In a daily share trading, a buyer buys shares in the morning and sells it on same day. If the trader is allowed to make at most 2 transactions in a day, where as second transaction can only start after first one is complete (Sell->buy->sell->buy). Given stock prices throughout day, find out maximum profit that a share trader could have made.

Examples:

```
Input:   price[] = {10, 22, 5, 75, 65, 80}
Output:  87
Trader earns 87 as sum of 12 and 75
Buy at price 10, sell at 22, buy at 5 and sell at 80

Input:   price[] = {2, 30, 15, 10, 8, 25, 80}
Output:  100
Trader earns 100 as sum of 28 and 72
Buy at price 2, sell at 30, buy at 8 and sell at 80

Input:   price[] = {100, 30, 15, 10, 8, 25, 80};
Output:  72
Buy at price 8 and sell at 80.

Input:   price[] = {90, 80, 70, 60, 50}
Output:  0
Not possible to earn.
```

We strongly recommend to minimize your browser and try this yourself first.

A **Simple Solution** is to to consider every index 'i' and do following

```
Max profit with at most two transactions =
       MAX {max profit with one transaction and subarray price[0..i] +
            max profit with one transaction and aubarray price[i+1..n-1]  }
i varies from 0 to n-1.
```

Maximum possible using one transaction can be calculated using following O(n) algorithm

Maximum difference between two elements such that larger element appears after the smaller number

Time complexity of above simple solution is O(n^2).

We can do this O(n) using following **Efficient Solution**. The idea is to store maximum possible profit of every subarray and solve the problem in following two phases.

1) Create a table profit[0..n-1] and initialize all values in it 0.

2) Traverse price[] from right to left and update profit[i] such that profit[i] stores maximum profit achievable from one transaction in subarray price[i..n-1]

3) Traverse price[] from left to right and update profit[i] such that profit[i] stores maximum profit such that profit[i] contains maximum achievable profit from two transactions in subarray price[0..i].

4) Return profit[n-1]

To do step 1, we need to keep track of maximum price from right to left side and to do step 2, we need to keep track of minimum price from left to right. Why we traverse in reverse directions? The idea is to save space, in second step, we use same array for both purposes, maximum with 1 transaction and maximum with 2 transactions. After an iteration i, the array profit[0..i] contains maximum profit with 2 transactions and profit[i+1..n-1] contains profit with two transactions.

Below is C++ implementation of above idea.

```
// C++ program to find maximum possible profit with at most
// two transactions
#include<iostream>
using namespace std;

// Returns maximum profit with two transactions on a given
// list of stock prices, price[0..n-1]
int maxProfit(int price[], int n)
{
    // Create profit array and initialize it as 0
    int *profit = new int[n];
    for (int i=0; i<n; i++)
        profit[i] = 0;

    /* Get the maximum profit with only one transaction
       allowed. After this loop, profit[i] contains maximum
       profit from price[i..n-1] using at most one trans. */
    int max_price = price[n-1];
    for (int i=n-2;i>=0;i--)
    {
        // max_price has maximum of price[i..n-1]
        if (price[i] > max_price)
            max_price = price[i];

        // we can get profit[i] by taking maximum of:
        // a) previous maximum, i.e., profit[i+1]
        // b) profit by buying at price[i] and selling at
        //    max_price
        profit[i] = max(profit[i+1], max_price-price[i]);
    }

    /* Get the maximum profit with two transactions allowed
       After this loop, profit[n-1] contains the result */
    int min_price = price[0];
    for (int i=1; i<n; i++)
    {
        // min_price is minimum price in price[0..i]
        if (price[i] < min_price)
            min_price = price[i];

        // Maximum profit is maximum of:
        // a) previous maximum, i.e., profit[i-1]
        // b) (Buy, Sell) at (min_price, price[i]) and add
        //    profit of other trans. stored in profit[i]
        profit[i] = max(profit[i-1], profit[i] +
                                    (price[i]-min_price) );
    }
    int result = profit[n-1];

    delete [] profit; // To avoid memory leak

    return result;
}

// Drive program
int main()
{
    int price[] = {2, 30, 15, 10, 8, 25, 80};
    int n = sizeof(price)/sizeof(price[0]);
    cout << "Maximum Profit = " << maxProfit(price, n);
    return 0;
```

```
}
```

Output:

```
Maximum Profit = 100
```

Time complexity of the above solution is O(n).