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

06. October 2018

Datenstrukturen & Algorithmen — Blatt P3 — HS 17

Hand-in: Sunday, 21. October 2018, 23:59 clock via Online Judge (source code only).
Questions concerning the assignment will be discussed as usual in the forum.

This homework serve as a warm up practice for the upcoming programming exercises. It's main purpose is to introduce the Judge, the automatic online grading system used to evaluate correctness and timing of each solution. As such, it assumes prior knowledge in variables, methods, loops and conditional statements only. To get started, consider the technical guide that will help you setup Eclipse IDE (Integrated Development Environment), and explain you how to submit your solutions online: `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/technical_guide.pdf`

Exercise P3.1 *Biker.*

Consider a biking route given as a sequence of numbers, each representing the altitude at a given point along the route. In cycling, cumulative elevation gain refers to the sum of every gain in elevation throughout an entire trip. Write a program that finds the lowest altitude along the biker's route, the highest altitude, and the cumulative gain throughout the entire route.

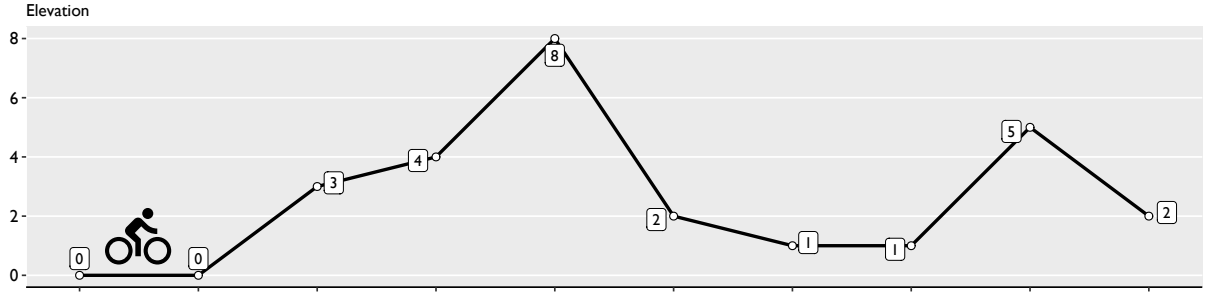

Figure 1: (a) Illustration of a biking route given as altitude points i.e. elevation profile

Input The first line of the input contains the number n representing the number of altitude points given, such $2 \leq n \leq 2^{32} - 1$. Then it is followed by sequence of n numbers in the range $[-2^{32}, 2^{32} - 1]$.

Output The output consists of 3 lines, such that the first one represents the maximal altitude reached along the route, the second one represents the minimal altitude, and the last one represents the elevation gain which will be in the range $[0, 2^{32} - 1]$. The output is terminated with an end-line character.

Grading You get 3 bonus points if your program works for all inputs. Your algorithm should have an asymptotic time complexity of $O(n)$ with reasonable hidden constants. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H3P1`. The enrollment password is "`asymptotic`".

Example

Input:

```
10
0 0 3 4 8 2 1 1 5 2
```

Output:

```
Maximum Height: 8
Minimum Height: 0
Elevation Gain: 12
```

Notes For this exercise we provide an archive on the lecture website, available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H3P1.Biker.zip` 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 P3.2 *Roman Numbers.*

Roman numerals are represented by seven different letters: I, V, X, L, C, D and M. Which represent the numbers 1, 5, 10, 50, 100, 500 and 1,000. These seven letters are used to make thousands of numbers. For example, the Roman numeral for two is written as "II", just two one's added together. The numeral twelve is written as, XII, which is simply X + II. If we take this a step further; the number twenty-seven is written as XXVII, which when broken down looks like XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, this is not always the case. The Romans didn't like four of the same numerals written in a row, so they developed a system of subtraction.

The Roman numeral for three is written as "III", however, the numeral for four is not "IIII". Instead we use the subtractive principle. The number four is written as "IV", the numerals for one and five. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as "IX". There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

The number 1994 is a great example of these rules. It is represented by the Roman numerals MCMXCIV. If we break it down then; M = 1000, CM = 900, XC = 90 and IV = 4.

For a given number n such that $0 < n < 10000$, write a program that outputs it's roman number representation. For numbers larger than 3999, assume that you can prefix as many M as needed. For example if given 7672, output MMMMMMMDCLXXII.

Input The first line of the input contains the number $n < 10000$ that represents the count of the numbers that are about to be converted to roman number representation. Then it is followed by sequence of n numbers in the range $[1, 9999]$.

Output The output contains n lines, each corresponding to the roman number representation of the numbers given in the input sequence. Each roman number representation is defined as a word of the 7 letters: I, V, X, L, C, D and M. The output is terminated with an end-line character.

Grading You get 3 bonus points if your program works for all inputs. Your algorithm should have an asymptotic time complexity of $O(n)$ with reasonable hidden constants. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H3P2`. The enrollment password is "`asymptotic`".

Example

Input:

```
4
5 147 149 1526
```

Output:

```
V
CXLVII
CXLIX
MDXXVI
```

Notes For this exercise we provide an archive on the lecture website, available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H3P2.RomanNumbers.zip` 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).