

*Code Optimization Contest*

You will solve a problem (a simple and classic one – may not even be new to some of you). Your solution will also be evaluated both functional, but also from the execution performance perspective.

**Contest rules:**

* *Write your solution in your language of choice (we encourage creativity)*
* *Your job is to implement the endpoint described further in this document.*
* *Deploy your solution in a* [*free OpenShift account*](https://www.openshift.com/products/online/) *or something with similar processing power (this is a restriction meant to ensure fairness, by forcing all competitors to use similar processing power, while keeping it accessible and free to everybody).*
* *Provide the URL of your endpoint to the organizers, until Wednesday, 2018 Nov 21st, EOD! Email addresses to send to:* [Vlad.Stan@endava.com; Daniel.Doboga@endava.com](mailto:Vlad.Stan@endava.com;%20Daniel.Doboga@endava.com)
* *We will call your endpoint during the competition window (Thursday Nov 22nd 15:30 – 17:00 EET) and you can assist live in the call from 16:00 to 17:00 EET.*

**How will the evaluation be done:**

For professional programmers, the problem solving is not a high challenge in itself; so we'll move the focus into a challenge for code optimization.

Your endpoint will be called from a client app made by the organizers. This caller will count the time between the moment when the request is sent to your endpoint and the moment when the response is received.

**Steps:**

1. The call to your endpoint gets measured for duration.
2. The response gets validated for correctness of the solution (any computation mistakes will get your solution disqualified).
3. The winner will be the author of the solution which has passed over step 2 and has obtained the lowest total execution time, measured at step 1.

**The problem**

|  |
| --- |
| Let's assume I'm playing the stock market—buy low, sell high. I'm a day trader, so I want to get in and out of a stock before the day is done, and I want to time my trades so that I make the biggest gain possible.  The market has a rule that won't let me buy and sell in a pair of ticks—I have to wait for at least one tick to go buy. And obviously I can't buy in the future and sell in the past.  So, given a list of stock price ticks for the day, can you tell me what trades I should make to maximize my gain within the constraints of the market? Remember—buy low, sell high, and you can't sell before you buy.  **Input Description**  You'll be given a list of stock prices as a space separated list of two decimal floats (dollars and cents), listed in chronological order.  **Output Description**  Your program should emit the two trades in chronological order—what you think I should buy at and sell at.  **Example 1**  Input:  19.35 19.30 18.88 18.93 18.95 19.03 19.00 18.97 18.97 18.98  Output:  18.88 19.03  **Example 2**  Input:  9.20 8.03 10.02 8.08 8.14 8.10 8.31 8.28 8.35 8.34 8.39 8.45 8.38 8.38 8.32 8.36 8.28 8.28 8.38 8.48 8.49 8.54 8.73 8.72 8.76 8.74 8.87 8.82 8.81 8.82 8.85 8.85 8.86 8.63 8.70 8.68 8.72 8.77 8.69 8.65 8.70 8.98 8.98 8.87 8.71 9.17 9.34 9.28 8.98 9.02 9.16 9.15 9.07 9.14 9.13 9.10 9.16 9.06 9.10 9.15 9.11 8.72 8.86 8.83 8.70 8.69 8.73 8.73 8.67 8.70 8.69 8.81 8.82 8.83 8.91 8.80 8.97 8.86 8.81 8.87 8.82 8.78 8.82 8.77 8.54 8.32 8.33 8.32 8.51 8.53 8.52 8.41 8.55 8.31 8.38 8.34 8.34 8.19 8.17 8.16  Output:  8.03 9.34 |

**Endpoint specification**

* You will receive the input in the form of a zip file, containing multiple PNG files. Each PNG file contains a QR code. Each QR code contains an input for the problem. Below is an example, but the input from the contest will be different and will include much more images in the archive. Your endpoint will need to receive such an archive input as multipart file.

Example of input data:



* Some of the images will include valid input. A valid input is a sequence of valid positive numbers (may be integers or floating point), separated by a single space.
* Some of the images will include processing errors (anything that doesn’t fall in the above definition of a valid input). Inputs that contain errors should be totally skipped from the response.
* Your service must solve the problem, for each input.
* Response that your endpoint should return:

Status code should be 200

Response body format:

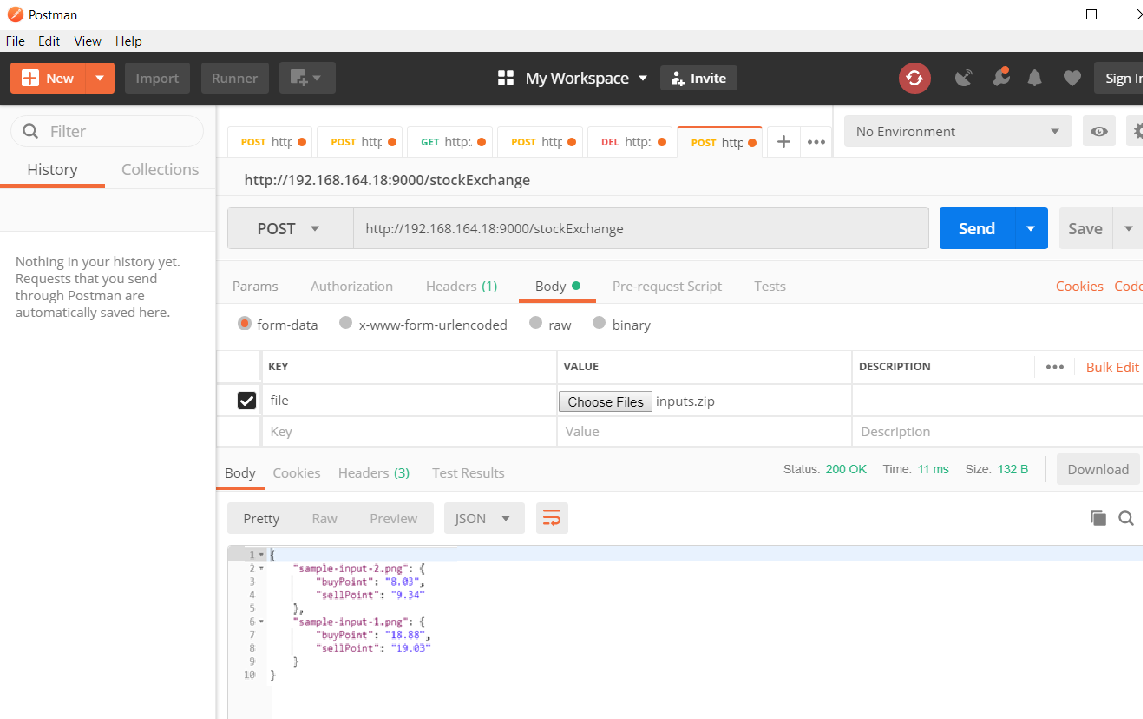
|  |
| --- |
| {  "<qr-image-file-name-1>": {  "buyPoint":"<buy-point-value>",  "sellPoint":"<sell-point-value>"  },  ....................  "<qr-image-file-name-n>": {  "buyPoint":"<buy-point-value>",  "sellPoint":"<sell-point-value>"  }  } |

Example of response body, for the above example of input data:

|  |
| --- |
| {  "sample-input-1.png": {  "buyPoint":"18.88",  "sellPoint":"19.03"  },  "sample-input-2.png": {  "buyPoint":"8.03",  "sellPoint":"9.34"  }  } |

**Additional info**

To be sure that your endpoint is valid, you may take the following as reference for the format. Your endpoint must behave similar, from the format perspective – and when sending the inputs.zip provided above as input, you should receive the same response as below *(please beware the below endpoint is just a mock which woks only with this zip file*):



<http://192.168.164.18:9000/stockExchange/>

The application that we will use to test your endpoint is hosted at the following address and you may use it during development to validate your solution:

<http://192.168.164.18:9000/stockExchangeDemo>

***ENJOY!***