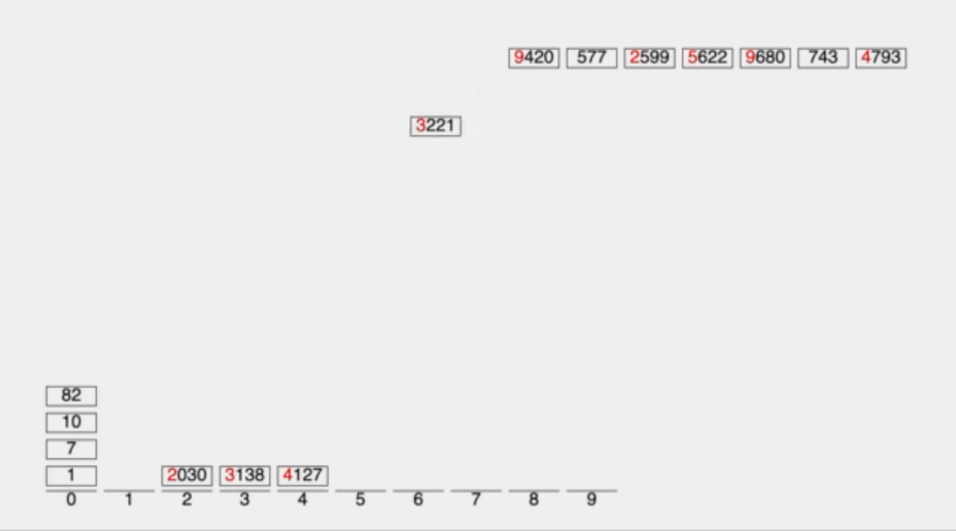
Radix Sort



VisuAlgo https://visualgo.net – You may need to open this in Firefox, Chrome, or Edge to see the video above

Radix sorting enters us into a new frontier of sorting that we haven't talked about: non-comparison based sorting. Up to this point all of the sorts we've talked about have been comparison based sorts. That is to say, we decide the order of the numbers based on asking the question is this element bigger than that one over-and-over again until numbers are in order and the rest of the algorithm is just optimizing how often we ask that question. The big O of these comparison based algorithms cannot be any faster n log n so in order to get beyond that, we have to change what we're doing. We have to sort based on other criteria.

So how do we ask sort a whole list of numbers if we never ask "is X bigger than Y"? Well, we sort *parts* of the array. We're going to sort the ones place first, so all the numbers in the ones place are in order from 0 to 9 e.g. if we had the list of `[109, 224, 901, 58]` after the first was we would have `[901, 224, 58, 109]`. Then we would sort on the tens place, ending up with `[901, 109, 224, 58]`. Then we'd sort by the hundreds, getting `[58, 109, 224, 901]`. And now we're sorted! Magic! It's a bit mind-bending but it does work! And honestly it's that simple.

So what's the mechanism of doing this? Buckets! For base doing a positive integer radix sort (it's possible to do others but let's stick to the easy one for now) we will create ten buckets, one for each integer zero to nine. Then we'll loop `d` times where `d` is the maximum length of a number in our array. So if the longest number in our array is 90932 then `d` would be 5. Then in the inner loop we would enqueue all the numbers in the buckets based on what digit we were sorting and then dequeue them back into the main array. JS doesn't have queue semantics so we'll use push for enqueue and shift for dequeue. We repeat this process until we get through `d` and the array is sorted!

Big O

So, this is way different than what we've been doing. So far we've just had `n` as a variable which represents how many items there are to sort. In radix sort (and other ones too) we have multiple variables. `n` is still important but now we have another variable in play here (usually called `k` or `w`. Let's go with `k` for our purposes.) `k` is going to represents the "maximum key length", or `d` as we referred to it as above. The more buckets we need, the larger the complexity. So intead of being $O(n^2)$ or O(n _ n), it ends up being O(n _ k). So is it better or worse than O(n log n) sorts? It depends! If you have a lot of numbers with lots of varied lengths that will bucket into a good distribution it can be very effective. If you numbers [1, 10, 100, 1000, 10000, 100000] etc it ends up being the worst sort. It ends up being $O(n^2)$ at that point.

What about the spatial complexity? It ends up being O(n + k) and this why radix sort is really only used in very specific circumstances: it's not great in terms of how much space it takes.