Eidgenössische
Technische Hochschule
Zürich

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

Department Informatik
Markus Püschel David Steurer Peter Widmayer
Chih-Hung Liu
Stefano Leucci

6. November 2017

Datenstrukturen & Algorithmen Blatt P7 HS 17

Hand-in: Bis Sonntag, 19. November 2017, 23:59 Uhr via Online Judge (nur Source Code). Fragen zur Aufgabenstellung oder Übersetzung werden wie üblich im Forum beantwortet.

Exercise P7.1 *Inversions.*

Consider an array $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ containing n distinct integers between 0 and 50000. An *inversion* is a pair of indices i, j with $0 \leq i < j < n$ such that $a_i > a_j$. Your task is to design an algorithm that, given A, computes the total number of inversions in A.

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 2 lines. The first line of each test-case contains the integer n. The second line of the test-case contains the n integers $a_0, \ldots, a_{n-1}$ in A, separated by a space.

Output The output consists of C lines, each containing a single integer. The i-th line is the answer to the i-th test-case, i.e., it contains the number of inversions in the corresponding input array A.

Grading You get 3 bonus points if your program works for all inputs. Your algorithm should have an asymptotic time complexity of $O(n \log n)$ with reasonable hidden constants. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=18997&problem=DA17P4.3`. The enrollment password is "`asymptotic`".

Example

Input:

```
2
5
4 6 12 8 20
7
2 12 8 43 1 7 3
```

Output:

```
1
12
```

Notes For this exercise we provide an archive on the lecture website containing a program template that will load the input and write the output for you. 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 `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:

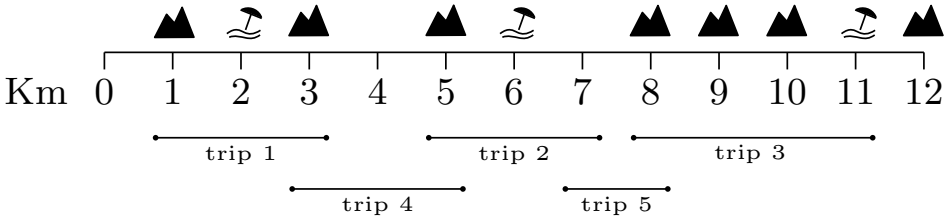
(1) there exists no j such that $b_t \leq s_j \leq e_t$;

(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 You get 3 bonus points if your program works for all inputs. 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=18997&problem=DA17P4.4`. 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 on the lecture website containing a program template that will load the input and write the output for you. 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 `java.util.Scanner` class).