Radix Sort

The fastest way to sort numbers

What I wanted to show

- “I wrote a faster sorting algorithm”
- https://probablydance.com/2016/12/27/i-wrote-a-faster-sorting-algorithm/
- Claims he can sort twice as fast as sort
- It seems to be a general purpose Radix sort
- I did not have time to explore further with all the time constraints

Instead

- I will show you an efficient Radix sort called
- American Flag Sort
- Which is the basis of his algorithm
- Sorts numbers or keys that are numbers
- It is a non comparative sort, that is it does not compare elements like quick, heap, insertion, merge or the infamous bubble sorts.
- It is straight linear O(n + k)

Not intended to be idiomatically correct, organized to discuss algorithm
I would have written this in C or C++ but decided to make it Perly

```
sub script {
    my (@input) = @_;

    my $len = @input;

    # correlate to the radix
    my $max = 0;
    foreach my $item (@input) {
        $max = $item if $item > $max;
    }
    $len = $max if ($max > $len);

    my @index = ( 0 .. $len);

    print_array ("INDEX", -1, @index);
    print_array ("INPUT", -1, @input);

    my @counts;
    my @offsets;
    my @outputs;

    foreach my $i (@index) {
        $counts [$i] = 0;
        $offsets [$i] = 0;
    }

    foreach my $i (0 .. @input - 1) {
        $outputs [$i] = 0;
    }

    # update the counts, i.e. count how many times for each input
    foreach my $input (@input) {
        $counts[$input] ++;
    }

    # create offsets array
    foreach my $i (1 .. $len - 1) {
        my $sum = 0;
        foreach my $j (0 .. $i - 1) {
            $sum += $counts[$j];
        }

        $offsets [$i] = $sum;
    }
```

We need a number of buckets at least the size of our input, but also at least the size of our largest integer.

3 more sets of buckets, @counts, @offsets and finally @outputs, initialize to all zeros

This trick here allows for efficient layout.   Some Radix sorts make each bucket a linked list of duplicate keys. So add “counts” to “offsets” we have the location of that key in the output.

Now Sort Damn it.

```
    print_array ("COUNTS", -1, @counts);
    print_array ("OFFSETS", -1, @offsets);

    # now proceed

    print "\n";
    foreach my $i ( 0 .. @input - 1) {
        print "\n";
        print '-'x78 . "\n";

        my $item = $input[$i];
        my $idx = $item - 1;

        print "OPERATION BEFORE INDEX $i :$item:\n";
        print_array ("INDEX", $i, @index);
        print_array ("INPUT", $i, @input);
        print_array ("COUNTS", $idx, @counts);
        print_array ("OFFSETS", $idx, @offsets);
        print_array ("OUTPUTS", -1, @outputs);

        # wrote out in excrutiating clarity to explain
        my $place_idx = $counts [$idx];
        $counts [$idx]++;
        $place_idx += $offsets [$idx];

        printf "\n       ITEM %d IDX %d PLACE %d\n\n", $item, $idx, $place_idx;

        $outputs [$place_idx] = $item;

        print "OPERATION AFTER :$item:\n";
        print_array ("INDEX", $i, @index);
        print_array ("INPUT", $i, @input);
        print_array ("COUNTS", $idx, @counts);
        print_array ("OFFSETS", $idx, @offsets);
        print_array ("OUTPUTS", $place_idx, @outputs);
    }

    return 0;
```

print_array is for the pseudo animation

Here is the magic, add the counts to the offsets and you have the location in the output array. Then increment the counts.

I will illustrate now by running it.

....