 Practice   Compete   Jobs   Rank   Leaderboard

Dashboard > Algorithms > Dynamic Programming > Decibinary Numbers

Badge Progress (Details)

Points: 126   Rank: 270330

Decibinary Numbers 🔖

by robinyu

Problem	Submissions	Leaderboard	Discussions	Editorial 🔒

Let's talk about *binary numbers*. We have an n-digit binary number, b, and we denote the digit at index i (zero-indexed from right to left) to be b_i. We can find the *decimal* value of b using the following formula:

$$(b)_2 \Rightarrow b_{n-1} \cdot 2^{n-1} + \ldots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0 = (?)_{10}$$

For example, if binary number $b = 10010$, we compute its decimal value like so:

$$(10010)_2 \Rightarrow 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (18)_{10}$$

Meanwhile, in our well-known decimal number system where each digit ranges from 0 to 9, the value of some decimal number, d, can be expanded in the same way:

$$d = d_{n-1} \cdot 10^{n-1} + \ldots + d_2 \cdot 10^2 + d_1 \cdot 10^1 + d_0 \cdot 10^0$$

Now that we've discussed both systems, let's combine decimal and binary numbers in a new system we call *decibinary*! In this number system, each digit ranges from 0 to 9 (like the decimal number system), but the *place value* of each digit corresponds to the one in the binary number system! For example, the decibinary number 2016 represents the decimal number 24 because:

$$(2016)_{decibinary} \Rightarrow 2 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 6 \cdot 2^0 = (24)_{10}$$

Pretty cool system, right? Unfortunately, there's a problem; two different decibinary numbers could evaluate to the same decimal value! For example, the decibinary number 2008 also evaluates to the decimal value 24:

$$(2008)_{decibinary} \Rightarrow 2 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 8 \cdot 2^0 = (24)_{10}$$

This is a major problem because our new number system has no real applications beyond this challenge!

Consider an infinite list of non-negative decibinary numbers that is sorted according to the following rules:

- The decibinary numbers are sorted in increasing order of the decimal value that they evaluate to.

- Any two decibinary numbers that evaluate to the same decimal value are ordered by increasing decimal value, meaning the equivalent decibinary values are strictly interpreted and compared as decimal values and the smaller decimal value is ordered first. For example, $(2)_{decibinary}$ and $(10)_{decibinary}$ both evaluate to $(2)_{10}$. We would order $(2)_{decibinary}$ before $(10)_{decibinary}$ because $(2)_{10} < (10)_{10}$.

Here is a list of first few decibinary numbers:

x	Decibinary	Decimal
1	0	0
2	1	1
3	2	2
4	10	2
5	3	3
6	11	3
7	4	4
8	12	4
9	20	4
10	100	4
.	...	...
20	110	6

You will be given q queries in the form of an integer, x. For each x, find and print the the xth decibinary number in the list on a new line.

Input Format

The first line contains an integer, q, denoting the number of queries.
Each line i of the q subsequent lines contains a single integer, x, describing a query.

Constraints

- $1 \le q \le 10^5$
- $1 \le x \le 10^{16}$

Subtasks

- $1 \le x \le 50$ for **10%** of the maximum score
- $1 \le x \le 9000$ for **30%** of the maximum score
- $1 \le x \le 10^7$ for **60%** of the maximum score

Output Format

For each query, print a single integer denoting the the xth decibinary number in the list. Note that this must be the actual decibinary number and *not* its decimal value.

Sample Input 0

```
5
1
2
3
4
10
```

Sample Output 0

```
0
1
2
10
100
```

Explanation 0

For each x, we print the xth decibinary number on a new line. See the figure in the problem statement.

f   y   in

Current Buffer (saved locally, editable)   ⑂ ↻   Java 7 ⌄   ⤢ ⚙

```java
1  import java.io.*;
2  import java.util.*;
3  import java.text.*;
4  import java.math.*;
5  import java.util.regex.*;
6
7  public class Solution {
8
9      static long decibinaryNumbers(long x) {
10         // Complete this function
11     }
12
13     public static void main(String[] args) {
14         Scanner in = new Scanner(System.in);
15         int q = in.nextInt();
16         for(int a0 = 0; a0 < q; a0++){
17             long x = in.nextLong();
18             long result = decibinaryNumbers(x);
19             System.out.println(result);
20         }
21         in.close();
22     }
23 }
24
```

Line: 1 Col: 1

⬆ Upload Code as File   ☐ Test against custom input   Run Code   Submit Code