Module 7 Homework - Divide and Conquer

We have seen divide-and-conquer applied to design 4 algorithms:

- binary search
- merge sort
- quick sort
- quick select

Here, we apply divide-and-conquer to design a new algorithm that solves a new problem - how much could we have made off Bitcoin?

Description

You are provided with the price of Bitcoin at market opening for each year from 2015-2020 in csv files. Our goal is to figure out the maximum profit we could have made in each year. There are two constraints to this problem:

- We can only buy and sell once
- The sell date must be after the buy date

A brute source solution is provided, but it won't quite work with the CSVs you have been provided. The CSVs give the price of Bitcoin every day, but the brute source algorithm expects a list of *change in value* for each day, relative to the day before. For instance, if the price over one week was:

```
[100, 105, 97, 200, 150]
```

The change in value each day would be

```
[0, 5, -8, 103, -50]
```

Deliverable one - `price_to_profit`

Write a function `price_to_profit` that takes as input the price on a series of days (a list) and returns a list of the change in value each day.

```
>>> x = price_to_profit( [100, 105, 97, 200, 150] )
>>> print(x)
[0, 5, -8, 103, -50]
```

Deliverable two - max_profit

Write a function `max_profit` that uses divide and conquer to find the optimal profit you could have, given a value-per-day input as implemented above.

```
>>> x = price_to_profit( [100, 105, 97, 200, 150] )
>>> y = max_profit(x)
>>> print(y)
103
```

Using the example given, the max profit is trivial - we simply buy at the lowest point and sell at the highest. In general, the lowest point may not come until after the highest, so we need a more clever algorithm than `return max(L) - min(L)`.

Brute Force

A brute force algorithm that solves this problem is provided to help you test your function - we find the maximum profit by calculating every possible sell date for every possible buy date.

```python
def max_profit_brute(L):
    n = len(L)
    max_sum = 0

    # outer loop finds the max profit for each buy day
    for i in range(n):
        total = 0

        # inner loop finds the profit for each sell day
        for j in range(i+1, n):
            total += L[j] # total profit if we sell on day j
            if total > max_sum: max_sum = total

    return max_sum
```

This has a running time of O(n^2):

i	number of calcs
0	n-1
1	n-2
...	...
n-1	1
sum	1/2 * ((n-1)^2 + (n-1))

Divide-and-Conquer

We can solve this problem much more efficiently using divide-and-conquer. If we cut our list in half, the maximum profit comes from a buy-sell pair that is either:

- buy and sell date in the left half
- buy and sell date in the right half
- buy date in the left half, sell date in the right

Some pseudo code to help get started:

```python
def max_profit(L, left, right):  # O(nlogn)
   # base case? Return today's profit

   # find the max profit in the left-hand sublist
   # find the max profit in the right-hand sublist
   # find the max profit that crosses from left to right (requires a separate function)

   # return the best profit - left, right, or crossing

def max_profit_crossing(L, left, right, median):   # O(n)
   # Starting from the median, find the best price moving left

   # starting from just after the median, find the best price moving right

   # return the best profit:
   #   * left of median (inclusive)
   #   * right of median (exclusive)
   #   * buy on the left, sell on the right (does not require a recursive call)
```

Using this approach, we still require O(n) checks to find the maximum profit at each level of recursion, but the number of levels of recursion reduces from O(n) to O(logn).

Submission

At a minimum, submit the following file with the functions noted:

- `max_profit.py`
  - `price_to_profit()`
  - `max_profit()`

Students must submit to Mimir **individually** by the due date (typically, the second Wednesday after this module opens at 11:59 pm EST) to receive credit.

Grading

- 10 - `price_to_profit` (required for some other tests to work)
- 90 - `max_profit`
  - 10 - works on small list
  - 40 - works on 6 years of bitcoin data
  - 40 - works on huge lists (requires O(nlogn) solution)

Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly, and it has resulted in many improvements.