Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer

5. November 2018

Datenstrukturen & Algorithmen — Blatt P7 — HS 18

Please remember the rules of honest conduct:

- Programming exercises are to be solved alone
- Do not copy code from any source
- Do not show your code to others

Hand-in: Sunday, 18. November 2018, 23:59 clock via Online Judge (source code only).
Questions concerning the assignment will be discussed as usual in the forum.

Exercise P7.1 *Bitcoins.*

As an employee of the Federal Tax Administrations office in Switzerland you have been tasked to monitor bitcoin transactions. In particular, you are interested to discover the range of the 1/3 most valuable transactions ever made using bitcoin. The bitcoin network is a distributed database that constantly gets updated, and continuously grows in size. As a result, you need to create a live system that can efficiently consume new transactions, and report the range from the $\lfloor n/3 \rfloor$-th most valuable transaction to the most valuable transaction. Therefore, your system supports two operations:

1. **Insert**. The insertion is done by entering the number 1 on the standard input, followed by a number V ($1 \leq V \leq 10^9$) that indicates the (integer) value of the new transaction. Every time a transaction is added, the number of transactions n in the system is increased by 1.
2. **Report**. The reporting is done by entering the number 2 on the standard input. Then the monitoring system will report the $\lfloor n/3 \rfloor$-th transaction and the first transaction, assuming that all n transactions have been previously sorted in a decreasing order. If $n < 3$ at the time the reporting routine is being invoked, the system will print out the message *Not enough transactions*.

Note that as the monitoring system is live, it is capable of executing both operations in any order (i.e., *insert* and *report* can come one after the other) and the *report* routine can be as frequent as the *insert* routine. Also note that every time the *report* routine is invoked, it will perform an analysis on the transactions already available by the system.

Input The first line of the input consists of the number Q ($1 \leq Q \leq 5 \cdot 10^5$) that indicates the number of routines that will be invoked. Each of the next Q lines contain either an *insert* routine in the form of "`1 V`" or a *report* routine in the form of "`2`" as described above.

Output The output consists of R ($R \leq Q$) lines such that R corresponds to the number of *report* routines present in the input. Each line is either the message *Not enough transactions* or two numbers L and H in the form of "`L - H`" such that L is the $\lfloor n/3 \rfloor$-th most valuable transaction and H is the most valuable transaction present in the system when the *report* routine was invoked. The output is terminated with an end-line character.

Grading You get 3 bonus points if your program works for all inputs. Ideally, your algorithm should require $O(1)$ time for the *report* routine and $O(\log(n))$ for the *insert* routine (with reasonable hidden constants). Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H7P1`. The enrollment password is "`asymptotic`".

Example

Input:

```
12
1 1
1 7
2
1 9
1 8
1 5
1 6
2
1 21
2
1 9
2
```

Output:

```
Not enough transactions
8 - 9
9 - 21
9 - 21
```

A detailed explanation for the 12 routines above:

1. Insert 1 to the array. Current array is `[1]`.
2. Insert 7 to the array. Current array is `[7,1]`.
3. Report. Array size is less than 3. Output is `Not enough transactions`.
4. Insert 9 to the array. Current array is `[9,7,1]`.
5. Insert 8 to the array. Current array is `[9,8,7,1]`.
6. Insert 5 to the array. Current array is `[9,8,7,5,1]`.
7. Insert 6 to the array. Current array is `[9,8,7,6,5,1]`.
8. Report. Array size is 6. $\lfloor n/3 \rfloor = 2$, and the 2-nd element in the sorted array is 8 and highest is 9, therefore $L = 8$ and $H = 9$. Output is `8 - 9`.
9. Insert 21 to the array. Current array is `[21,9,8,7,6,5,1]`.
10. Report. Array size is 7. $\lfloor n/3 \rfloor = 2$, therefore $L = 9$ and $H = 21$. Output is `9 - 21`.
11. Insert 9 to the array. Current array is `[21,9,9,8,7,6,5,1]`.
12. Report. Array size is 8. $\lfloor n/3 \rfloor = 2$, therefore $L = 9$ and $H = 21$. Output is `9 - 21`.

Notes For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H7P1.Bitcoins.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class).

Exercise P7.2 *Mountain Trip.*

A road is n kilometers long and passes through several cities. Each city can be either a mountain city or a sea city. There are M mountain cities, the i-th of which is located m_i kilometers after the beginning of the road. Similarly, there are S sea cities and the i-th sea city is located s_i kilometers after the beginning of the road (m_i and s_i are integers between 0 and n, endpoints included, and each kilometer of the road can traverse at most one city).

A travel agency offers T possible trips. The i-th trip starts from kilometer b_i and ends at kilometer e_i of the road, visiting all the cities in-between (endpoints included). Alice wants to buy a trip that visits the largest number of mountain cities and that does not visit any sea city.

Your task is to design an algorithm that finds the best trip for Alice.

Input The input consists of a set of instances, or *test-cases*, of the previous problem. The first line of the input contains the number C of test-cases, and each test-case consists of 5 lines. The first line of each test-case contains the four integers n, M, S, and T. The second line contains M integers, where the i-th integer is the position m_i of the i-th mountain city. The third line contains S integers, where the i-th integer is the position s_i of the i-th sea city. The fourth line contains T integers, where the i-th integer is the number b_i. Finally, the fifth line also contains T integers, where the i-th integer is the number e_i.

Output The output consists of C lines, where the i-th line is the answer to the i-th test-case and contains the index of the best trip, i.e., an integer t such that $1 \leq t \leq T$ and:

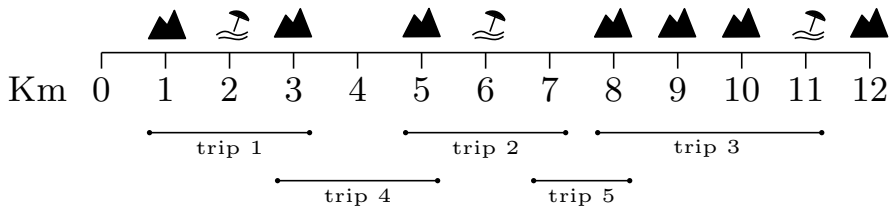
(1) there exists no j such that $b_t \leq s_j \leq e_t$;

(2) for every index $r \neq t$ that satisfies condition (1), $|\{j : b_r \leq m_j \leq e_r\}| < |\{j : b_t \leq m_j \leq e_t\}|$.

You can assume that such an index t always exists.

Grading This exercise awards no bonus points. Your algorithm should require $O((M+S+T)\log(M+S))$ time (with reasonable hidden constants). Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H7P2`. The enrollment password is "`asymptotic`".

Example



Input (corresponding to the instance in the previous picture):

```
1
12 7 3 5
10 8 5 3 9 1 12
```

```
6 2 11
1 5 8 3 7
3 7 11 5 8
```

Output:

```
4
```

Notes For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H7P2.MountainTrip.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class).