***Stock Buy and Sell Problem***

The cost of a stock on each day is given in an array. Find the maximum profit that you can make by buying and selling on those days. If the given array of prices is sorted in decreasing order, then profit cannot be earned at all.

**Examples:**

***Input:****arr[] = {100, 180, 260, 310, 40, 535, 695}*  
***Output:****865*  
***Explanation:****Buy the stock on day 0 and sell it on day 3 => 310 – 100 = 210*  
*Buy the stock on day 4 and sell it on day 6 => 695 – 40 = 655*  
*Maximum Profit  = 210 + 655 = 865*

***Input:****arr[] = {4, 2, 2, 2, 4}*  
***Output:****2*  
***Explanation:****Buy the stock on day 1 and sell it on day 4 => 4 – 2 = 2*  
*Maximum Profit  = 2*

A simple approach is to try buying the stocks and selling them every single day when profitable and keep updating the maximum profit so far.

Follow the steps below to solve the problem:

* Try to buy every stock from **start** to **end – 1**
* After that again call the maxProfit function to calculate answer
* curr\_profit = price[j] – price[i] + maxProfit(start, i – 1) + maxProfit(j + 1, end)
* profit = max(profit, curr\_profit)

Below is the implementation of the above approach:

C++Java

// Java implementation of the approach

import java.util.\*;

class GFG {

// Function to return the maximum profit

// that can be made after buying and

// selling the given stocks

static int maxProfit(int price[], int start, int end)

{

// If the stocks can't be bought

if (end <= start)

return 0;

// Initialise the profit

int profit = 0;

// The day at which the stock

// must be bought

for (int i = start; i < end; i++) {

// The day at which the

// stock must be sold

for (int j = i + 1; j <= end; j++) {

// If buying the stock at ith day and

// selling it at jth day is profitable

if (price[j] > price[i]) {

// Update the current profit

int curr\_profit

= price[j] - price[i]

+ maxProfit(price, start, i - 1)

+ maxProfit(price, j + 1, end);

// Update the maximum profit so far

profit = Math.max(profit, curr\_profit);

}

}

}

return profit;

}

// Driver code

public static void main(String[] args)

{

int price[] = { 100, 180, 260, 310, 40, 535, 695 };

int n = price.length;

System.out.print(maxProfit(price, 0, n - 1));

}

}

**Output**

865

**Time Complexity:** O(N2), Trying to buy every stock and exploring all possibilities.  
**Auxiliary Space:** O(1)

**Stock Buy Sell to Maximize Profit using Valley Peak Approach:**

In this approach, we just need to find the next greater element and subtract it from the current element so that the difference keeps increasing until we reach a minimum. If the sequence is a decreasing sequence, so the maximum profit possible is 0.

Follow the steps below to solve the problem:

* maxProfit = 0
* if price[i] > price[i – 1]
  + maxProfit = maxProfit + price[i] – price[i – 1]

Below is the implementation of the above approach:

C++Java

import java.util.\*;

import java.io.\*;

import java.lang.\*;

class GFG

{

static int maxProfit(int price[], int n)

{

int profit = 0;

for(int i = 1; i < n; i++)

{

if(price[i] > price[i - 1])

profit += price[i] - price[i -1];

}

return profit;

}

public static void main(String args[])

{

int arr[] = {1, 5, 3, 8, 12}, n = 5;

System.out.println(maxProfit(arr, n));

}

}

**Output**

13

**Time Complexity**: O(N), Traversing over the array of size N.  
**Auxiliary** **Space:**O(1)