# Practice problem "Best Time to Buy and Sell Stock"

*To learn how to use this template, check out the course "Data Structures and Algorithms in Python".*

## How to run the code and save your work

The recommended way to run this notebook is to click the "Run" button at the top of this page, and select "Run on Binder". This will run the notebook on mybinder.org, a free online service for running Jupyter notebooks.

This tutorial is an executable Jupyter notebook. You can *run* this tutorial and experiment with the code examples in a couple of ways: *using free online resources* (recommended) or *on your computer*.

### Option 1: Running using free online resources (1-click, recommended)

The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. You can also select "Run on Colab" or "Run on Kaggle", but you'll need to create an account on Google Colab or Kaggle to use these platforms.

### Option 2: Running on your computer locally

To run the code on your computer locally, you'll need to set up Python, download the notebook and install the required libraries. We recommend using the Conda distribution of Python. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

### Saving your work

Before staring the assignment, let's save a snapshot of the assignment to your Jovian profile, so that you can access it later, and continue your work.

```
project_name = 'data-structures-course-submit-version' # give it an appropriate name
```

```
!pip install jovian --upgrade --quiet
```

```
import jovian
```

```
jovian.commit(project=project_name)
```

```
[jovian] Updating notebook "megha-goyate/data-structures-course-submit-version" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/data-structures-course-submit-version
```

```
'https://jovian.ai/megha-goyate/data-structures-course-submit-version'
```

## Problem Statement

> Best Time to Buy and Sell Stock You are given an array prices where prices[i] is the price of a given stock on the ith day.

Find the maximum profit you can achieve. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times).

Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Example 1:

Input: prices = [7,1,5,3,6,4] Output: 7 Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4. Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Example 2:

Input: prices = [1,2,3,4,5] Output: 4 Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4. Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are engaging multiple transactions at the same time. You must sell before buying again.

Example 3:

Input: prices = [7,6,4,3,1] Output: 0 Explanation: In this case, no transaction is done, i.e., max profit = 0.

Constraints:

1 <= prices.length <= 3 * 10^4

0 <= prices[i] <= 10^4 Source: https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/solution/

# The Method

Here's the systematic strategy we'll apply for solving problems:

1. State the problem clearly. Identify the input & output formats.
2. Come up with some example inputs & outputs. Try to cover all edge cases.
3. Come up with a correct solution for the problem. State it in plain English.
4. Implement the solution and test it using example inputs. Fix bugs, if any.
5. Analyze the algorithm's complexity and identify inefficiencies, if any.
6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

This approach is explained in detail in Lesson 1 of the course. Let's apply this approach step-by-step.

# Solution

1. State the problem clearly. Identify the input & output formats. While this problem is stated clearly enough, it's always useful to try and express in your own words, in a way that makes it most clear for you.

Problem

The problem input a array stock[i]( the stock of the i th day, the limit maximum count of array is smaller then 3 * 10^4 , and limit maximum value is smaller then 10^4 ). Then outout the maximum profit(inside one day, just can buy or sell one time. ) For example input array stock[7,3,5,2,6], if buy in day two, the price is 3,then sell in day 3,

then re-buy in day 4, the price is 2, then sell in day 5, the price is 6. This will get the maximum profit, (5-3)+(6-2) = 6. so the max profit is 6.

Input

1.stock[i] ( the array of the price in these day)

1.Output

maxcount , (The maximum profit of the buy and sell combination in Stock[i] )

Save and upload your work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/data-structures-course-submit-version" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/data-structures-course-submit-version
```

```
'https://jovian.ai/megha-goyate/data-structures-course-submit-version'
```

## 2. Come up with some example inputs & outputs. Try to cover all edge cases.

Our function should be able to handle any set of valid inputs we pass into it. Here's a list of some possible variations we might encounter:

General case An empty list A list containing just one element The list contains repeating numbers The case just have Downward trend The case just have Upward trend Zero-list The case have mutiple valleys and peaks and first day is higher second day, the last day is low the previous day The day no any increase and decrease We'll express our test cases as dictionaries, to test them easily. Each dictionary will contain 2 keys: input (a dictionary itself containing one key for each argument to the function and output (the expected result from the function).

```
test0 = {
    'input': {
        'stock': [2, 6, 7, 8, 9, 3, 5, 6]
    },
    'output': 10
}
```

```
test1 = {
    'input': {
        'stock': []
    },
    'output':0
}
```

```
test2 = {
    'input': {
        'stock': [6]
    },
    'output':0
}
```

```
test3 = {
    'input': {
        'stock': [3, 6, 6, 6, 8, 8, 5, 6]
    },
    'output':6
}
```

```
test4 = {
    'input': {
        'stock': [7, 5, 4, 2]
    },
    'output':0
}
```

```
test5 = {
    'input': {
        'stock': [2, 4, 5, 7]
    },
    'output':5
}
```

```
test6 = {
    'input': {
        'stock': [0, 0, 0, 0, 0, 0]
    },
    'output':0
}
```

```
test7 = {
    'input': {
        'stock': [ 7, 1, 6, 2, 4, 5 , 3, 6, 3]
    },
    'output':11
}
```

```
test8 = {
    'input': {
        'stock': [ 3, 3, 3, 3, 3, 3]
    },
```

```
    'output':0
}
```

Create one test case for each of the scenarios listed above. We'll store our test cases in an array called `tests`.

```
tests = [test0, test1, test2, test3, test4, test5, test6, test7, test8]
```

## 3. Come up with a correct solution for the problem. State it in plain English.

Our first goal should always be to come up with a correct solution to the problem, which may not necessarily be the most efficient solution. Come with a correct solution and explain it in simple words below:

Brute Force Approach If we want to find out the maximum profit combination inside the stock array.First, the most direct way is using the Brute Force Approach to search all the different profit combination and compare which profit of the combination is maximum profit.

We can using the loop and recursion to achieve the Brute Force.

1.We can input the stock array, and then counting number 0 (because array is using 0 to count the first number.), we can use the counting number to compare all the number inside the array.

2.Then we can using "if" statement to check that is it already check all the number, if yes, just return 0 to end this

3.Setting the variable (maxcount) for using to save the maximum profit

4.Then use the counting number 0 to setting two for loop to campare the two number , start from compare array[0] and array[1], the first number and next number.

5.if the( first number ) smaller then the (next number), this is meaning buy in the day of the (first number), then sell in the day of the (next number) have a profit, so using the (next number) minus the (first number) to calculate the profit.

6.But the array maybe is not just two number(two day), so we need to add the max profit about the array number after the (next number), so we need to using recursion to re-call the function, but given the (next number) + 1 for the counting number, this will make the function start from array(next number +1) to calculate and compare.

7.when we calculate the profit of differnt combination inside the each recursion, we need to find out the most profit one to transfer the result to upper level of recursion ,(to sum up the most profit one in this level of recursion and the profit of upper recursion. So we compare if (profit) > (maxprofit), put the (profit) become the (maxprofit).

8.Finally when the (maxprofit) transfer to the top level, we still need to compare each (maxprofit) is the real maximum profit, first we already set the (maxcount) is 0 in the step 3, so we will using if (maxprofit > maxcount), to compare each (maxprofit), which one is the maximum, then put the number of ( maxprofit) to the (maxcount).

9.Finally return the (maxcount), this is the final maximum profit of the stock array.

For example: [7,1,5,3,6,4] 1. array[0] is 7, array [1] is 1, so 7>1, skip. 2. 7>5,skip,7>3,skip,7>6,skip,7>4,skip 3. 1<5, profit is (5-1) + the maxprofit of the number after 5, so run the recursion, start from 3. 3.1. 3<6 profit is (6-3)+ the maxprofit of the number after 6,so run the recursion, start from 4. 3.1.1. 4 is end the loop, return to upper level. 3.2 3<4 3.2.1 after 4, is the end the loop, return to upper level. 4. 1<3 . . . .

***After calculate all combination start from 1 , then will calculate start from 5, then 3, then 6, then 4. Final compare each (maxprofit) to get a maximum profit (maxcount)

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/data-structures-course-submit-version" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/data-structures-course-submit-version
```

'https://jovian.ai/megha-goyate/data-structures-course-submit-version'

## 4. Implement the solution and test it using example inputs. Fix bugs, if any.

```
def bruteforce_cal(stock,count):
    if (count >= len(stock)):
        return 0
    maxcount = 0
    for i in range(count, len(stock)):
        maxprofit = 0
        for j in range(i+1, len(stock)):
            if (stock[i] < stock[j]):
                profit = bruteforce_cal(stock, j + 1) + stock[j] - stock[i]
                if (profit > maxprofit):
                    maxprofit = profit

        if (maxprofit > maxcount):
            maxcount = maxprofit

    return maxcount
```

```
def stock_finder(stock):
    result = bruteforce_cal(stock,0)
    return result
```

We can test the function by passing the input to it directly or by using the `evaluate_test_case` function from `jovian` .

```
from jovian.pythondsa import evaluate_test_case
```

Evaluate your function against all the test cases together using the `evaluate_test_cases` (plural) function from `jovian` .

```
from jovian.pythondsa import evaluate_test_cases
```

```
results = evaluate_test_cases(stock_finder, tests)
```

```
TEST CASE #0
```

Input:
{'stock': [2, 6, 7, 8, 9, 3, 5, 6]}

Expected Output:
10

Actual Output:
10

Execution Time:
0.085 ms

Test Result:
PASSED

TEST CASE #1

Input:
{'stock': []}

Expected Output:
0

Actual Output:
0

Execution Time:
0.003 ms

Test Result:
PASSED

TEST CASE #2

Input:
{'stock': [6]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.004 ms

Test Result:
PASSED

TEST CASE #3

Input:
{'stock': [3, 6, 6, 6, 8, 8, 5, 6]}

Expected Output:
6

Actual Output:
6

Execution Time:
0.058 ms

Test Result:
PASSED

TEST CASE #4

Input:
{'stock': [7, 5, 4, 2]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.008 ms

Test Result:
PASSED

TEST CASE #5

Input:
{'stock': [2, 4, 5, 7]}

Expected Output:
5

Actual Output:
5

Execution Time:
0.013 ms

Test Result:
PASSED

TEST CASE #6

Input:
{'stock': [0, 0, 0, 0, 0, 0]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.007 ms

Test Result:
PASSED

TEST CASE #7

Input:
{'stock': [7, 1, 6, 2, 4, 5, 3, 6, 3]}

Expected Output:
11

Actual Output:
11

Execution Time:
0.071 ms

Test Result:
PASSED

TEST CASE #8

Input:
{'stock': [3, 3, 3, 3, 3, 3]}

Expected Output:
0

Actual Output:
0

Execution Time:
0.007 ms

Test Result:
PASSED

SUMMARY

TOTAL: 9, PASSED: 9, FAILED: 0

Verify that all the test cases were evaluated. We expect them all to fail, since we haven't implemented the function yet.

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/data-structures-course-submit-version" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/data-structures-course-submit-version
```

```
'https://jovian.ai/megha-goyate/data-structures-course-submit-version'
```

```

```

## 5. Analyze the algorithm's complexity and identify inefficiencies, if any.

# Time complexity is O(n^n).

#Space complexity is O(n).

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/data-structures-course-submit-version" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/data-structures-course-submit-version
```

```
'https://jovian.ai/megha-goyate/data-structures-course-submit-version'
```

## 6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

# effective Way

When we think more about the stock array[day price], given example: [7,1,5,3,6,4] When you polt these number to the graph, you will see different peak and valley. We can realize it, we can just buy in each lowest point (the valley) and sell in hightest point (the peak) sequentially by the Day, then will easily to get a maximum profit

```
#jovian.commit()
```

## 7. Come up with a correct solution for the problem. State it in plain English.

Come with the optimized correct solution and explain it in simple words below:

Depending the effective Way, we can put the step like that:

1.Create a variable called (totalearn) to save the total how much money we earn, initializing to 0 first.

2.Create a for loop, loop from 1 to the number before the length of stock, because this can make we can count from the second number to the end of the stock array.

3.Setting the compare condition using if statement, if the (next number) of the array is bigger then previous one , this meaning next day of the price is highter then yesterday, then we can buy on previous day and sell in that day.

4.So using the array[this day] minus array[this day -1] to calculate the profit, and save to the (totalearn).

5.After looping to the end, we accumulate all the profit to (totalearn).

6.Then return the (totalearn), this is the maximum profit.

For example: [7,1,5,3,6,4]

1. totalearn = 0 1.1 1<7, totalearn still = 0.
2. 5>1, totalearn = 4.
3. 3<5, totalearn still = 4.
4. 6>3, totalearn = 4+3 = 7.
5. 4<6, totalearn still = 7.

Finally, return 7.

7 is maximum profit.

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/data-structures-course-submit-version" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/data-structures-course-submit-version
```

```
'https://jovian.ai/megha-goyate/data-structures-course-submit-version'
```

## 8. Implement the solution and test it using example inputs. Fix bugs, if any.

```
def quicksolution(stock):
    totalearn = 0
    for i in range(1, len(stock)):
        if (stock[i] > stock[i - 1]):
            totalearn += stock[i] - stock[i - 1]

    return totalearn
```

## 9. Analyze the algorithm's complexity and identify inefficiencies, if any.

# Time complexity is O(n).

# Space complexity is O(1).

If you found the problem on an external platform, you can make a submission to test your solution.

Share your approach and start a discussion on the Jovian forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/78

```
jovian.commit()
```

```

```

```

```