Radix Sort

- A key can be thought of as a string of character from some alphabet.
- If the keys are integers stored in *base* d, the alphabet is $\{0, 1, \ldots, d-1\}$.
  - In base 10, the alphabet is
  $$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$
  - In base 2, the alphabet is $\{0, 1\}$.
- **Radix sort** sorts the keys according to one digit at a time.
- This is best seen with an example:

123		581		123		123
435		123		435		246
396		763		945		257
945		394		246		394
257	$\Rightarrow$	435	$\Rightarrow$	257	$\Rightarrow$	396
394		945		763		435
246		396		581		581
581		246		394		763
763		257		396		945
		$\uparrow$ (last digit)		$\uparrow$ (middle digit)		$\uparrow$ (first digit)

Radix Sort Details

- How and why did the example work?
- The sort must start with the *least* significant digit first. Why?
- We must use a *stable* sort. Why?
- Assume
  - The keys are (at most) d digits.
  - Each key is stored in an array of size d
  - The least significant digit is first.
- Here is the shell of the algorithm:

```
  RadixSort(int **A, int n,int d) {
     for(i=1;i<d;i++)
         StableSortColumn(A,n,i);
}
```

- The routine **StableSortColumn(**A**,**n**,**i**)** does a stable sort of the array A based on the values in column i.

Radix Sort Example 2

- The following list has binary keys, written with least significant bit on the right. We sort it with Radix sort, using *naive sort* on each column.

010		010		000		000
000		000		100		001
101		100		101		010
001	$\Rightarrow$	110	$\Rightarrow$	001	$\Rightarrow$	011
111		101		010		100
011		001		110		101
100		111		111		110
110		011		011		111
		$\uparrow$ (right bit)		$\uparrow$ (middle bit)		$\uparrow$ (left bit)

The Correctness of Radix Sort

We show that any two keys are in the correct relative order at the end of the algorithm

Proof:

- Assume the keys are stored with least significant bit on the right.
- Given two keys, let k be the leftmost bit-position where they differ:

0	1	0	1	1

0	1	1	0	1

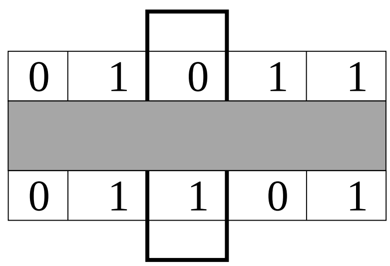
k

- At step k the two keys are put in the correct relative order
- The relative order of the two keys does not change, since they have the same key in the rest of the columns, which we stable sort.

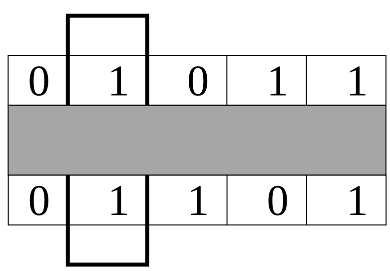
Illustration of Correctness

- Consider a sort on an array with these two keys (It makes no difference what order they are in when the sort begin):

- When the sort visits the kth column, 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



Time Complexity of Radix Sort

- Let b be the length of the keys.
- Let $O(T(n))$ be the complexity of the stable sort we use.
- Then it is clear that the complexity of **Radix Sort** is $O(b \times T(n))$.
- Can **Radix Sort** beat the other sorts in practice? What assumptions do you have to make?
- A sort called **Counting Sort** can be very useful as the stable sort within **Radix Sort**.