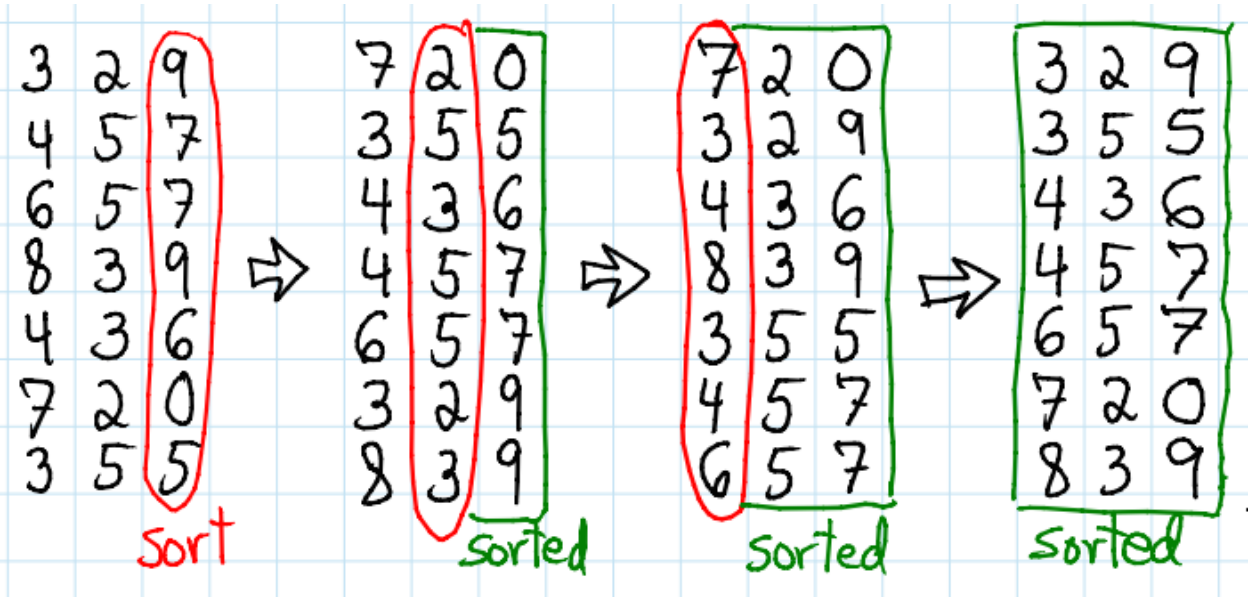
Radix Sort (Non Comparison Sort)

Radix sort works by sorting each digit from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort the input data.

[329, 457, 657, 839, 436, 720, 355]



170, 45, 75, 90, 802, 24, 2, 66

Sorting by first least significant digit (ones place)

17<u>0</u>, 9<u>0</u>, 80<u>2</u>, <u>2</u>, 2<u>4</u>, 4<u>5</u>, 7<u>5</u>, 6<u>6</u>

Sorting by second least significant digit (tens place)

8<u>0</u>2, 2, <u>2</u>4, <u>4</u>5, <u>6</u>6, 1<u>7</u>0, <u>7</u>5, <u>9</u>0

Sorting by third least significant digit (hundreds place)

2, 24, 45, 66, 75, 90, <u>1</u>70, <u>8</u>02

Radix sort is a **stable sort**, which means it preserves the relative order of elements that have the same key value.

To find the maximum number of digits in the given input array, we will need to find out the maximum number in the given input array.

Extracting the individual digits of a number

(802/1) % 10 = 2

(802/10) % 10 = 0

(802/100) % 10 = 8

```
public static void radixsort(int arr[], int n) {
		// Find the maximum number to know number of digits
		int m = getMax(arr, n);

		// Do counting sort for every digit. Note that instead
		// of passing digit number, exp is passed.
		for (int exp = 1; m / exp > 0; exp *= 10) {
		// Call counting sort for each digit, moving from LSD to MSD
			countSort(arr, n, exp);
		}

}
```

```
public static void countSort(int arr[], int n, int exp) {
		int output[] = new int[n]; // output array
		int i;
		int count[] = new int[10];
		Arrays.fill(count, 0);

		// Store count of occurrences in count[]
		for (i = 0; i < n; i++)
			count[(arr[i] / exp) % 10] = count[(arr[i] / exp) % 10] + 1;


		// Change count[i] so that count[i] now contains
		// actual position of this digit in output[]
		for (i = 1; i < 10; i++)
			count[i] = count[i] + count[i - 1];


		// Build the output array
		for (i = n - 1; i >= 0; i--) {
			output[count[(arr[i] / exp) % 10] - 1] = arr[i];
			count[(arr[i] / exp) % 10]--;
		}

		// Copy the output array to arr[], so that arr[] now
		// contains sorted numbers according to current digit
		for (i = 0; i < n; i++)
			arr[i] = output[i];
}
```

Its space requirement is the same as for counting sort, output array of size n and count array of size k , where k is the radix of the number. e.g. k is 10 for decimal digits, k is 26 for alphabet letters and k is 2 for binary bits.

Runtime Analysis

Because radix sort applies counting sort once for each of the p positions of digits in the data, radix sort runs in p times the runtime complexity of counting sort, or O(pn + pk).

e.g. [8, 1, 101, 10, 2147483647]

For the above input data, Radix sort will call counting sort 10 times, so as number of digits increases in the input data, the performance of radix sort will get worse.

So Radix Sort performs well only when number of digits for the input data is small.

Although O(n) is theoretically better than O(n lgn), the radix sort is rarely faster than the O(n lgn) sorting algorithms (merge sort, quick sort, and heap sort). That is because it has a lot of overhead extracting digits and copying arrays.