Data Structures and Algorithms

(CS210A)

Lecture 39

- Integer sorting continued
- Search data structure for integers : Hashing

Types of sorting algorithms

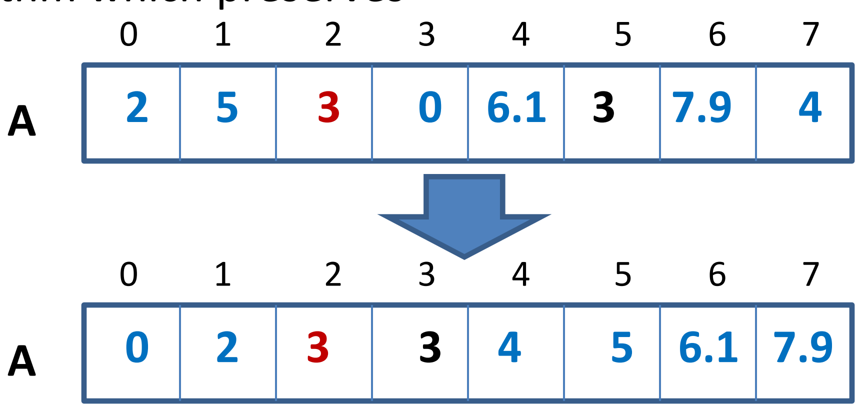
In Place Sorting algorithm:

A sorting algorithm which uses

Example: Heap sort, Quick sort.

Stable Sorting algorithm:

A sorting algorithm which preserves



Example: Merge sort.

Integer Sorting algorithms

Continued from last class

Counting sort: algorithm for sorting integers

Input: An array **A** storing n integers in the range $[0 \ldots k-1]$.

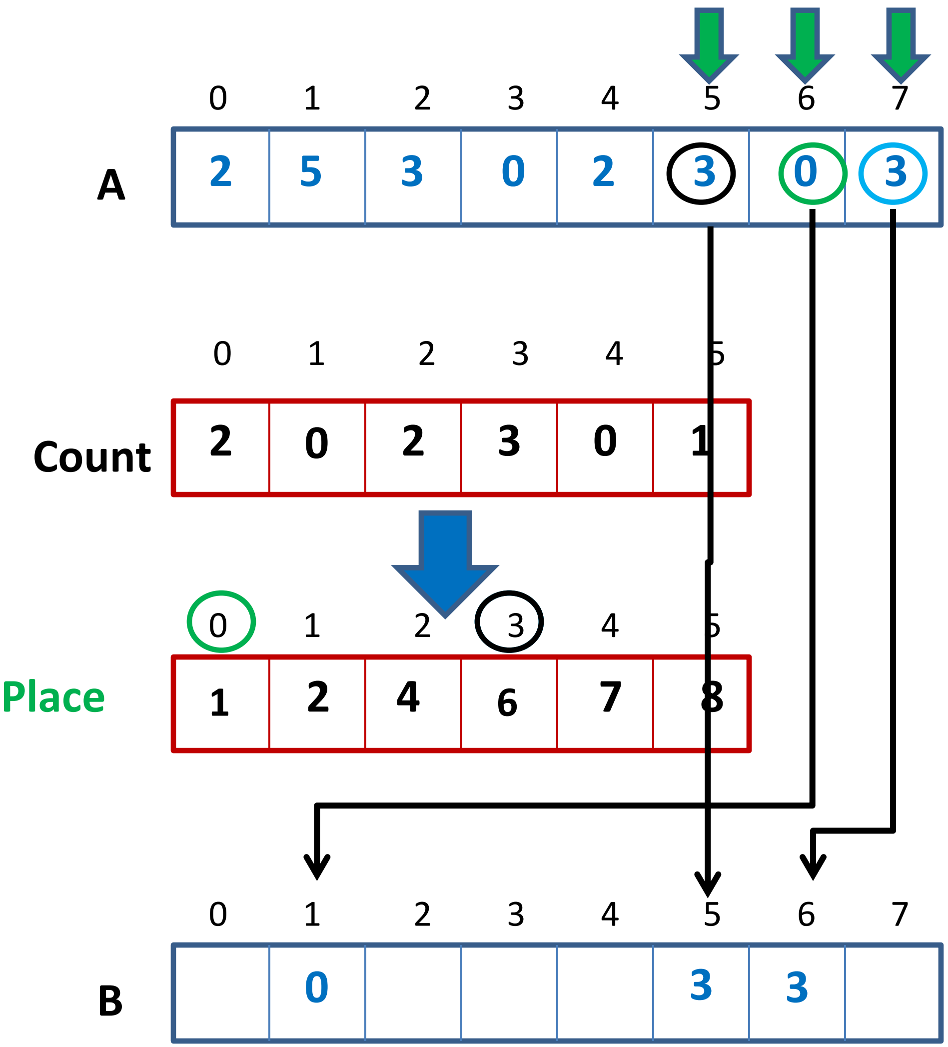
k = O(n)

Output: Sorted array **A**.

Running time: O($n+k$) in **word RAM** model of computation.

Extra space: O($n+k$)

Counting sort: a visual description



Why did we scan elements of **A** in reverse order (from index $n-1$ to 0) while placing them in the final sorted array **B** ?

Answer:

- To ensure that Counting sort is **stable**.
- The reason why stability is required will
- become clear soon ☺

Counting sort: algorithm for sorting integers

Algorithm (A[$0...n-1$], k)

For j=0 **to** $k-1$ **do** Count[j]← 0;

For i=0 **to** $n-1$ **do** Count[A[i]]← Count[A[i]] +1;

Place[0]← Count[0];

For j=1 **to** $k-1$ **do** Place[j]← Place[$j-1$] + Count[j] ;

For i=$n-1$ to 0 **do**

{ B[Place[A[i]]-1]← A[i];

Place[A[i]] ← Place[A[i]]-1;

}

return B;

Each arithmetic operations

involves O(**log** n + **log** k) bits

Counting sort: algorithm for sorting integers

Key points of Counting sort:

- It performs arithmetic operations involving O(**log** n + **log** k) bits
  (O(1) time in **word RAM**).
- It is a **stable** sorting algorithm.

Theorem: An array storing n integers in the range $[0..k-1]$
can be sorted in O(n+k) time and
using total O(n+k) space in **word RAM** model.

➔ **For $k \le n$,**

➔ For $k = n^t$,

(too bad for $t > 1$. ☹)

Question:

How to sort n integers in the range $[0..n^t]$ in

Radix Sort

Digits of an integer

$$507266$$

No. of **digits** = 6

value of **digit** $\in \{0, \ldots, 9\}$

$$\underbrace{1011}\underbrace{0001}\underbrace{0101}\underbrace{1111}$$

No. of **digits** = 4

value of **digit** $\in \{0, \ldots, 15\}$

It is up to us how we define digit ?

Radix Sort

Input: An array **A** storing n integers, where

(i) each integer has exactly d **digits**.

(ii) each **digit** has **value** $< k$

(iii) $k < n$.

Output: Sorted array **A**.

Running time:

O(dn) in **word RAM** model of computation.

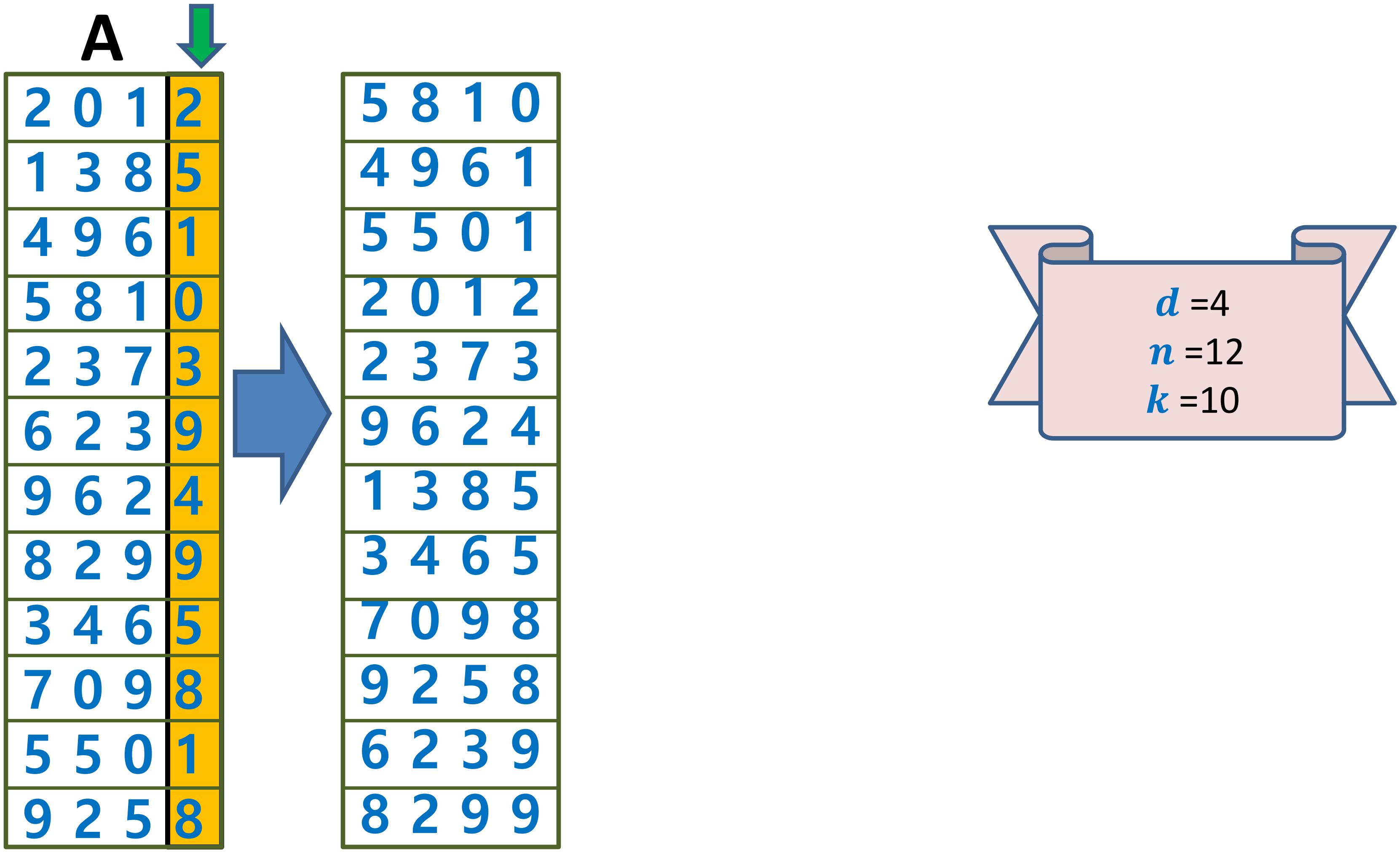
Extra space:

O($n + k$)

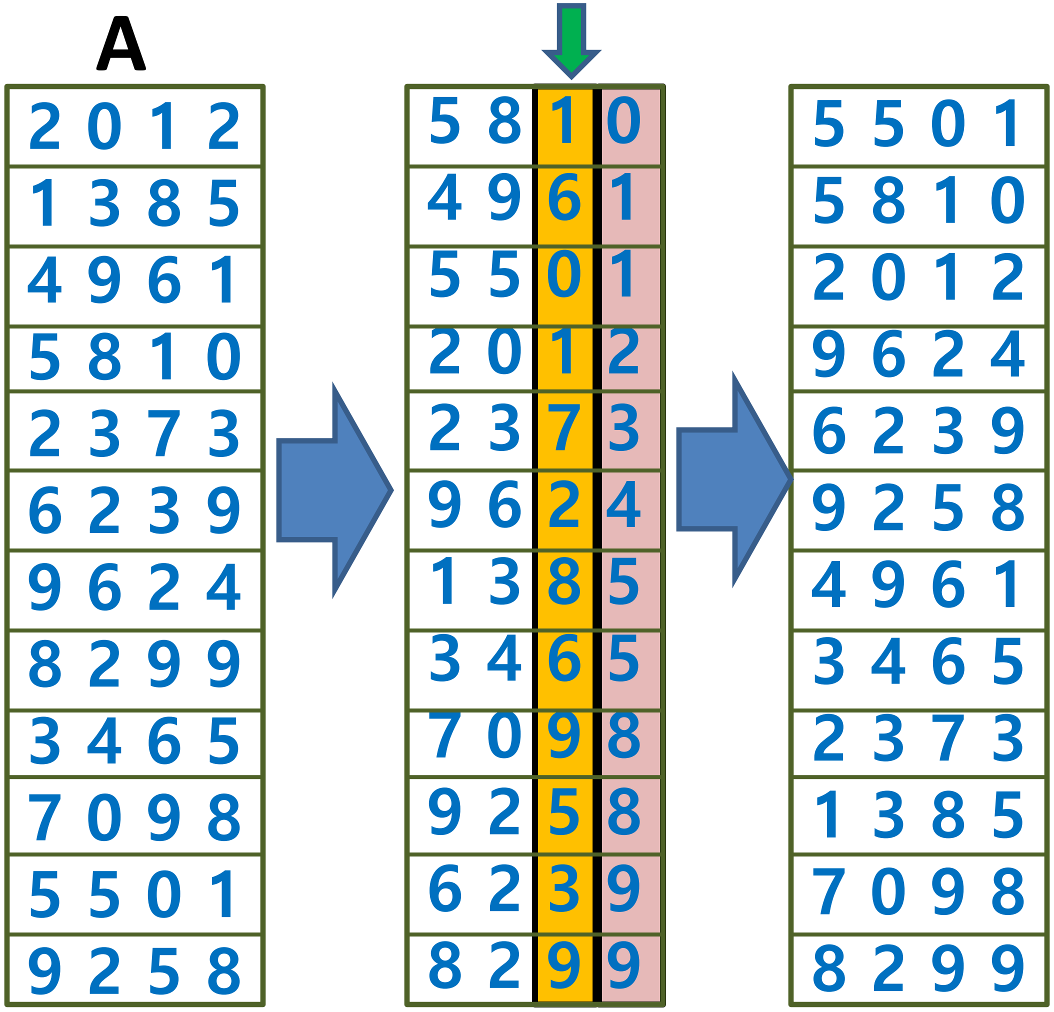
Important points:

- makes use of a **count sort.**
- Heavily relies on the fact that **count sort** is a **stable sort** algorithm.

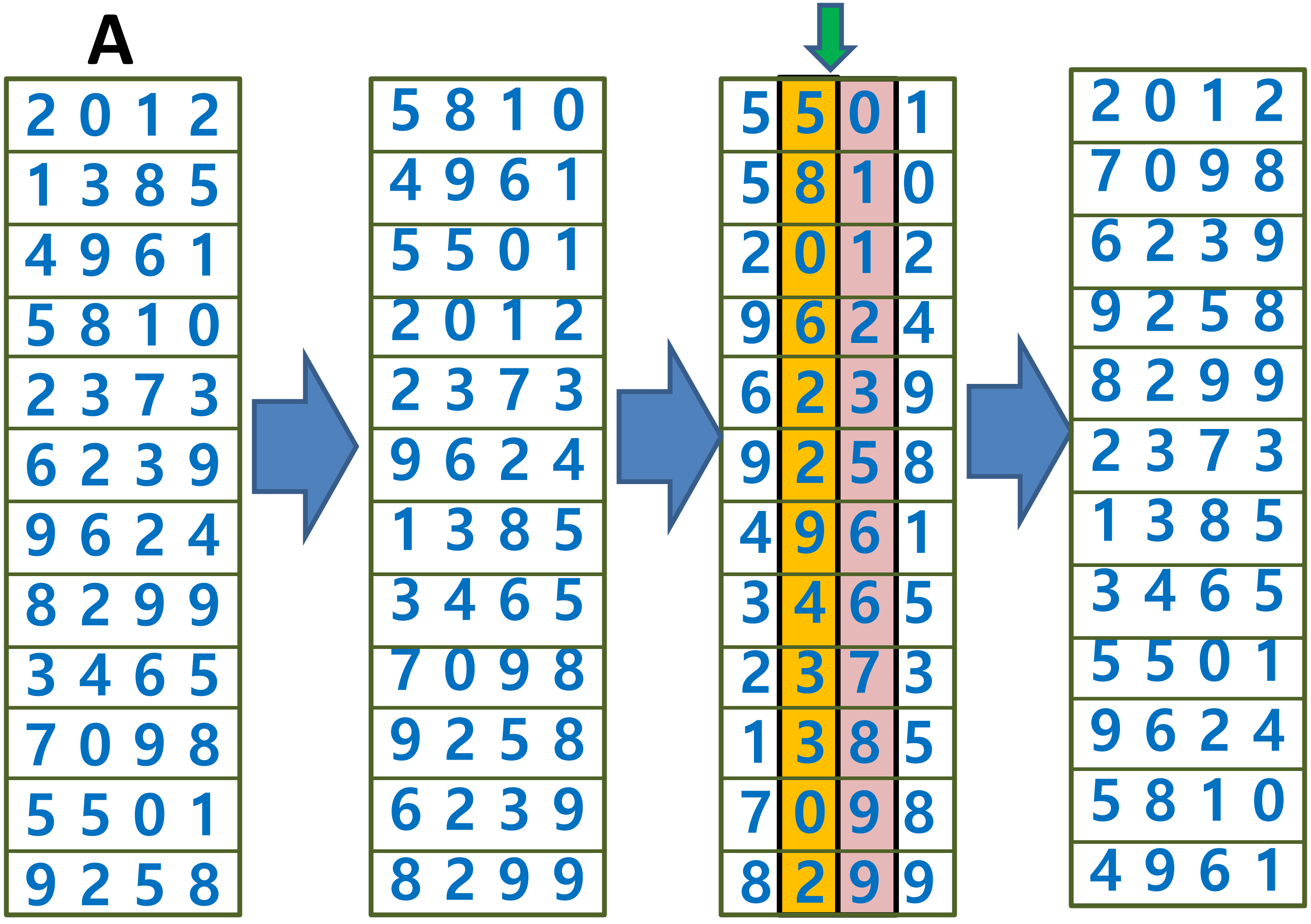
Demonstration of Radix Sort through example



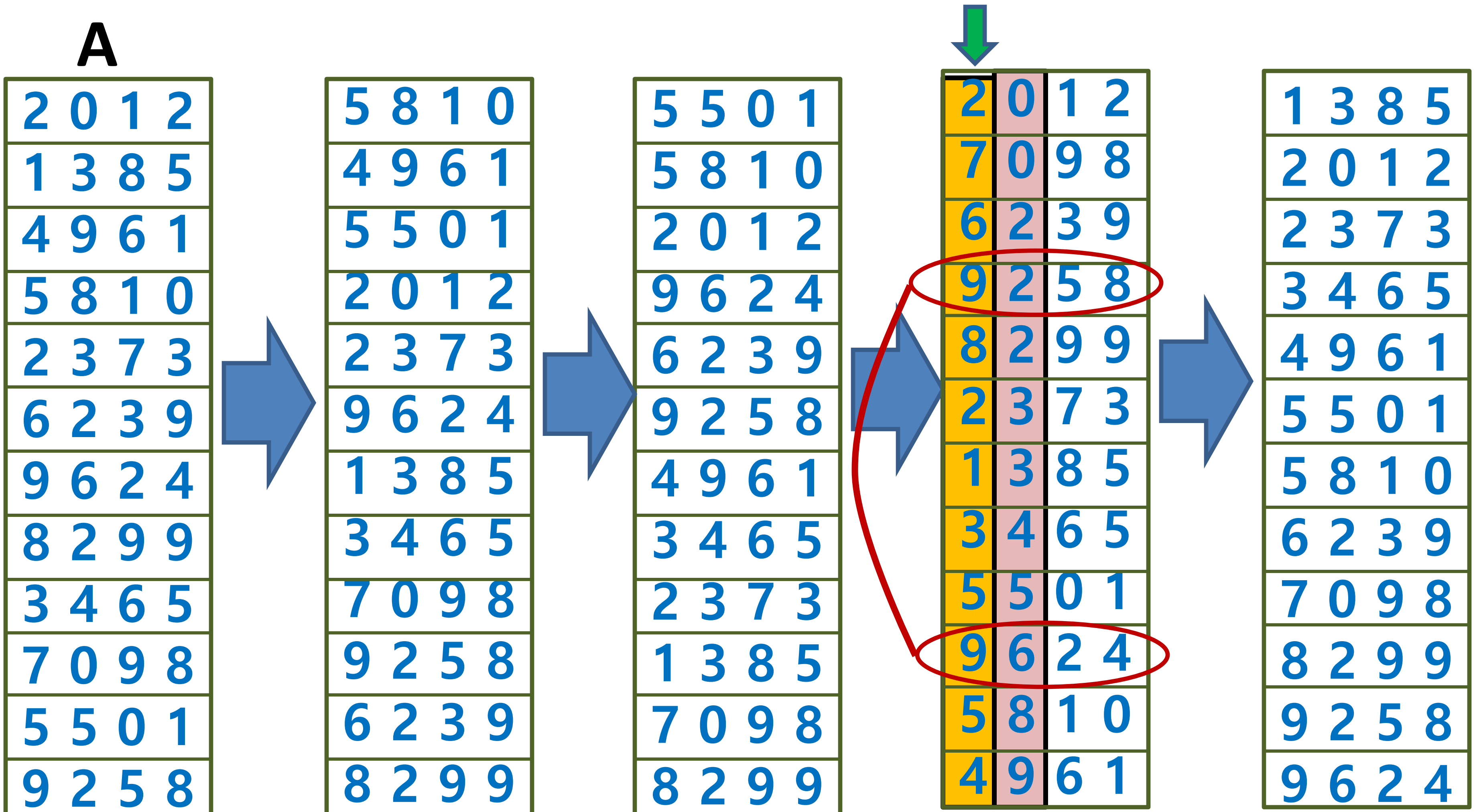
Demonstration of Radix Sort through example



Demonstration of Radix Sort through example



Demonstration of Radix Sort through example



Can you see where we are exploiting the fact that **Countsort** is a **stable** sorting algorithm ?

Radix Sort

RadixSort(**A**[$0 \ldots n-1$], d, k)

{ **For** j=1 **to** d **do**

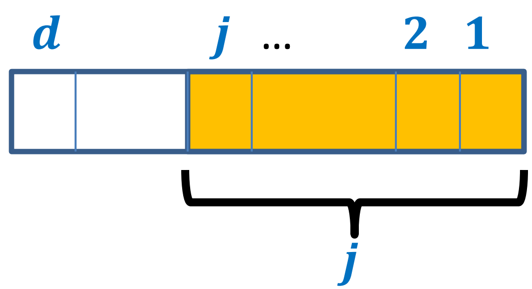
Execute CountSort(**A**,k) with j**th digit** as the **key**;

return **A**;

}

Correctness:

A number stored in **A** ➔



Inductive assertion:

At the end of j**th** iteration, array **A** is sorted according to the last j digits.

During the induction step, you will have to use the fact that **Countsort** is a **stable** sorting algorithm.

Radix Sort

RadixSort(**A**[$0 \ldots n-1$], d, k)

{ **For** j=1 **to** d **do**

  Execute CountSort(**A**,k) with j**th digit** as the **key;**

 return **A;**

}

Time complexity:

- A single execution of **CountSort**(**A**,k) runs in **O**($n+k$) **time** and **O**($n+k$) **space**.
- For $k<n$,

  ➔ a single execution of **CountSort**(**A**,k) runs in **O**(n) time.

  ➔ Time complexity of radix sort = **O**(dn).
- ➔Extra space used = **O**(n)

Question: How to use Radix sort to sort n integers in range $[0..n^t]$ in **O**(tn) time and **O**(n) space ?

Answer:

	d	k	Time complexity
1 bit	$t \log n$	2	**O**($tn \log n$) ☹
$\log n$ bits	t	n	**O**(tn) ☺

What digit to use ?

Power of the word RAM model

- **Very fast** algorithms for **sorting integers:**

  Example: n integers in range $[0..n^{10}]$ in **O**(n) time and **O**(n) space ?

- **Lesson:**

  Do not always go after **Merge sort** and **Quick sort** when input is integers.

- **Interesting programming exercise** (for summer vacation)**:**

  Compare Quick sort with **Radix sort** for sorting **long** integers.