Daily Coding Problem

Good morning! Here's your coding interview problem for today.

This problem was asked by Affirm.

Given a array of numbers representing the stock prices of a company in chronological order, write a function that calculates the maximum profit you could have made from buying and selling that stock. You're also given a number `fee` that represents a transaction fee for each buy and sell transaction.

You must buy before you can sell the stock, but you can make as many transactions as you like.

For example, given [1, 3, 2, 8, 4, 10] and `fee = 2`, you should return 9, since you could buy the stock at 1 dollar, and sell at 8 dollars, and then buy it at 4 dollars and sell it at 10 dollars. Since we did two transactions, there is a 4 dollar fee, so we have 7 + 6 = 13 profit minus 4 dollars of fees.

Comparison with the Microsoft challenge

<u>Changes required:</u>

In the recursive method, it no longer needs to go through all items.
In Microsoft challenge, the items were not orderly. So even though the j loop was the second consecutive item… the index for the k loop could potentially be an element lower than j.
So both of these were possibility i<j<k or i<j>k

In Affirm challenge, its chronological order… So, the following is critical (see below).
Otherwise it will end up progressing chronologically but move historically to stock prices once the recursion occurs. Also i<j<k

```
for i
   for j
        //method call…

                nextnumberconsecutive = nextnumbercheck(i);
  end for j
end for i
```

static boolean nextnumbercheck(int firstNum)

for(k = j+1; k<nums.length; k++)

Also it is worth trying to analyse why three loops were used in total for consecutive number example (realised it was not necessary as stated below).
And how this can be reduced to a single nested for loop in the Afirrm example (again not possible as explained).

for i
    for j
        recursive method call

recursive method
 for k

As oppose to:

 for i

recursive method call()

recursive method()
for j

Demonstration for two nested for loops (consecutive number example)

static int []nums = new int[]{7,6,5,4,3,2,200,201,9,9};

With this structure for i (inside main method) (run once)
for j (main method (run once)
do
(recursive method)
}while(nextnumberconsecutive);

recursive method()
for k (run through entire nums.length until while loop fails..)

if i=0 nums[0]=7.
It would run through j loop and not find nums[j]==nums[i]+differenceCheck

if i=1 nums[1]=6
It would run through j loop and find nums[j]==nums[i]+differenceCheck where j=0. The value

of nums[j] would be 7
it would now force the recursive method...
it would execute the k loop (0 to nums.length) in recursive method.
No instances of nums[k] where it equals to nums[j] +1.

if i=2 nums[2]=5
It would run through j loop and find nums[j]==nums[i]+differenceCheck where j=1.
The value of nums[j] would be 6
it would now force the recursive method...
it would execute the k loop (0 to nums.length) in recursive method.
it would run through the k loop and find instance of nums[k] = nums[j] + 1 where k =0
The value of nums[k] would be 7

Demonstration for single nested for loops (consecutive number example)

static int []nums = new int[]{7,6,5,4,3,2,200,201,9,9};

With this structure for i (inside main method) (run once)
do{
(recursive method)
}while(nextnumberconsecutive);

recursive method()
for j (run through entire nums.length until while condition fails)

if i=0 nums[0]=7
it would now force the recursive method...
it would execute the j loop (0 to nums.length) in recursive method.
It would run through j loop and not find nums[j]==nums[i]+differenceCheck
(NO ISSUES SO FAR)

if i=1 nums[1]=6
it would now force the recursive method...
it would execute the j loop (0 to nums.length) in recursive method.
It would run through j loop and find nums[j]==nums[i]+differenceCheck where j=0. The value of nums[j] would be 7
(NO ISSUES SO FAR)

if i=2 nums[2]=5
it would now force the recursive method...
it would execute the j loop (0 to nums.length) in recursive method.
It would run through j loop and find nums[j]==nums[i]+differenceCheck where j=1. The value of nums[j] would be 6
(NO ISSUES SO FAR)
It would run through j loop and find no instances where nums[j]==nums[i]+differenceCheck even though the nums[0]=7. This is because nums[i] + differenceCheck would still be 6.

This was actually a simple issue to resolve. Every time it found a match in the recursive call, it had to increase the value in differenceCheck by 1...
In this case nums[i] + differenceCheck(2) would be 5+2 = 7. It would also set differenceCheck back to 1 in start of the for i loop
(NO ISSUES SO FAR)

For all my progress in coding, I consider this to be something I should have picked up on. But it can easily happen since I picked up on old code and started enhancing it. This is something I need to refrain from..

Now, it has to be realised at this point that although it seems possible that either a nested loop or double nested loop would solve problem for stock prices, neither are actually suitable.
It would perhaps be good enough for If there were few entries for stock prices as per the example in the challenge.

But if there were lots prices such as {1, 3, 2,7,4,9,4,8} or {1,3,2,8,6,5,7,10}
It can be seen that B=1 S=3, B=2, S=7, B=4, S=9, B=4 S=8
This is one possibility. We could easily skip B=2, S=7, but then B=4, S=9 would be invalid since B=4 is higher than S=3.
There would be lots of issues since it might be possible to remove multiple buy and sell entries in between at various points.

With the stock price code, following conditions need to be met, but it becomes illogical as you move forward through j.

```
if (j!=nums.length)
nums[j]>nums[i]  &&   nums[j+1]<nums[j]


If (j=nums.length-1)
 nums[j]>nums[i]  &&   nums[j+1]<nums[j]  && nums[j+2]>nums[j+1]
```

I did try along this path last time, and it becomes very hit and miss...
There is no point trying to solve the smaller problem via improvisation, it should scale upwards...

Perhaps its an indication that my level of coding is still not ready yet.
But I can recognise the pitfalls a bit better now without committing to a difficult challenge. This is something I have learnt through my coding....

We know that these are all chronological..

So in the first for loop i nested with for loop j, it can refer to items in pairs such as #
1,3 (this is allowed since it can be bought at 2), 1,2 (not allowed since it would be bought at 8), 1,4 not allowed since there is only opportunity to B at 10 but not sell, 1,10 is fully valid...

if i pairs with last item, the last item has to be higher...
Any other circumstance follows the rule above.

This is the straight forward aspect of the problem (with a single buy and a single sell).
It becomes complicated on multiple buys and sells if there was long chronological prices...

But we know, that any trading will occur after position j (unless last item) with multiple buy and sells.
The below prices need to be extended to ensure correct logic is applied:
1,3,2,8,6,5,7,10

B=1 S=3
B=2 S=8 or B=2 S=6 or B=2 S=7 or B=2 S=10
Whichever sell option is taken, the next buy index would be a position after the sell.
But the logic is very extreme.

Since here are a few examples of valid scenarios. It is even more difficult since the problem does not state that stock has to be sold at last entry. It is an assumption based on their example... In which case, the below would be void...
B=1 S=3 B=2 S=8 B=5 S=7
B=1 S=2 B=6 S=7
THERE ARE MANY MORE!

For example, given [1, 3, 2, 8, 4, 10]