTION(m, n)
end function

function FIND-SOLUTION(i, j)
    if $M[i][j] = M[i-1][j]$ then
        FIND-SOLUTION(i-1, j)
    else if $M[i][j] = M[i][j-1]$ then
        FIND-SOLUTION(i, j-1)
    else
        print $m, P[m][n], n$
    end if
end function
```

Algorithm 4 A $\Theta(m * n^{2k})$ time brute force algorithm for solving Problem2

```
function BRUTE-FORCE(k, m, n, prices[m][n])
    $result \leftarrow \{\}$
    $maxProfit, answer \leftarrow$ FIND-SOLUTION(1, k, false, 0, 0, result)
    print $answer$
end function

function FIND-SOLUTION(day, trans, considerBuy, stock, buyDay, result)
    $maxProfit \leftarrow 0$
    if $day > n$ or $trans = 0$ then
        return $maxProfit, result$
    end if
    if $considerBuy = true$ then
        $currProfit, currResult \leftarrow$ FIND-SOLUTION(day + 1, trans, true, stock, buyDay, result)
        if $currProfit > maxProfit$ then
            $maxProfit \leftarrow currProfit$
            $optResult \leftarrow currResult$
        end if
        $diff \leftarrow prices[stock][day] - prices[stock][buyDay]$
        $result \leftarrow result \cup (stock, buyDay, day)$
        $currProfit, currResult \leftarrow$ FIND-SOLUTION(day, trans - 1, false, 0, 0, result)
        $currProfit \leftarrow currProfit + diff$
        if $currProfit > maxProfit$ then
            $maxProfit \leftarrow currProfit$
            $optResult \leftarrow currResult$
        end if
    else
        $currProfit, currResult \leftarrow$ FIND-SOLUTION(day + 1, trans, false, 0, 0, result)
        if $currProfit > maxProfit$ then
            $maxProfit \leftarrow currProfit$
            $optResult \leftarrow currResult$
        end if
        for i=1 to m do
            $currProfit, currResult \leftarrow$ FIND-SOLUTION(day + 1, trans, true, i, day, result)
            if $currProfit > maxProfit$ then
                $maxProfit \leftarrow currProfit$
                $optResult \leftarrow currResult$
            end if
        end for
    end if
    return $maxProfit, optResult$
end function
```

Algorithm 5 A $\Theta(m * n^2 * k)$ time dynamic programming algorithm for solving Problem2

```
function DYNAMIC-PROGRAMMING(k, m, n, stocks[m][n])
    $buy \leftarrow \{\}$
    $sell \leftarrow \{\}$
    $stockNumber \leftarrow \{\}$
    FIND-MAX-PROFIT(k, m, n, stocks)
    $reqTrans \leftarrow 0$
    $max \leftarrow 0$
    for i=0 to k do
        if $dp[i][m][n-1] > max$ then
            $max \leftarrow dp[i][m][n-1]$
            $reqTrans \leftarrow i$
        end if
    end for
    FIND-BUY-SELL-INDEX(k, m, n-1, 1, 0, stocks, reqTrans, 0)
    $i \leftarrow length(sell) - 1$
    $j \leftarrow length(buy) - 1$
    $s \leftarrow length(stockNumber) - 1$
    while $i >= 0$ and $j >= 0$ do
        print $stockNumber[s] - 1, buy[j]$
        $s \leftarrow s - 1$
        print $sell[i]$
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
        $s \leftarrow s - 1$
    end while
end function

function FIND-MAX-PROFIT(k, m, n, stocks)
    for i=1 to k do
        for j=1 to m do
            for p=1 to n-1 do
                $maxProfit \leftarrow -\infty$
                for q=0 to p-1 do
                    $currentProfit \leftarrow stocks[j-1][p] - stocks[j-1][q] + dp[i-1][m][q]$
                    if $currentProfit > maxProfit$ then
                        $maxProfit \leftarrow currentProfit$
                    end if
                end for
                $dp[i][j][p] \leftarrow max\{maxProfit, dp[i][j][p-1], dp[i][j-1][p]\}$
            end for
        end for
    end for
end function
```

```
function FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum, sellNum, required, stock, transLeft, prevStockIndex)
    if $transLeft = 0$ then
        return
    end if
    if $sellNum = 1$ then
        if $dp[tIndex][stockNum][dayNum] \neq dp[tIndex][stockNum][dayNum]$ and $dp[tIndex][stockNum][dayNum] \neq dp[tIndex][stockNum-1][dayNum]$ then
            $sell \leftarrow sell \cup dayNum$
            $stockNumber \leftarrow stockNumber \cup stockNum$
            FIND-BUY-SELL-INDEX(tIndex-1, length(dp[0])-1, day-1, 0, dp[tIndex][stockNum][day] - stock[stockNum - 1][day], stock, transLeft, stockNum)
        else if $dp[tIndex][stockNum][dayNum] = dp[tIndex][stockNum][dayNum-1]$ then
            FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum - 1, sellNum, required, stock, Transneft, stockNum)
        else
            FIND-BUY-SELL-INDEX(tIndex, stockNum - 1, dayNum, sellNum, required, stock, transLeft, stockNum)
        end if
    else
        if $dp[tIndex][stockNum][dayNum] - stock[prevStockIndex-1][dayNum] = required$ then
            $buy \leftarrow buy \cup dayNum$
            $stockNumber \leftarrow stockNumber \cup prevStockIndex$
            FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum, 1, required, stock, transLeft - 1, stockNum)
        else
            FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum - 1, 0, required, stock, transLeft, prevStockIndex)
        end if
    end if
end function
```

Algorithm 6 A $\Theta(m * n * k)$ time dynamic programming algorithm for solving Problem2

```
function DYNAMIC-PROGRAMMING(k, m, n, stocks[m][n])
    $buy \leftarrow \{\}$
    $sell \leftarrow \{\}$
    $stockNumber \leftarrow \{\}$
    FIND-MAX-PROFIT(k, m, n, stocks)
    $reqTrans \leftarrow 0$
    $max \leftarrow 0$
    for i=0 to k do
        if $dp[i][m][n-1] > max$ then
            $max \leftarrow dp[i][m][n-1]$
            $reqTrans \leftarrow i$
        end if
    end for
    FIND-BUY-SELL-INDEX(k, m, n-1, 1, 0, stocks, reqTrans, 0)
    $i \leftarrow length(sell) - 1$
    $j \leftarrow length(buy) - 1$
    $s \leftarrow length(stockNumber) - 1$
    while $i >= 0$ and $j >= 0$ do
        print $stockNumber[s] - 1, buy[j]$
        $s \leftarrow s - 1$
        print $sell[i]$
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
        $s \leftarrow s - 1$
    end while
end function

function FIND-MAX-PROFIT(k, m, n, stocks)
    for i=1 to k do
        for i=x to n-1 do
            $dp[i][0][x] \leftarrow 0$
        end for
        for j=1 to m do
            $maxDiff \leftarrow -stocks[j-1][0]$
            for p=1 to n-1 do
                $dp[i][j][p] \leftarrow max\{stocks[j-1][p] + maxDiff, dp[i][j][p-1], dp[i][j-1][p]\}$
                $maxDiff \leftarrow max\{maxDiff, dp[i-1][m][p] - stocks[j-1][p]\}$
            end for
        end for
    end for
end function
```

```
function FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum, sellNum, required, stock, transLeft, prevStockIndex)
    if $transLeft = 0$ then
        return
    end if
    if $sellNum = 1$ then
        if $dp[tIndex][stockNum][dayNum] \neq dp[tIndex][stockNum][dayNum]$ and $dp[tIndex][stockNum][dayNum] \neq dp[tIndex][stockNum - 1][dayNum]$ then
            $sell \leftarrow sell \cup dayNum$
            $stockNumber \leftarrow stockNumber \cup stockNum$
            FIND-BUY-SELL-INDEX(tIndex-1, length(dp[0])-1, day-1, 0, dp[tIndex][stockNum][day] - stock[stockNum - 1][day], stock, transLeft, stockNum)
        else if $dp[tIndex][stockNum][dayNum] = dp[tIndex][stockNum][dayNum - 1]$ then
            FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum - 1, sellNum, required, stock, Transneft, stockNum)
        else
            FIND-BUY-SELL-INDEX(tIndex, stockNum - 1, dayNum, sellNum, required, stock, transLeft, stockNum)
        end if
    else
        if $dp[tIndex][stockNum][dayNum] - stock[prevStockIndex - 1][dayNum] = required$ then
            $buy \leftarrow buy \cup dayNum$
            $stockNumber \leftarrow stockNumber \cup prevStockIndex$
            FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum, 1, required, stock, transLeft - 1, stockNum)
        else
            FIND-BUY-SELL-INDEX(tIndex, stockNum, dayNum - 1, 0, required, stock, transLeft, prevStockIndex)
        end if
    end if
end function
```

Algorithm 7 A $\Theta(m2^n)$ time brute force algorithm for solving Problem3

```
function BRUTE-FORCE(c, m, n, prices[m][n])
    $l \leftarrow \{\}$
    $l1 \leftarrow \{\}$
    $globalMax \leftarrow 0$
    FIND-MAX-PROFIT(0, 1, 0, l1, 0)
    for i=0 to length(l1) do
        print $l[i][0], l[i][1], l[i][2]$
    end for
end function
function FIND-MAX-PROFIT(day, buy, prevBuy, l1, prevProfit)
    if $day >= n$ then
        return 0
    end if
    if $buy = 1$ then
        $profit1 \leftarrow$ FIND-MAX-PROFIT(day + 1, buy, prevBuy, l1, prevProfit)
        $profit2 \leftarrow$ FIND-MAX-PROFIT(day + 1, 0, day, l1, prevProfit)
        return $max\{profit1, profit2\}$
    else
        $profit1 \leftarrow$ FIND-MAX-PROFIT(day + 1, buy, prevBuy, l1, prevProfit)
        $maxProfit \leftarrow 0$
        $stockIndex \leftarrow 0$
        for i=0 to m-1 do
            if $prices[i][day] - prices[i][prevBuy] > maxProfit$ then
                $maxProfit \leftarrow prices[i][day] - prices[i][prevBuy]$
                $stockIndex \leftarrow i$
            end if
        end for
        $current \leftarrow \{\}$
        $l2 \leftarrow l1$
        $current \leftarrow current \cup stockIndex$
        $current \leftarrow current \cup prevBuy$
        $current \leftarrow current \cup day$
        $l2 \leftarrow l2 \cup current$
        $sellProfit \leftarrow maxProfit + prevProfit$
        if $sellProfit >= globalMax$ then
            if $sellProfit = globalMax$ then
                if $length(l) = 0$ or $length(l) > length(l1)$ then
                    $l \leftarrow l2$
                end if
            else
                $l \leftarrow l2$
                $globalMax \leftarrow sellProfit$
            end if
        end if
        $profit2 \leftarrow$ FIND-MAX-PROFIT(day + 1 + c, 1, 0, l2, sellProfit) + maxProfit
        return $max\{profit1, profit2\}$
    end if
end function
```

Algorithm 8 A $\Theta(m * n^2)$ time dynamic programming algorithm for solving Problem3

```
function DYNAMIC-PROGRAMMING(c, m, n, prices[m][n])
    $l \leftarrow \{\}$
    $l1 \leftarrow \{\}$
    $globalMax \leftarrow 0$
    print FIND-MAX-PROFIT(0, 1, 0, l1, 0)
end function
function FIND-MAX-PROFIT(day, buy, prevBuy, l1, prevProfit)
    if $day >= n$ then
        return 0
    end if
    if $prevBuy \neq 0$ and $dp[day][prevBuy][buy] \neq 0$ then
        return $dp[day][prevBuy][buy]$
    end if
    if $buy = 1$ then
        $profit1 \leftarrow$ FIND-MAX-PROFIT(day + 1, buy, prevBuy, l1, prevProfit)
        $profit2 \leftarrow$ FIND-MAX-PROFIT(day + 1, 0, day, l1, prevProfit)
        $dp[day][prevBuy][buy] \leftarrow max\{profit1, profit2\}$
        return $max\{profit1, profit2\}$
    else
        $profit1 \leftarrow$ FIND-MAX-PROFIT(day + 1, buy, prevBuy, l1, prevProfit)
        $maxProfit \leftarrow 0$
        $stockIndex \leftarrow 0$
        for i=0 to m-1 do
            if $prices[i][day] - prices[i][prevBuy] > maxProfit$ then
                $maxProfit \leftarrow prices[i][day] - prices[i][prevBuy]$
                $stockIndex \leftarrow i$
            end if
        end for
        $current \leftarrow \{\}$
        $l2 \leftarrow l1$
        $current \leftarrow current \cup stockIndex$
        $current \leftarrow current \cup prevBuy$
        $current \leftarrow current \cup day$
        $l2 \leftarrow l2 \cup current$
        $sellProfit \leftarrow maxProfit + prevProfit$
        if $sellProfit >= globalMax$ then
            if $sellProfit = globalMax$ then
                if $length(l) = 0$ or $length(l) > length(l1)$ then
                    $l \leftarrow l2$
                end if
            else
                $l \leftarrow l2$
                $globalMax \leftarrow sellProfit$
            end if
        end if
        $profit2 \leftarrow$ FIND-MAX-PROFIT(day + 1 + c, 1, 0, l2, sellProfit) + maxProfit
        $dp[day][prevBuy][buy] \leftarrow max\{profit1, profit2\}$
        return $max\{profit1, profit2\}$
    end if
end function
```

Algorithm 9 A $\Theta(m * n)$ time dynamic programming algorithm for solving Problem3

```
        function DYNAMIC-PROGRAMMING(c, m, n, prices[m][n])
            for j=0 to m do
                $M[j][1] \leftarrow 0$;
                $N[j][1] \leftarrow -prices[j][1]$;
            end for
            for j=0 to n do
                $M[0][j] \leftarrow 0$;
                $N[0][j] \leftarrow 0$;
            end for
            COMPUTE-OPT-SELL(m, n);
            FIND-SOLUTION-SELL(m, n);)
        end function

        function COMPUTE-OPT-SELL(m, j)
            if $M[m][j]$ is uninitialized then M[m][j] ← $max${ COMPUTE-OPT-BUY(m, j - 1) $+prices[m][j]$, COMPUTE-OPT-SELL(m, j - 1) ,COMPUTE-OPT-SELL(m - 1, j) }
        end if
        return $M[m][j]$
    end function

    function COMPUTE-OPT-BUY(m, j)
        if $N[m][j]$ is uninitialized then N[m][j] ← $max${ MAXIMUM(j - c - 1) $-prices[m][j]$, COMPUTE-OPT-BUY(m, j - 1) }
    end if
    return $N[m][j]$
end function
```

```
function MAXIMUM(j)
    if $X[j]$ is uninitialized then          max ← 0
        for i=1 to m do
            $value \leftarrow$ COMPUTE-OPT-SELL(i, j)
            if $value > max$ then
                $max \leftarrow value$
                $X[j] \leftarrow i$
            end if
        end for

        return $M[X[j]][j]$


        function FIND-SOLUTION-SELL(m, n)
            if $m <= 0$ or $n <= 0$ then
                return
            end if
            if $M[m][n] = M[m-1][n]$ then
                FIND-SOLUTION-SELL(m - 1, n)
            else if $M[m][n] = M[m][n-1]$ then
                FIND-SOLUTION-SELL(m, n - 1)
            else
                FIND-SOLUTION-BUY(m, n - 1)
                print $n$
            end if
        end function

        function FIND-SOLUTION-BUY(m, n)
            if $m <= 0$ or $n <= 0$ then
                return
            end if
            if $N[m][n] = N[m][n-1]$ then
                FIND-SOLUTION-BUY(m, n - 1)
            else
                if $n - c - 1 > 1$ and $X[k-1][n]$ is initialized then
                    FIND-SOLUTION-SELL(X[n - c - 1], n)
                end if
                print $m, n$
            end if
        end function=0
```