Prerequisites

Basic order book management

Imagine that you are in a market which only trades one type of item (e.g. tomatoes of equal quality; generally known as "shares"). There will be a certain amount of tomatoes (shares) being offered at certain prices. Also, there will be people willing to buy at a certain price. Imagine that everyone who could buy/sell at an acceptable price(limit price) immediately does that and leaves the market. This way (most of the time), nobody can perform a trade right now and everyone has to wait until something changes (e.g. someone reconsiders "acceptable" price, or a new person appears). Those "limit orders" (people willing to buy/sell in certain quantities) are the limit order book. In some cases people are willing to buy/sell at any price (that's called a "market order"), such a person is always going to perform the trade and then leaves the market.

In other words, each price level (for simplicity let's think of it as an integer value) can be either bid (there are people willing to buy at this price), ask (people are willing to sell at this price) or spread (nobody is willing to buy or sell at this price).

Generally, the order book looks like the following example (B - bid, S - spread, A - ask), size defines how many shares can be bought/sold at this price:

| **Price** | **Size** | **Type** | **Comment** |
| --- | --- | --- | --- |
| 99 | 0 | A | Size is zero, but it is still an ask price, because it is higher than the best ask |
| 98 | 50 | A | Best ask (lowest non-empty ask price level) |
| 97 | 0 | S |  |
| 96 | 0 | S |  |
| 95 | 40 | B | Best bid (highest non-empty bid price level) |
| 94 | 30 | B |  |
| 93 | 0 | B |  |
| 92 | 77 | B |  |

The best bid is always lower than the best ask. (for this task it's not important why it is so, but otherwise, those limit orders would execute. We do not ask you to implement this behavior in your solution).

Your Task

In order to complete the task, you will have to create an implementation of an order book. It will be checked for correctness in the following way:  
Initially, you have an empty order book. Then, updates to the book are applied, and the book should respond to queries as described under the "Input/Output data format" section below.

## **Task requirements**

* Create an executable **.jar** file. You can use Java version up to 18 inclusive. Please, be aware that if you use any Java version older than 15, some classes which were included into JRE in previous versions (like **javafx**) are not available in JRE 15. In this case the solution will fail with an exception on the first test case.
* Read the input data from the file named **input.txt** which will be located in the current working directory. Please, access the file by name, e.g. **"input.txt"** not **"/home/user/input.txt"** so it will be resolved to the current working directory automatically.
* As a result of execution, your **.jar** file should create a file containing the output data, named **output.txt** in the same folder (working directory).
* Performance of your solution matters! Note that the memory limit for the tests is set to 128 MB (**-Xmx128M**).
* The solution **.jar** file **must also** contain the source code (files with **.java** or **.kt** extension).
* The solution **.jar** file size must be less than **10 MB**. You are free to include any libraries, just make sure their usage is justified.
* After you have solved the task, send it via the form in the section below (please do not publish it anywhere else). Your submission will be automatically tested, so it is required that you strictly follow the input/output data format. If your **.jar** file is not executable, or does not contain source code, it will not be accepted.
* We value candidates who try to think over the solution before sending it as finished - please, don't send a lot of similar solutions with random changes.

Input/Output Data Format

## **Input file**

Each line in the file can be one of the following:

* Updates to the limit order book in the following format:
  + **u,<price>,<size>,bid** - set bid size at **<price>** to **<size>**
  + **u,<price>,<size>,ask** - set ask size at **<price>** to **<size>**
* Queries in the following format:
  + **q,best\_bid** - print best bid price and size
  + **q,best\_ask** - print best ask price and size
  + **q,size,<price>** - print size at specified price (bid, ask or spread).
* And market orders in the following format:
  + **o,buy,<size>** - removes **<size>** shares out of asks, most cheap ones.
  + **o,sell,<size>** - removes **<size>** shares out of bids, most expensive ones

In case of a buy order this is similar to going to a market (assuming that you want to buy **<size>** similar items there, and that all instances have identical quality, so price is the only factor) - you buy **<size>** units at the cheapest price available.

Queries, market orders, and limit order book updates are in arbitrary sequence. Each line in the file is either one of the three and ends with a UNIX newline character - **\n**.

Input values range:

Price - 1...109

Size - 0...108

Example of input file:

**u,9,1,bid**

**u,11,5,ask**

**q,best\_bid**

**u,10,2,bid**

**q,best\_bid**

**o,sell,1**

**q,size,10**

**u,9,0,bid**

**u,11,0,ask**

## **Output file**

Example of output file (for this input file):

**9,1**

**10,2**

**1**

How to Check Your Solution

Before sending the solution, it is recommended to check how it works on your side. Please, follow the steps below:

1. Create a file **input.txt** with the contents described in the section Input/Output data format .
2. Put the **input.txt** into a folder with **your\_solution.jar** file (the name of the jar file doesn't matter).
3. Open the command line in the same folder. Run **java -Xmx128M -jar your\_solution.jar**. Use your file name instead of "your\_solution".
4. Assuming your solution works correctly, **output.txt** file should appear in the same folder, with the contents identical to what is shown in the Input/Output data format section.

Frequently Asked Questions

### **My solution is rejected**

Please make sure your **.jar** file also contains source files. You can add source files to your **.jar** archive manually if configuring your building tool makes an issue.

### **What if an order book contains both ask and bid shares with an equal price...**

Bids and asks cannot be on the same price level. See the Data validity question.

### **What if an order book contains both ask and bid shares with an equal price...**

Bids and asks cannot be on the same price level. See the Data validity question.

### **What should happen if order BUY command is executed when there are no asks left or SELL command without any active bids available at that moment?**

See the Data validity question

### **What should happen to a market order when the order book does not contain enough shares in it?**

See the Data validity question

### **Test #1 failed**

Test #1 contains the example data from the task description so it should have passed, right? Please check that your solution meets all the task requirements, especially:

* Compiler version
* Correct paths to input/output file. Take into account that your code is executed on another machine, so the paths should not be absolute, and should be cross-platform

### **Some other tests failed. Can you give me some other examples of input data to check my solution?**

Sorry, we do not reveal test cases. The task description tells you, more or less explicitly, everything you need to know to complete the task. You may want to build your own test cases based on the task description.

### **I can’t pass your performance test but I have no OutOfMemoryException on my PC**

Performance is not solely limited to memory usage.

### **Data validity**

The tests are designed to skip the data validity checks so there will be:

* no **best\_bid(ask)** queries to an order book that contains no bids(asks)
* no market order with a size greater than the sum of shares available in the order book.
* no updates that violate the rule of order book correctness - bid levels prices are always lower than ask levels prices. For example, you will not see a sequence of updates like **u,9,1,bid; u,7,1,ask (OR) u,9,5,ask;** because the best bid is at 9, and asks must be always higher in price than bids, so this data is incorrect.
* no incompatible data (negative numbers etc.)

Additional Info

If your solution passes the tests, its execution time will be measured. Think that you can make it better? Well, nothing stops you from sending another solution =)

Please, note that the execution time can fluctuate in the range of +- 300...400 ms for the same solution.

We also have a [leaderboard](https://devtest1.hr.bookmap.com/leaderboard), but it is mostly made for fun, and your rank does not directly affect the interview process or your chances to get hired. Your highest score solution will appear there.

Do you need any extra information to complete this task? If so - feel free to ask, email: [hr@bookmap.com](mailto:hr@bookmap.com)

Send Your Solution

Please, [log in](https://devtest1.hr.bookmap.com/auth?type=login) to send the solution. Don't have an account? [Register here](https://devtest1.hr.bookmap.com/auth?type=register)