

COL106: Data Structures and Algorithms

Lecture 8

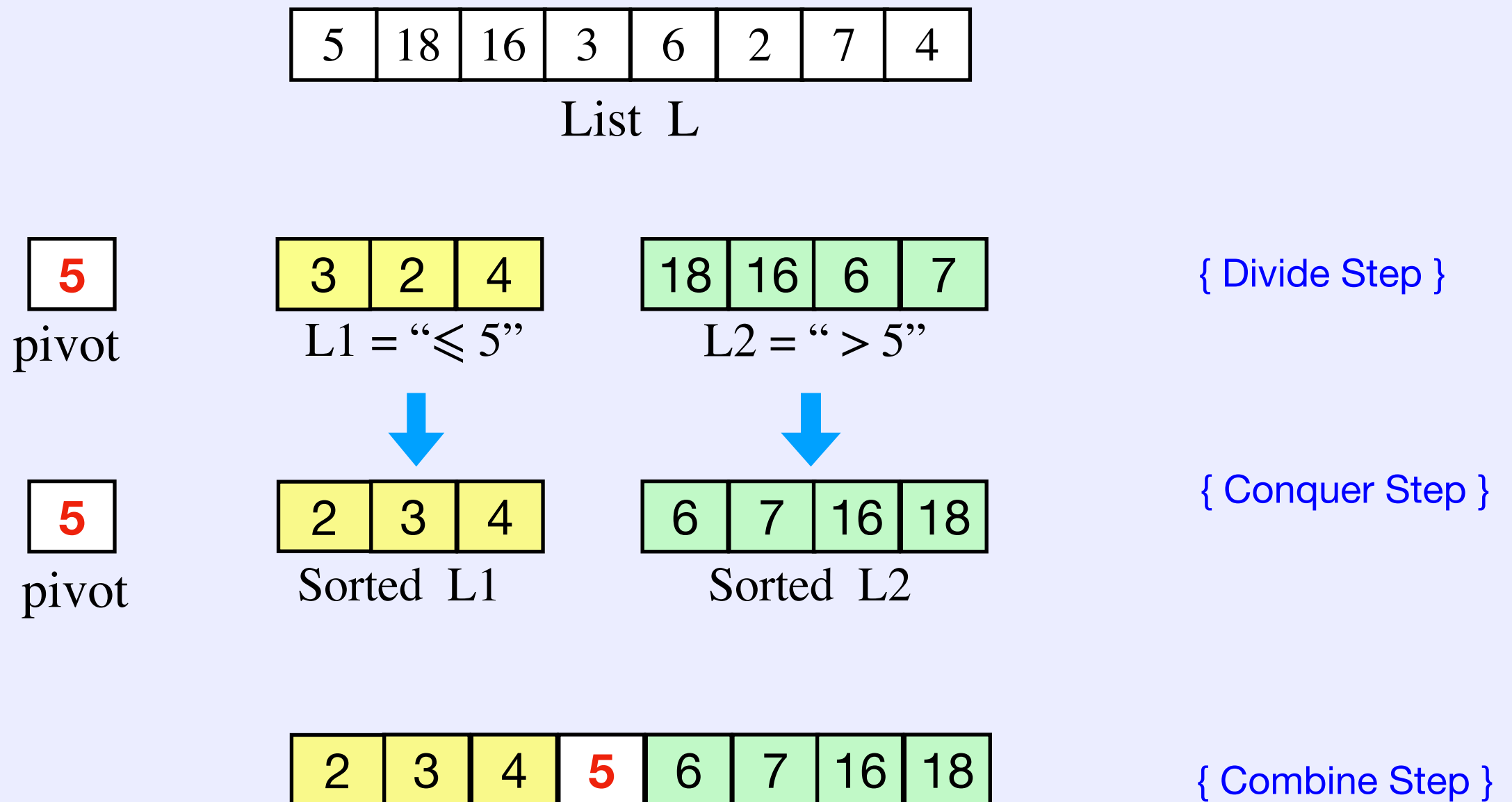
Keerti Choudhary
Department of Computer Science, IIT Delhi

Today's Lecture

1. Quick Sort Analysis
2. Quick Sort implementation using only $O(1)$ extra space
3. Bucket Sort, Count Sort
4. Radix Sort

Quick Sort

A sorting algorithm based on divide-and-conquer approach:



Quick Sort

QUICKSORT(L):

pivot = Random element of list L

$L_1, L_2 \leftarrow$ empty lists

For each ($x \in L \setminus \textit{pivot}$) do

 If ($x \leq \textit{pivot}$) : $L_1.\text{append}(x)$

 If ($x > \textit{pivot}$) : $L_2.\text{append}(x)$

Return QUICKSORT(L_1) $\circ$ *pivot* $\circ$ QUICKSORT(L_2)

Analysis

Let $T(n)$ = expected number of steps taken by the algorithm. Then,

$$T(n) = \frac{1}{n} \sum_{i=1}^n \left(T(i-1) + T(n-i) \right) + cn$$

Probability that pivot is
 i^{th} smallest element

Expected Time to
sort L_1 when its
size is $i-1$

Expected Time to
sort L_2 when its
size is $n-i$

Time to perform
split and merge
operations

Lower Bound

Let $T(n)$ = expected number of steps taken by the algorithm. Then,

$$\begin{aligned} T(n) &= \frac{1}{n} \sum_{i=1}^n \left(T(i) + T(n-i) \right) + cn \\ &\geq \frac{1}{n} \sum_{i=1}^n \left(T(n/2) + T(n/2) \right) + cn \\ &= 2T(n/2) + cn \end{aligned}$$

This implies, $T(n) \geq cn \log_2 n$

Upper Bound

Let $T(n)$ = expected number of steps taken by the algorithm. Then,

$$T(n) = \frac{1}{n} \sum_{i=1}^n \left(T(i-1) + T(n-i) \right) + cn$$

Probability that pivot is
 i^{th} smallest element

Expected Time to
sort L_1 when its
size is $i-1$

Expected Time to
sort L_2 when its
size is $n-i$

Time to perform
split and merge
operations

We have,

$$T(n) = \frac{2}{n} \left(T(1) + T(2) + \dots + T(n-1) \right) + cn$$

Let $T(n)$ = expected number of steps taken by the quick-sort algorithm.

We will prove using induction that $T(k) \leq 4c k \log k$ (where log is taken base 2).

$$T(n) = \frac{2}{n} \left(T(1) + T(2) + \cdots + T(n-1) \right) + cn$$

$$T(n) \leq \frac{2}{n} \left(\sum_{k=1}^{n-1} 4c k \log k \right) + cn$$

$$T(n) \leq \frac{2}{n} \left(\sum_{k=1}^{n/2} 4c k \log(n/2) + \sum_{k=n/2+1}^{n-1} 4c k \log n \right) + cn$$

$$T(n) \leq \frac{2}{n} \left(\sum_{k=1}^{n-1} 4c k \log n - \sum_{k=1}^{n/2} 4c k \log(2) \right) + cn$$

$$T(n) \leq \frac{2}{n} \left(4c \frac{(n-1)n}{2} \log n - 4c \frac{(n/2)(n/2+1)}{2} \right) + cn \leq 4c n \log n.$$

Quick Sort: In place sorting

QuickSort(*A*, start, end):

*/*Sort sub-array A[start, end] */*

if (start < end) **do**

$k = \mathbf{PARTITION}(A, \text{start}, \text{end})$

QuickSort(*A*, start, $k - 1$)

QuickSort(*A*, $k + 1$, end)

PARTITION (*A*, start, end)

pivot = *A*[start]

$i = \text{start}$, $j = \text{end}$

While ($i \leq j$) **do**

if $A[i] \leq \mathbf{pivot}$ **do** (increment i by 1)

else if $A[j] > \mathbf{pivot}$ **do** (decrement j by 1)

else do

swap $A[i]$ and $A[j]$

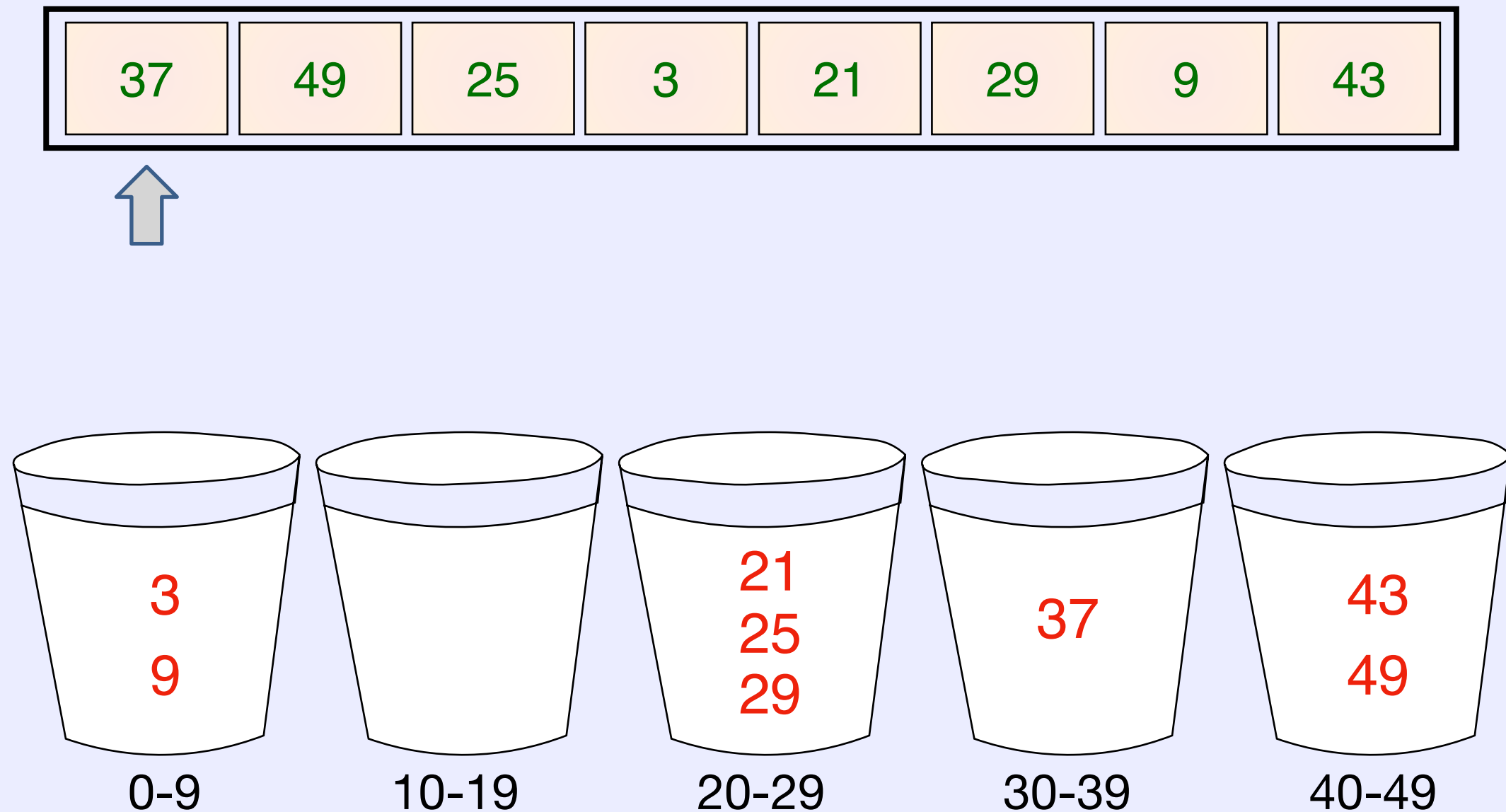
Increment i by 1, and decrement j by 1

swap $A[\text{start}]$ and $A[i - 1]$

Return ($i - 1$)

Bucket Sort

A non-comparison based sorting algorithm:



In the end: use buckets to obtain a sorted array!

Bucket Sort

Bucket-Sort(A, n, k):

/ Bucket-sort on array of size 'n' using 'k' buckets*/*

$B \leftarrow$ new array of k empty lists

$M \leftarrow 1 + \max_i A[i]$

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

Append $A[i]$ to $B[\lfloor A[i] \times (k/M) \rfloor]$

for $i = 0$ to $k - 1$ **do**

Sort $B[i]$

/ You can use any sorting algorithm*/*

return $B[0] \circ B[1] \circ \dots \circ B[k - 1]$

Remark: If $k = M$ then each bucket will contain identical elements.

Count Sort

Goal: To sort an array A containing n positive integers in time $O\left(n + \max_i A[i]\right)$

Count-Sort(A):

$M \leftarrow 1 + \max_i A[i]$

$C \leftarrow$ new array of size M whose entries are initialised to 0

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

Increment $C[A[i]]$ by 1

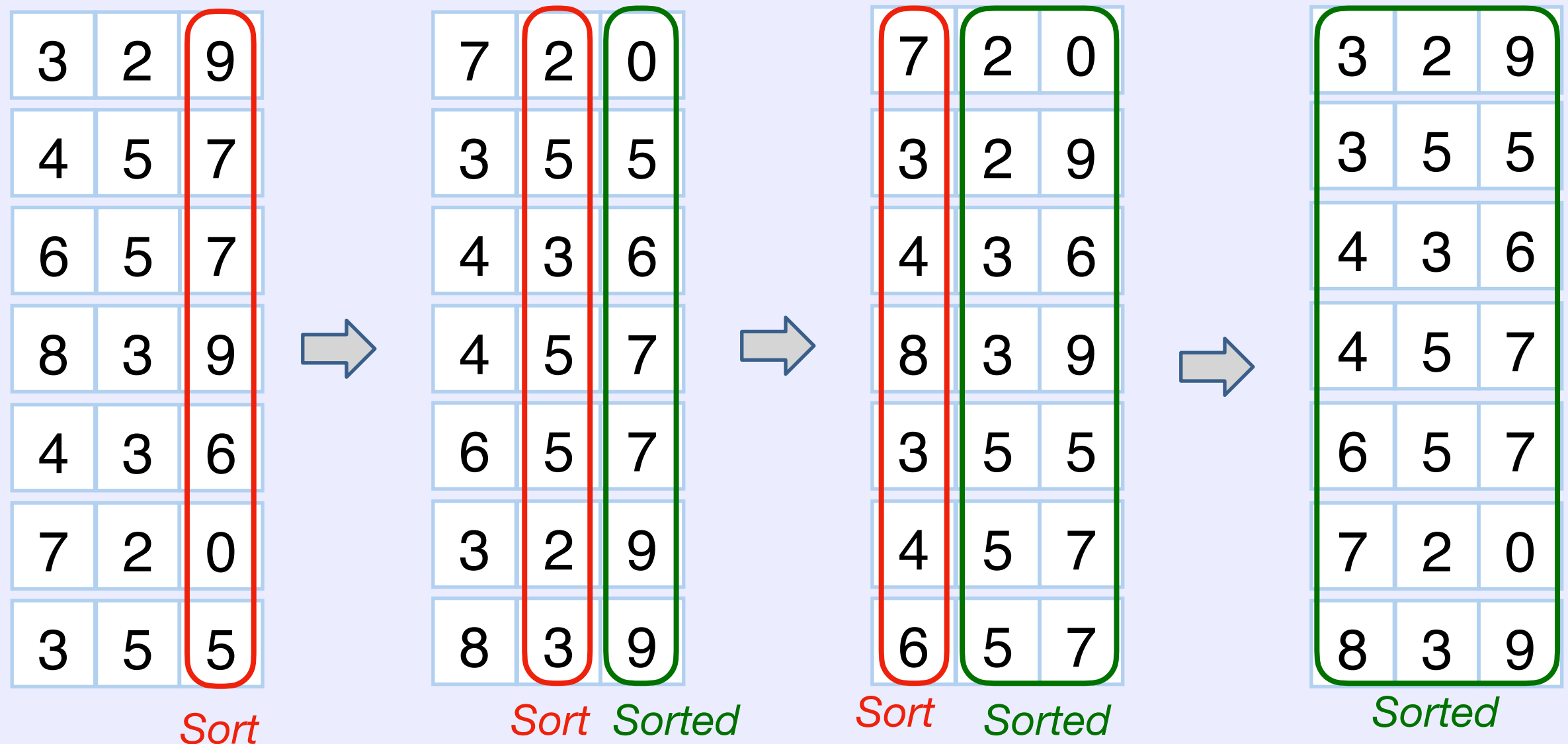
Use C to store in A the elements in sorted order

return A

Remark: An array A containing n integers in range $[0, \Theta(n)]$ can be sorted in $O(n)$ time.

Radix Sort

Sort digits (individual characters) in order of significance (i.e. from LSD to MSD)
(At each iteration perform sorting using Count Sort)



Application: Is useful for sorting strings, example: entry numbers.

Homework

1. Write pseudo-code for Radix Sort.
2. Prove that the time-complexity of Radix sort is $O(w \cdot n)$
where, n denotes the number of elements to be sorted, and
 w denotes the width (number of characters in each element).