D. Сортировка чисел [1 балл]

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Разработчики бэкенда часто взаимодействуют с многочисленными API и дополнительно обрабатывают результаты. Сейчас вам придется сделать именно это!
Во входном файле четыре строчки. В первой находится адрес сервера, во второй — номер порта. В следующих двух строках записаны два целых 32-разрядных числа: a и b. Необходимо осуществить GET-запрос к серверу по указанному номеру порта, передав значения чисел a и b в значениях одноименных CGI-параметров. Сервер ответит JSON-массивом из целых чисел. Необходимо отсортировать числа в порядке неубывания и распечатать в выходной файл — по одному числу в строке.

Гарантируется, что общее количество чисел в ответе не превосходит 100, при этом каждое из них — 32-разрядное знаковое целое число.

Формат ввода

Строка — URL сервера.
Целое число — порт сервера.
Целое число — число a.
Целое число — число b.

Формат вывода

Целые числа, отсортированные по неубыванию, из списка в ответе сервера, по одному в строке.

Пример

Ввод	Вывод
`http://127.0.0.1` `7777`	`-19` `-17`

Ввод	Вывод
	-2 2 4
2 4	6 8 17 256 1024

Примечания

Для решений на языке **python** доступны библиотеки json, requests и urllib.
Для решений на языке **Java** доступна библиотека json-simple версии 1.1.1. Соответствующие import'ы могут выглядеть так:

```
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
```

Для решений на языке **C++** доступны библиотеки libcurl v7.47.0 и nlohmann/json v3.8.0. Соответствующие include выглядят так:

```
#include <curl/curl.h>
#include "json.hpp"
```

Для решений на **golang** доступны все стандартные пакеты, включая encoding/json, net/http, sort и другие.
Для решений на **C#** доступны библиотеки System.Text.Json и Newtonsoft.Json. Соответствующие using могут выглядеть так:

```
using Newtonsoft.Json;
using System.Text.Json;
```

Пример ответа сервера для первого теста:

```
curl "http://127.0.0.1:7777?a=2&b=4"
[
  8,
  6,
  -2,
  2,
  4,
  17,
  256,
  1024,
  -17,
  -19
]
```

E. Ограничение запросов

Язык	Ограничение времени	Ограничение памяти	Ввод	Вывод
Все языки	3 секунды	256Mb	стандартный ввод	стандартный вывод
Golang 1.14.4 + network	2 секунды	256Mb		
Python 3.7 + network + requests	5 секунд	256Mb		
GNU c++ 11 + network + libcurl + json	2 секунды	256Mb		

Слава только-только устроился в Яндекс.Маркет и получил первое задание: написать компонент, отвечающий за ограничение запросов пользователей к сервису.
Характеристики компонента — лимит запросов на пользователя, лимит запросов на сервис и продолжительность интервала (duration), за который учитываются запросы. Запрос к сервису, поступивший в момент времени time, отклоняется, если он удовлетворяет хотя бы одному из двух условий:

1. Если в промежутке времени $[time-duration,time]$ от данного пользователя было принято к выполнению не меньше пользовательского лимита запросов — должен возвращаться код ошибки «Too Many Requests».
2. Если в промежутке времени $[time-duration,time]$ суммарно от всех пользователей было принято к выполнению не меньше сервисного лимита запросов — должен возвращаться код ошибки «Service Unavailable».

Условия проверяются последовательно друг за другом, пока не будет найдено первое условие, которому запрос удовлетворяет. Если запрос не удовлетворяет ни одному условию, то он выполняется. Пока что Слава еще не до конца погрузился в разработку и внутренние технологии, поэтому решил написать прототип, а вы можете помочь Славе.

Формат ввода

Программа получает на вход строку с тремя числами, разделенными пробелами: userLimit ($1 \le userLimit \le 5 \cdot 10^4$) — лимит запросов на пользователя, serviceLimit ($1 \le serviceLimit \le 5 \cdot 10^4$) — лимит запросов на сервис, duration ($1 \le duration \le 10^9$) — промежуток времени в миллисекундах, за который учитываются запросы для расчета лимитов.
В следующих строках поступают описания запросов. Каждый запрос представлен двумя разделенными пробелом числами: time ($1 \le time \le 10^9$) — время поступления запроса, userId ($1 \le userId \le 10^9$) — идентификатор пользователя, выполнившего запрос. Входные данные завершаются строкой с числом -1. Гарантируется, что все времена в описании запросов не убывают, а количество запросов не превышает $5 \cdot 10^4$.
Тестирующая система даст прочитать очередной запрос только после записи в стандартный вывод ответа на предыдущий запрос (не забудьте после вывода запроса сделать `flush`).

Формат вывода

Для каждого запроса необходимо вывести один из трех кодов ответа HTTP:

- 200 — если запрос будет выполнен.
- 429 — если запрос будет отклонен из-за превышения лимита запросов пользователем.
- 503 — если запрос будет отклонен из-за превышения сервисного лимита запросов.

Не забывайте о том, что ваша программа должна сбрасывать буфер вывода после вывода строчки с тайм-аутом. Для сброса буфера вывода можно использовать `fflush(stdout)` в C++, `System.out.flush()` в Java, `stdout.flush()` в Python.

Пример

Ввод	Вывод

```
2 5 5
1 100
1 100
2 100
2 200
2 300
2 400
2 500
3 500
5 200
6 100
7 200
-1
```

C. Опять JSON’ы перекладывать...

Ограничение времени	2 секунды
Ограничение памяти	256.0 Мб
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Всем хотя бы раз в жизни приходилось перекладывать JSON. Вот и для нового проекта под названием "Единое хранилище" необходимо переложить магазинные фиды. Для размещения на Яндекс.Маркете магазины передают товары из своего ассортимент в JSON-файлах. Одно товарное предложение описывается так:

```
{
    "offer_id": <string>,
    "market_sku": <int>,
    "price": <int>
}
```

где *offer_id* - уникальный среди всех фидов магазина идентификатор предложения, *market_sku* - идентификатор товара на Яндекс.Маркете, *price* - стоимость товара.

Весь фид магазина представляет собой JSON и выглядит так:

```
{
    "offers": [<offer>, <offer>, ...]
}
```

Вас попросили написать программу, которая объединит все фиды одного магазина в единый фид и отсортирует товары в порядке неубывания их стоимостей, а при их равенстве - по *offer_id*.

Формат ввода

В первой строке входных данных содержится целое число n - количество фидов магазина ($1 \le n \le 200$). Следующие n строк содержат по одному магазинному фиду на строку. Гарантируется, что строка - валидный JSON и удовлетворяет формату фида. В одном фиде не больше 200 товарных предложений. *offer_id* состоит из строчных и заглавных букв латинского алфавита и цифр, $1 \le |offer\_id| \le 10$, $1 \le market\_sku \le 2^{31}-1$, $1 \le price \le 10^6$.

Формат вывода

Выходной поток должен содержать один JSON-документ, удовлетворяющий формату товарного фида. Количество строк в выходном файле и табуляция не имеют значения.

Пример

Ввод

```
2
{"offers": [{"offer_id": "offer1", "market_sku": 10846332, "price": 1490},
{"offer_id": "offer2", "market_sku": 682644, "price": 499}]}
{"offers": [{"offer_id": "offer3", "market_sku": 832784, "price": 14000}]}
```

Вывод

```
{"offers":[{"market_sku":682644,"offer_id":"offer2","price":499},
{"market_sku":10846332,"offer_id":"offer1","price":1490},
{"market_sku":832784,"offer_id":"offer3","price":14000}]}
```

Примечания

Для решений на языке **python** доступны все стандартные библиотеки, включая json.
Для решений на языке **Java** доступна библиотека json-simple версии 1.1.1. Соответствующие import'ы могут выглядеть так:

```
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
```

Для решений на языке **C++** доступна библиотека nlohmann/json v3.8.0. Соответствующие include выглядят так:

```
#include "json.hpp"
```

Для решений на **golang** доступны все стандартные пакеты, включая encoding/json, sort и другие.

B. Разбиение на интервалы дат

	Все языки	Python 3.7 + network + requests
Ограничение времени	2 секунды	4 секунды
Ограничение памяти	256Mb	244Mb
Ввод	стандартный ввод или input.txt	
Вывод	стандартный вывод или output.txt	

Миша работает в команде Яндекс.Маркета, которая предоставляет производителям товаров аналитику о продажах. Сейчас Миша разбирается с периодизацией: нужно собирать данные по дням, неделям, месяцам, кварталам и годам. От клиентов приходят запросы, в которых указан период детализации и интервал: начальная и конечная даты. Так что первоначально Мише нужно разбить интервал на периоды. Так, если клиент хочет данные с 2020-01-10 по 2020-03-25 с детализацией по месяцам, то ему вернутся данные за три периода: с 2020-01-10 по 2020-01-31, с 2020-02-01 по 2020-02-29 и с 2020-03-01 по 2020-03-25. Помогите Мише, а то ему еще диплом писать надо!

Всего нужно поддержать пять видов временных интервалов:

1. `WEEK` — неделя с понедельника по воскресенье.
2. `MONTH` — месяц.
3. `QUARTER` — интервалы в три месяца: январь — март, апрель — июнь, июль — сентябрь, октябрь — декабрь.
4. `YEAR` — год с 1 января по 31 декабря.
5. `REVIEW` — периоды, за которые оцениваются достижения сотрудников Яндекса. Летний период длится с 1 апреля по 30 сентября, зимний — с 1 октября по 31 марта.

Формат ввода

В первой строке дан типа интервала type — строка, принимающая одно из следующих значений: `WEEK`, `MONTH`, `QUARTER`, `YEAR`, `REVIEW`. Во второй строке через пробел даны начальная и конечная даты start и end (start≤end) в формате `yyyy-MM-dd`. Гарантируется, что обе даты лежат в промежутке с 1 января 2000 года по 31 декабря 3999 года включительно.

Формат вывода

В первой строке ответа выведите одно целое число N — количество промежутков. В последующих N строках на i-й строке выведите через пробел дату начала и конца i-го промежутка в формате `yyyy-MM-dd`. Промежутки должны выводиться в порядке возрастания начальной даты.

Пример 1

Ввод	Вывод
MONTH 2020-01-10 2020-03-25	3 2020-01-10 2020-01-31 2020-02-01 2020-02-29 2020-03-01 2020-03-25

Пример 2

Ввод	Вывод
WEEK 2020-01-26 2020-03-23	10 2020-01-26 2020-01-26 2020-01-27 2020-02-02 2020-02-03 2020-02-09 2020-02-10 2020-02-16 2020-02-17 2020-02-23 2020-02-24 2020-03-01 2020-03-02 2020-03-08 2020-03-09 2020-03-15 2020-03-16 2020-03-22 2020-03-23 2020-03-23

Пример 3

Ввод	Вывод
REVIEW 2016-09-20 2022-11-30	14 2016-09-20 2016-09-30 2016-10-01 2017-03-31 2017-04-01 2017-09-30 2017-10-01 2018-03-31 2018-04-01 2018-09-30 2018-10-01 2019-03-31 2019-04-01 2019-09-30 2019-10-01 2020-03-31 2020-04-01 2020-09-30 2020-10-01 2021-03-31

Ввод	Вывод
	2021-04-01 2021-09-30 2021-10-01 2022-03-31 2022-04-01 2022-09-30 2022-10-01 2022-11-30

YEAR
2020-01-10 2022-03-25