Question: https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/

Constraints: You can only hold ****at most one**** share of the stock at any time. However, you can buy it then immediately sell it on the ****same day****.  
   
So what we need to do is find out maximum profit by buying and selling multiple times.

In order to maximize the profit we should buy at local minimas, and sell at local maximas.

So the first approach that comes to my mind is using stacks to find out maxima and minima.

We push the 0th element to the initially then, check while iterating(from 1 to n) through the array that if prices[i]<stack’s top element, then that means we have reached local maxima for that transaction, now if the stack is empty after removing the maxima, then it implies that maxima and minima for that transaction were same and the profit would be 0.

Whereas if there are elements after eliminating the maxima then empty the stack and the last element of the stack would be minima in that case.

And the profit would be maxima-minima.

Solution:  
class Solution {

public int maxProfit(int[] prices) {

int sell, buy, profit=0;

Stack<Integer> stock = new Stack<>();

stock.push(prices[0]);

if(prices.length == 1){

return 0;

}

for(int i=1; i<prices.length; i++){

if(stock.peek()>prices[i]){

sell=stock.peek();

buy=-1;

while(!stock.empty()){

buy=stock.peek();

stock.pop();

}

if(buy != -1){

profit+=(sell-buy);

}

}

stock.push(prices[i]);

}

if(!stock.empty())

{

sell=stock.peek();

buy=-1;

while(!stock.empty())

{

buy=stock.peek();

stock.pop();

}

if(buy != -1){

profit+=(sell-buy);

}

}

return profit;

}

}

Now if we think carefully we will find that we are once visiting the minima and then re-itirating the stack to find minima when we find maxima, rather we can optimize it.

We can find where the increasing subarray start and end.

Iterate the array and check

* while(i<n-1 && prices[i]>prices[i+1])

Increase value of i. then we find the ending point of decreasing subarray, which will be local minima.

* while(i<n-1 && prices[i]<prices[i+1])

Increase value of i. then we find the ending point of increasing subarray, which will be local maxima.

Then that would help us find profit, i.e., profit=maxima-minima

Solution:

class Solution {

public int maxProfit(int[] prices) {

int profit=0, buy=0, sell=0, n=prices.length;

for(int i=0; i<n; i++){

while(i<n-1 && prices[i]>prices[i+1])i++;

buy=prices[i];

while(i<n-1 && prices[i]<prices[i+1])i++;

sell=prices[i];

profit+=sell-buy;

}

return profit;

}

}

Github Link :<https://lnkd.in/ecwtJeaz>