Two non-negative integers are called siblings if they can be obtained from each other by rearranging the digits of their decimal representations. For example, 123 and 213 are siblings. 535 and 355 are also siblings.

A set consisting of a non-negative integer N and all of its siblings is called the family of N. For example, the family of 553 comprises three numbers: 355, 535 and 553.

Write a function: int solution(int N);

that, given a non-negative integer N, returns the largest number in the family of N. The function should return −1 if the result exceeds 100,000,000.

For example, given N = 213 the function should return 321. Given N = 553 the function should return 553.

Assume that:

N is an integer within the range [0..2,147,483,647].

Complexity:

expected worst-case time complexity is O(N);

expected worst-case space complexity is O(N).

Answer:

private static int largestNumber(int data) {

int num = data;

int[] times = new int[10];

while (num != 0) {

if (num == 0) {

break;

}

int val = num % 10;

times[val]++;

num /= 10;

}

String largestNumber = "";

for (int i = 9; i >= 0; i--) {

for (int j = 0; j < times[i]; j++) {

largestNumber += i;

}

}

return Integer.parseInt(largestNumber);

}

// you can also use imports, for example:

// import java.util.\*;

// you can write to stdout for debugging purposes, e.g.

// System.out.println("this is a debug message");

class Solution {

public int solution(int InputTest) {

int SampleNo = InputTest;

int[] b = new int[100];

while (SampleNo != 0) {

if (SampleNo == 0) {

break;

}

if (SampleNo > 100000000){

return -1;

}

if (SampleNo < 0) {

return 0;

}

int val = SampleNo % 10;

b[val]++;

SampleNo /= 10;

}

String Output = "";

for (int i=99; i>= 0; i--) {

for (int a = 0; a < b[i]; a++) {

Output += i;

}

}

try {

Integer.parseInt(Output);

} catch (NumberFormatException e){

return 0;

}

return Integer.parseInt(Output);

}

}