Difficulty: 🟥 **Category:** Sorting **Successful Submissions:** 1,499+

Radix Sort

Write a function that takes in an array of non-negative integers and returns a sorted version of that array. Use the Radix Sort algorithm to sort the array.

If you're unfamiliar with Radix Sort, we recommend watching the Conceptual Overview section of this question's video explanation before starting to code.

Sample Input

```
array = [8762, 654, 3008, 345, 87, 65, 234, 12, 2]
```

Sample Output

```
[2, 12, 65, 87, 234, 345, 654, 3008, 8762]
```

Hints

Hint 1

Radix Sort sorts numbers by looking only at one of their digits at a time. It first sorts all of the given numbers by their ones' column, then by their tens' column, then by their hundreds' column, and so on and so forth until they're fully sorted.

Hint 2

Radix Sort uses an intermediary sorting algorithm to sort numbers one digits' column at a time. The goal of Radix Sort is to perform a more efficient sort than popular sorting algorithms like Merge Sort or Quick Sort for inputs that are well suited to be sorted by their individual digits' columns. With this in mind, what intermediary sorting algorithm should we use with Radix Sort? Keep in mind that this sorting algorithm will run multiple times, sorting one digits' column at a time.

Hint 3

The most popular sorting algorithm to use with Radix Sort is Counting Sort. Counting Sort takes advantage of the fact that we know the range of possible values that we need to sort. When sorting numbers, we know that we only need to sort digits, which will always be in the range `0-9`. Therefore, we can count how many times these digits occur and use those counts to populate a new sorted array. We'll perform counting sort multiple times, once for each digits' column that we're sorting, starting with the ones' column. We need to ensure that our counting sort performs a stable sort, so that we don't lose information from previous iterations of sorting. Counting sort runs in `O(n)` time, which means that we might have a much more efficient sorting algorithm if the largest number in our input contains few digits. See the Conceptual Overview section of this question's video explanation for a more in-depth explanation.

Optimal Space & Time Complexity

O(d * (n + b)) time | O(n + b) space - where n is the length of the input array, d is the max number of digits, and b is the base of the numbering system used