# 6 Advanced Problem: Maximizing Your Salary

We strongly recommend you start solving advanced problems only when you are done with the basic problems (for some advanced problems, algorithms are not covered in the video lectures and require additional ideas to be solved; for some other advanced problems, algorithms are covered in the lectures, but implementing them is a more challenging task than for other problems).

## Problem Introduction


As the last question of a successful interview, your boss gives you a few pieces of paper with numbers on it and asks you to compose a largest number from these numbers. The resulting number is going to be your salary, so you are very much interested in maximizing this number. How can you do this?

In the lectures, we considered the following algorithm for composing the largest number out of the given *single-digit numbers*.

```
LargestNumber(Digits):
answer ← empty string
while Digits is not empty:
  maxDigit ← −∞
  for digit in Digits:
    if digit ≥ maxDigit:
      maxDigit ← digit
  append maxDigit to answer
  remove maxDigit from Digits
return answer
```

Unfortunately, this algorithm works only in case the input consists of single-digit numbers. For example, for an input consisting of two integers 23 and 3 (23 is not a single-digit number!) it returns 233, while the largest number is in fact 323. In other words, using the largest number from the input as the first number *is not a safe move*.

Your goal in this problem is to tweak the above algorithm so that it works not only for single-digit numbers, but for arbitrary positive integers.

## Problem Description

**Task.** Compose the largest number out of a set of integers.

**Input Format.** The first line of the input contains an integer $n$. The second line contains integers $a_1, a_2, \ldots, a_n$.

**Constraints.** $1 \le n \le 100$; $1 \le a_i \le 10^3$ for all $1 \le i \le n$.

**Output Format.** Output the largest number that can be composed out of $a_1, a_2, \ldots, a_n$.

**Time Limits.**

| language | C | C++ | Java | Python | C# | Haskell | JavaScript | Ruby | Scala |
|---|---|---|---|---|---|---|---|---|---|
| time (sec) | 1 | 1 | 1.5 | 5 | 1.5 | 2 | 5 | 5 | 3 |

**Memory Limit.** 512MB.

**Sample 1.**

Input:

```
2
21 2
```

Output:

```
221
```

Explanation:
Note that in this case the above algorithm also returns an incorrect answer 212.

**Sample 2.**

Input:

```
5
9 4 6 1 9
```

Output:

```
99641
```

Explanation:
In this case, the input consists of single-digit numbers only, so the algorithm above computes the right answer.

**Sample 3.**

Input:

```
3
23 39 92
```

Output:

```
923923
```

Explanation:
As a coincidence, for this input the above algorithm produces the right result, though the input does not have any single-digit numbers.

## Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch.

## What To Do

Interestingly, for solving this problem, all you need to do is to replace the check $digit \geq maxDigit$ with a call IsGreaterOrEqual($digit, maxDigit$) for an appropriately implemented function IsGreaterOrEqual. For example, IsGreaterOrEqual(2, 21) should return `True`.

## Need Help?

Ask a question or see the questions asked by other learners at this forum thread.