The idea here is basically implement a String comparator to decide which String should come first during concatenation. Because when you have 2 numbers (let's convert them into String), you'll face only 2 cases:

For example:

**String** s1 = "9";

**String** s2 = "31";

**String** case1 = s1 + s2; // 931

**String** case2 = s2 + s1; // 319

Apparently, case1 is greater than case2 in terms of value.  
So, we should always put s1 in front of s2.

If case1 > case2, place s1 before s2;

If case2 > case1, place s2 before s1.

----------------------------------------------------------------------------------------------------------------

**Input:** [3,30,34,5,9]

**Output:** "9534330"

**Intuition**

To construct the largest number, we want to ensure that the most significant digits are occupied by the largest digits.

**Algorithm**

First, we convert each integer to a string. Then, we sort the array of strings.

While it might be tempting to simply sort the numbers in descending order, this causes problems for sets of numbers with the same leading digit. For example, sorting the problem example in descending order would produce the number 9534303, while the correct answer can be achieved by transposing the 3 and the 30. Therefore, for each pairwise comparison during the sort, we compare the numbers achieved by concatenating the pair in both orders.

Once the array is sorted, the most "signficant" number will be at the front. There is a minor edge case that comes up when the array consists of only zeroes, so if the most significant number is 0, we can simply return 0. Otherwise, we build a string out of the sorted array and return it.

**Complexity Analysis**

* Time complexity: O(*nlgn*)

Although we are doing extra work in our comparator, it is only by a constant factor. Therefore, the overall runtime is dominated by the complexity of sort, which is O(*nlgn*) in Python and Java.

* Space complexity:  O(*n*)

Here, we allocate O(*n*) additional space to store the copy of nums. Although we could do that work in place (if we decide that it is okay to modify nums), we must allocate O(*n*) space for the final return string. Therefore, the overall memory footprint is linear in the length of nums.