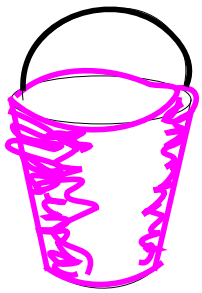
More Sorting

- radix sort
- bucket sort
- in-place sorting
- stable sorting



Radix Sort

- Unlike other sorting methods, radix sort considers the structure of the keys
- Assume keys are represented in a base M number system (M = radix), i.e., if M = 2, the keys are represented in binary

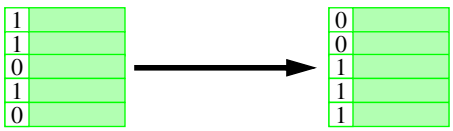
	8	4	2	1	
					weight
9 =	1	0	0	1	($b = 4$)
	3	2	1	0	bit #

- Sorting is done by comparing bits in the same position
- Extension to keys that are alphanumeric strings

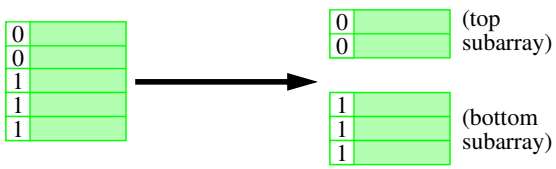
Radix Exchange Sort

Examine bits from ***left*** to ***right***:

1. **Sort array with respect to leftmost bit**



2. **Partition array**



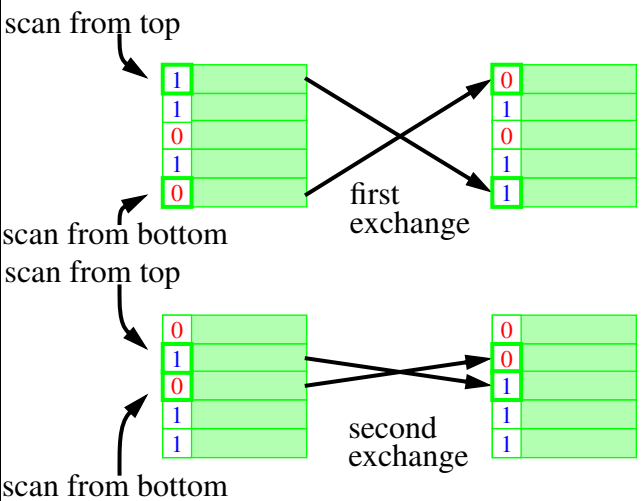
3. **Recursion**
   - recursively sort top subarray, ignoring leftmost bit
   - recursively sort bottom subarray, ignoring leftmost bit

Time to sort n b-bit numbers: $O(b\ n)$

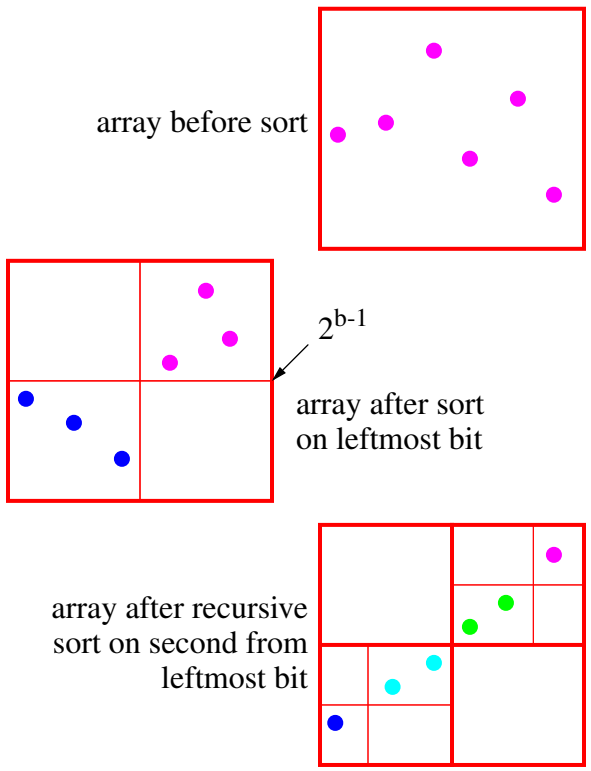
Radix Exchange Sort

How do we do the sort from the previous page?
Same idea as partition in Quicksort.

repeat
  scan top-down to find key starting with 1;
  scan bottom-up to find key starting with 0;
  exchange keys;
until scan indices cross;



Radix Exchange Sort



Radix Exchange Sort vs. Quicksort

Similarities

- both partition array
- both recursively sort sub-arrays

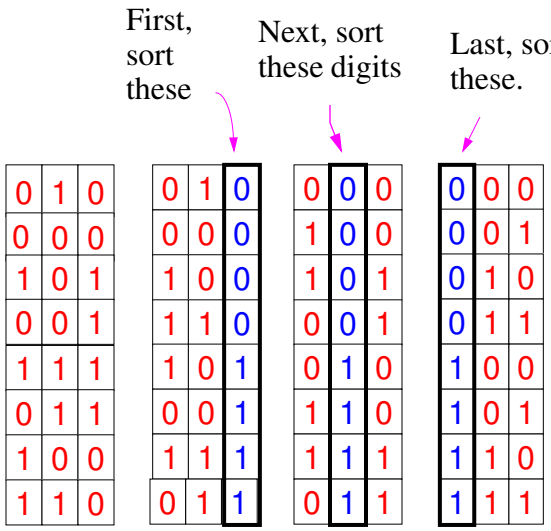
Differences

- ***Method of partitioning***
  - radix exchange divides array based on greater than or less than 2^{b-1}
  - quicksort partitions based on greater than or less than some element of the array
- ***Time complexity***
  - Radix exchange O (bn)
  - Quicksort average case O ($n \log n$)
  - Quicksort worst case O (n^2)

Straight Radix Sort

Examines bits from ***right*** to ***left***

for k := 0 **to** b–1
  sort the array in a ***stable*** way,
  looking only at bit k

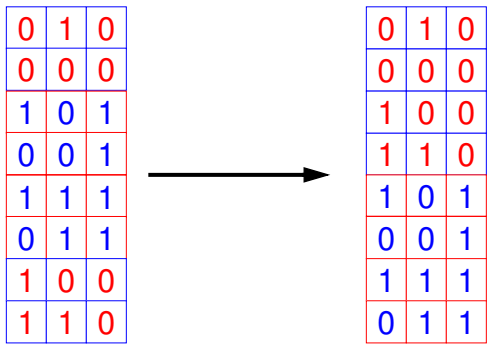


Note order of these bits after sort.

I forgot what it means to "sort in a stable way"!!!

In a stable sort, the initial relative order of equal keys is unchanged.

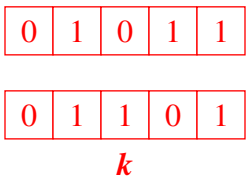
For example, observe the first step of the sort from the previous page:



Note that the relative order of those keys ending with 0 is unchanged, and the same is true for elements ending in 1

The Algorithm is Correct (right?)

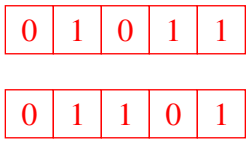
- We show that any two keys are in the correct relative order at the end of the algorithm
- Given two keys, let ***k*** be the leftmost bit-position where they differ

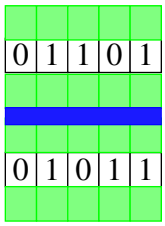


- At step ***k*** the two keys are put in the correct relative order
- Because of ***stability***, the successive steps do not change the relative order of the two keys

For Instance,

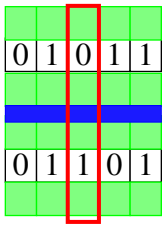
Consider a sort on an array with these two keys:

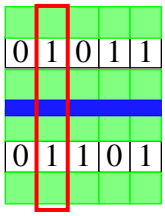




It makes no difference what order they are in when the sort begins.

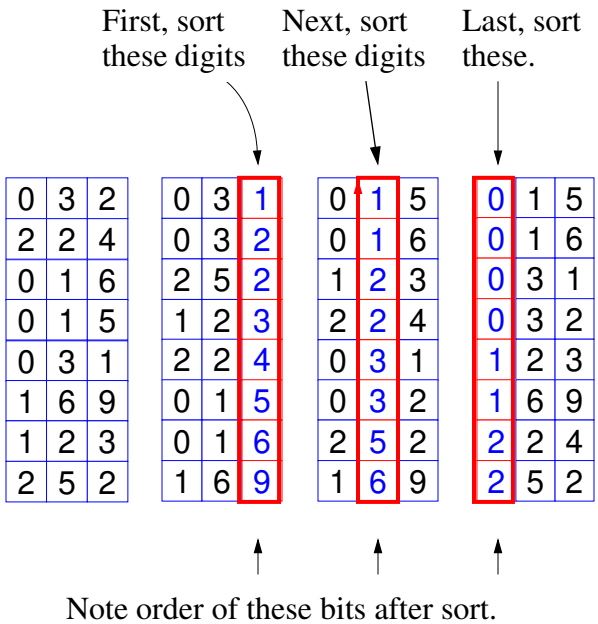
When the sort visits bit ***k***, the keys are put in the correct relative order.





Because the sort is stable, the order of the two keys will not be changed when bits > ***k*** are compared.

Radix sorting can be applied to decimal numbers



Note order of these bits after sort.

Voila!

Straight Radix Sort Time Complexity

for $k = 0$ **to** $b - 1$
  sort the array in a *stable* way,
  looking only at bit k

Suppose we can perform the stable sort above in O(n) time. The total time complexity would be

$$O(bn)$$

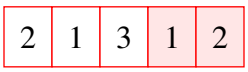
As you might have guessed, we can perform a stable sort based on the keys' k^{th} digit in O(n) time.

The method, you ask? Why it's Bucket Sort, of course.

Bucket Sort

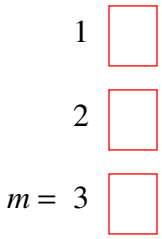
- n numbers
- Each number ∈ {1, 2, 3, ... m}
- Stable
- Time: O ($n + m$)

For example, $m = 3$ and our array is:



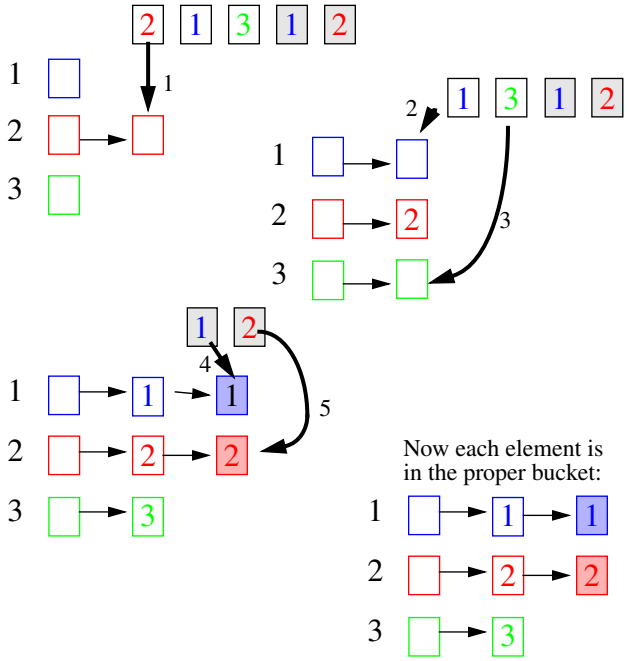
(note that there are two "2"s and two "1"s)

First, we create M "buckets"



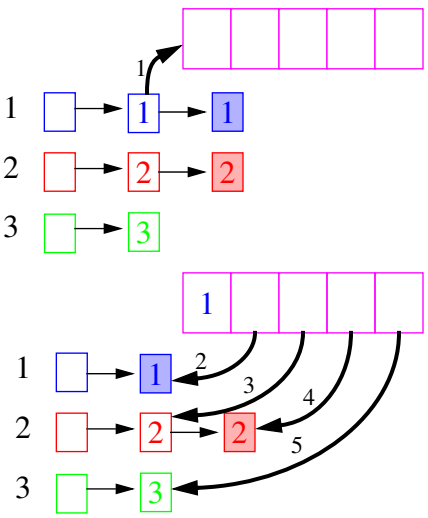
Bucket Sort

Each element of the array is put in one of the m "buckets"

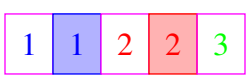


Bucket Sort

Now, pull the elements from the buckets into the array



At last, the sorted array (sorted in a ***stable*** way):



In-Place Sorting

- A sorting algorithm is said to be ***in-place*** if
  - it does not employ auxiliary data structures (however, O(1) auxiliary variables are allowed)
  - it updates the input sequence only by means of operations replaceElement and swapElements
- Which sorting algorithms seen so far can be made to work in place?