CENG 218
Design and Analysis of Algorithms

Izmir Institute of Technology

Lecture 7: Sorting in linear time

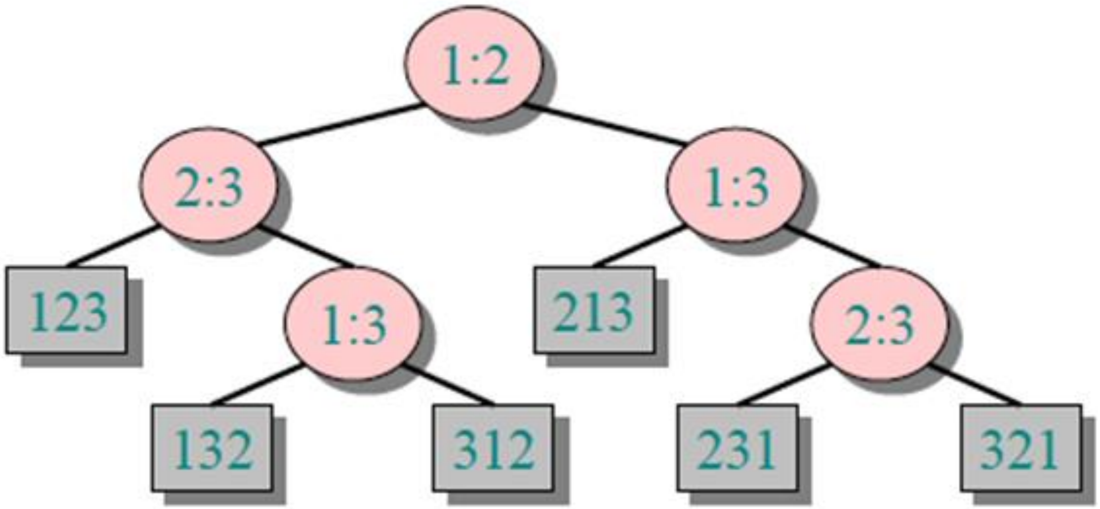
Slides were mostly prepared using the material provided by Prof. Charles E. Leiserson and Prof. Erik Demaine from MIT

How fast can we sort?

- It depends on the model, i.e. what you can do with the elements.
- All the sorting algorithms we have seen so far are *comparison sorts*: only use comparisons to determine the relative order of elements.
  E.g., insertion sort (n^2), merge sort ($n \lg n$), quicksort ($n \lg n$).
- Is $O(n \lg n)$ the best we can do with comparison sort model?
  Decision tree can help us answer this question.

Decision-tree example

Sort <a_1, a_2, a_3>

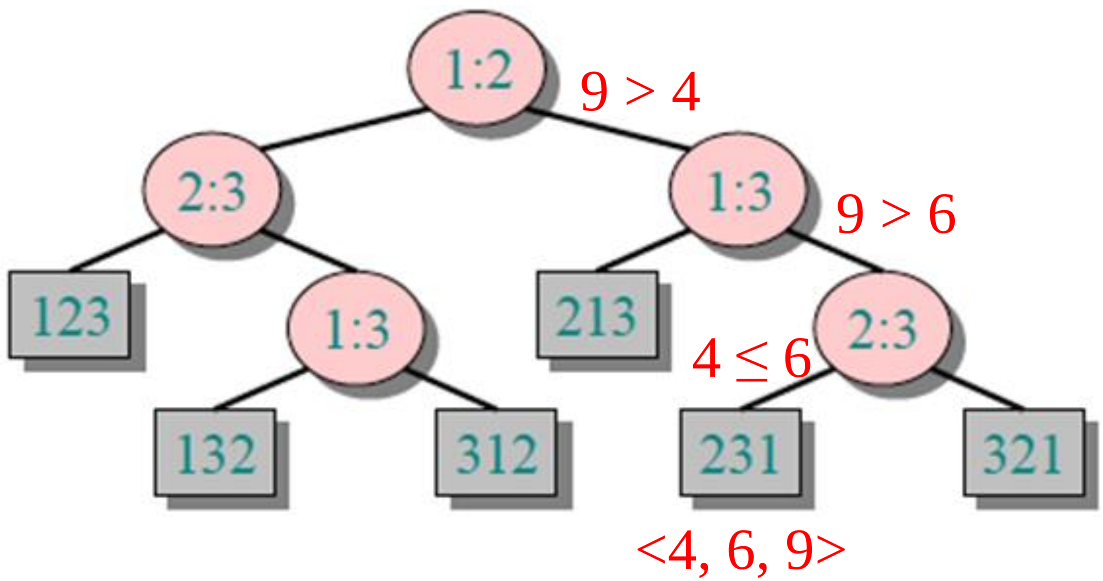


In general (n elements)

- Each internal node is labeled i:j for comparing a_i and a_j
- The left subtree shows subsequent comparisons if $a_i \leq a_j$
- The right subtree shows subsequent comparisons if $a_i > a_j$

Decision-tree example

Example: Sort <9, 4, 6>



Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$ has been established.

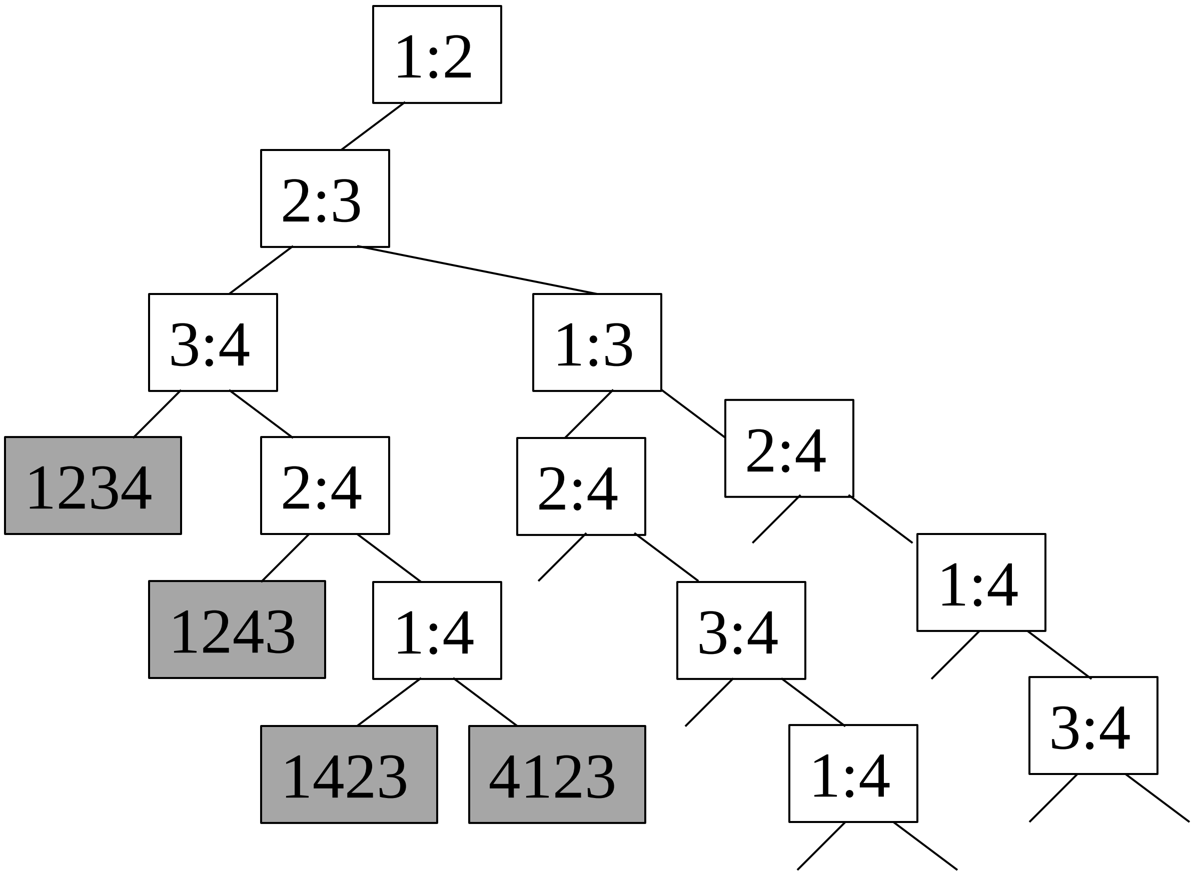
Decision-tree model

A decision tree can model the execution of any comparison sort:

- It serves as graphical representation of algorithms.
- One tree for each input size n (not so generic).
- View the algorithm as splitting whenever it compares two elements.
- The running time of the algorithm = the length of the path taken.
- Worst-case running time = height of tree.

Decision-tree, Insertion Sort, n=4

An uncomplete decision tree to model insertion sort:



Lower bound for decision-tree sorting

Theorem: Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof: The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations.

A height-h binary tree has $\leq 2^h$ leaves.

Thus, $2^h \geq n!$

$$
\begin{aligned}
h &\geq \lg(n!) && \text{(lg is mono. increasing)} \\
&\geq \lg((n/e)^n) && \text{(Stirling's formula)} \\
&= n(\lg n - \lg e) && \\
&= \Theta(n \lg n) && \text{(lg } e \text{ is constant)}
\end{aligned}
$$

Worst-case running time

h is $\Omega(n \lg n)$.

Sorting in linear time

Counting sort: No comparisons between elements.

- Input: $A[1 \,..\, n]$, where $A[\,j] \in \{1, 2, \ldots, k\}$.
- Output: $B[1 \,..\, n]$, sorted.
- Auxiliary storage: $C[1 \,..\, k]$.
  I.e. Size of C is the maximum number in A.

Counting sort

for $i \leftarrow 1$ **to** k

$\quad C[i] \leftarrow 0$

for $j \leftarrow 1$ **to** n

$\quad C[A[j]] \leftarrow C[A[j]] + 1 \quad \rhd\ C[i] = |\{\text{key} = i\}|$

for $i \leftarrow 2$ **to** k

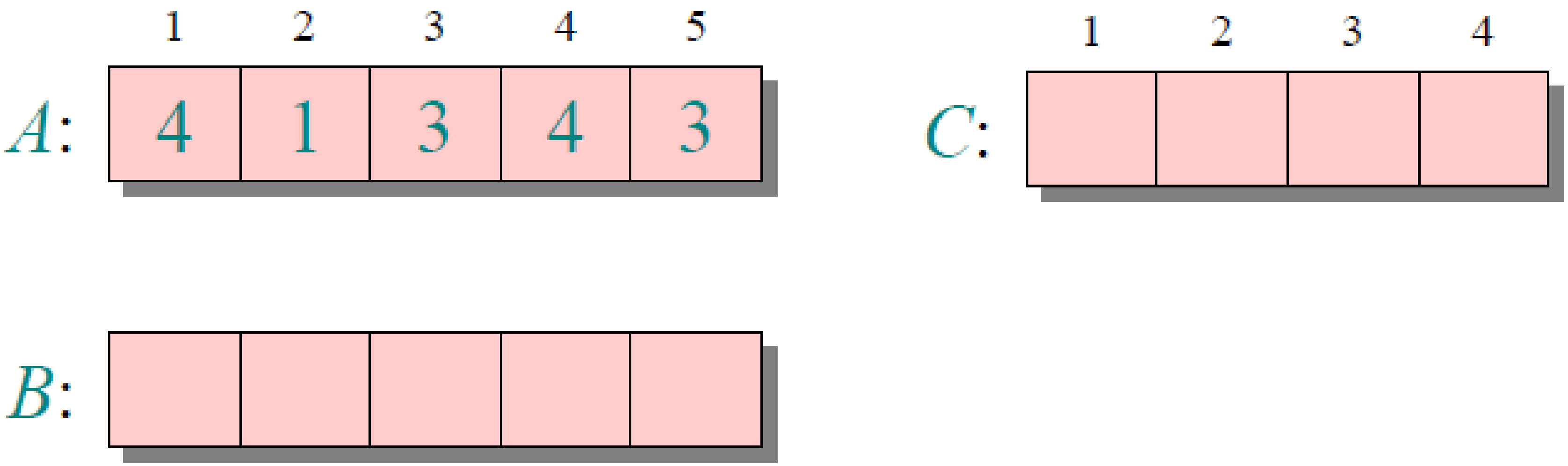
$\quad C[i] \leftarrow C[i] + C[i-1] \quad \rhd\ C[i] = |\{\text{key} \leq i\}|$

for $j \leftarrow n$ **downto** 1

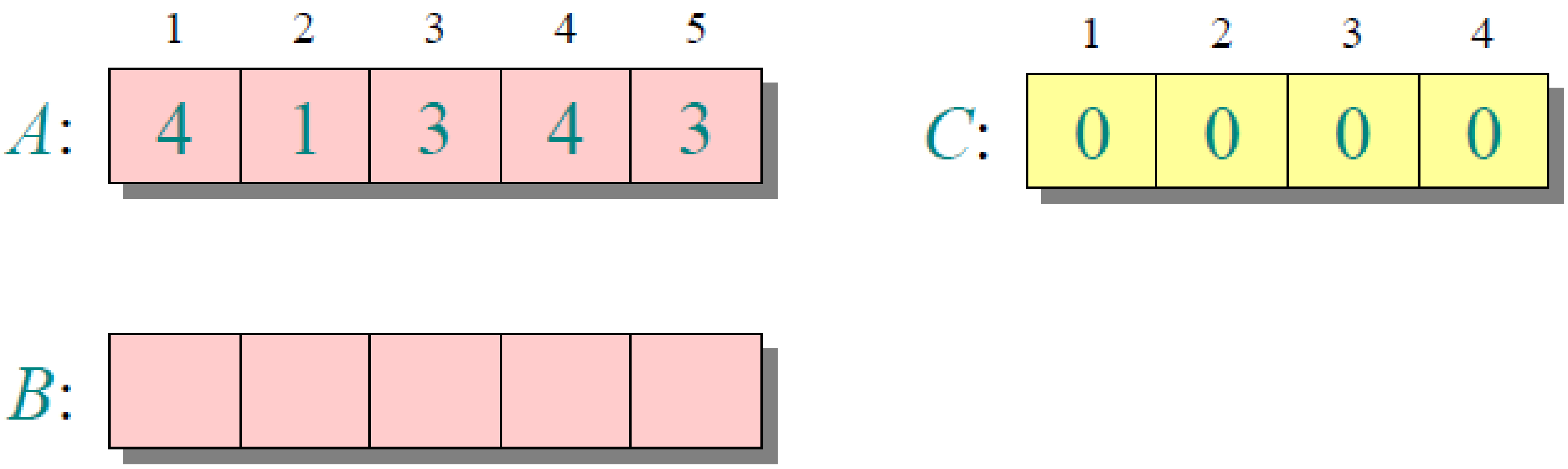
$\quad B[C[A[j]]] \leftarrow A[j]$

$\quad C[A[j]] \leftarrow C[A[j]] - 1$

Counting sort example



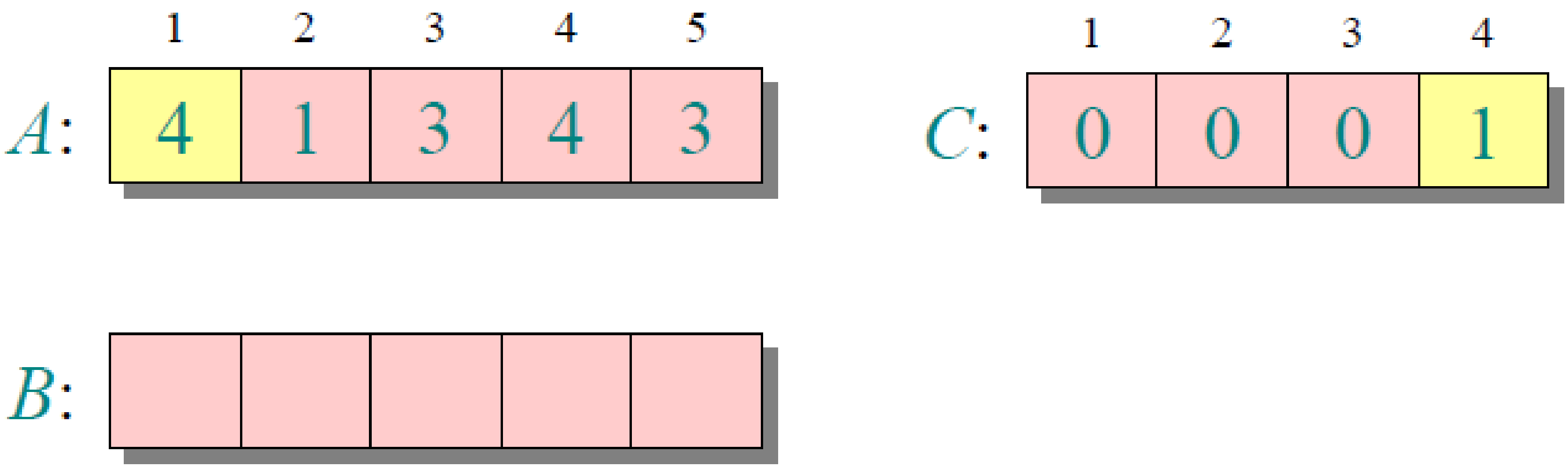
Loop 1



for $i \leftarrow 1$ **to** k

$\quad C[i] \leftarrow 0$

Loop 2



for $j \leftarrow 1$ **to** n

$$C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright\ C[i] = |\{\text{key} = i\}|$$

Loop 2

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	0	1

B:					

for $j \leftarrow 1$ **to** n

$C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright C[i] = |\{\text{key} = i\}|$

Loop 2

	1	2	3	4	5
A:	4	1	3	4	3

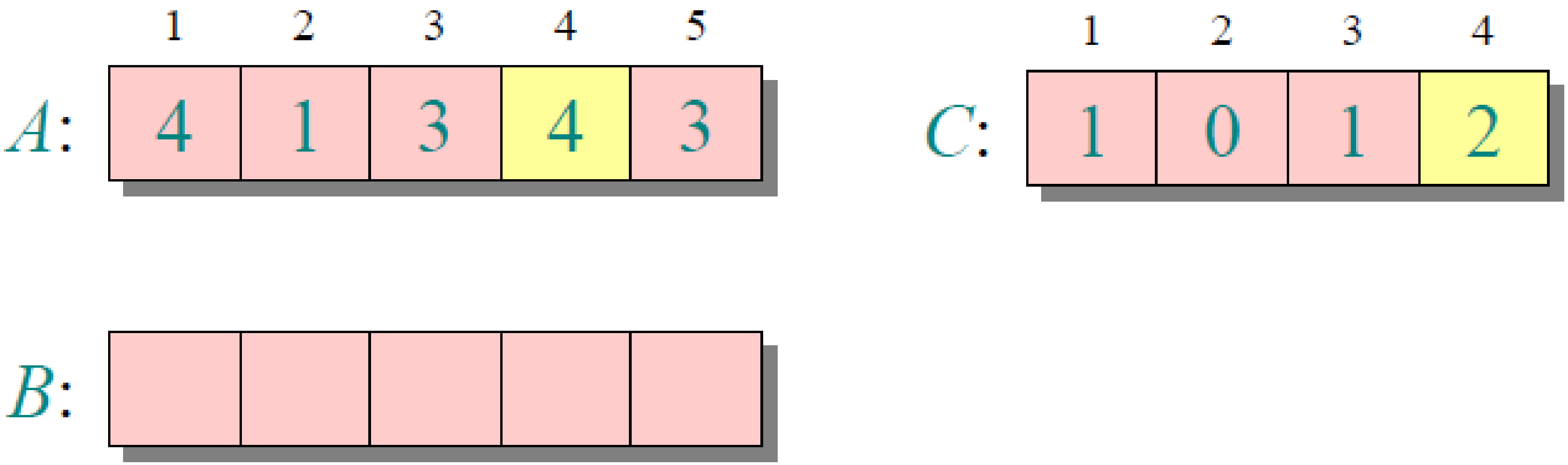
	1	2	3	4
C:	1	0	1	1

B:					

for $j \leftarrow 1$ **to** n

$$C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright\ C[i] = |\{\text{key} = i\}|$$

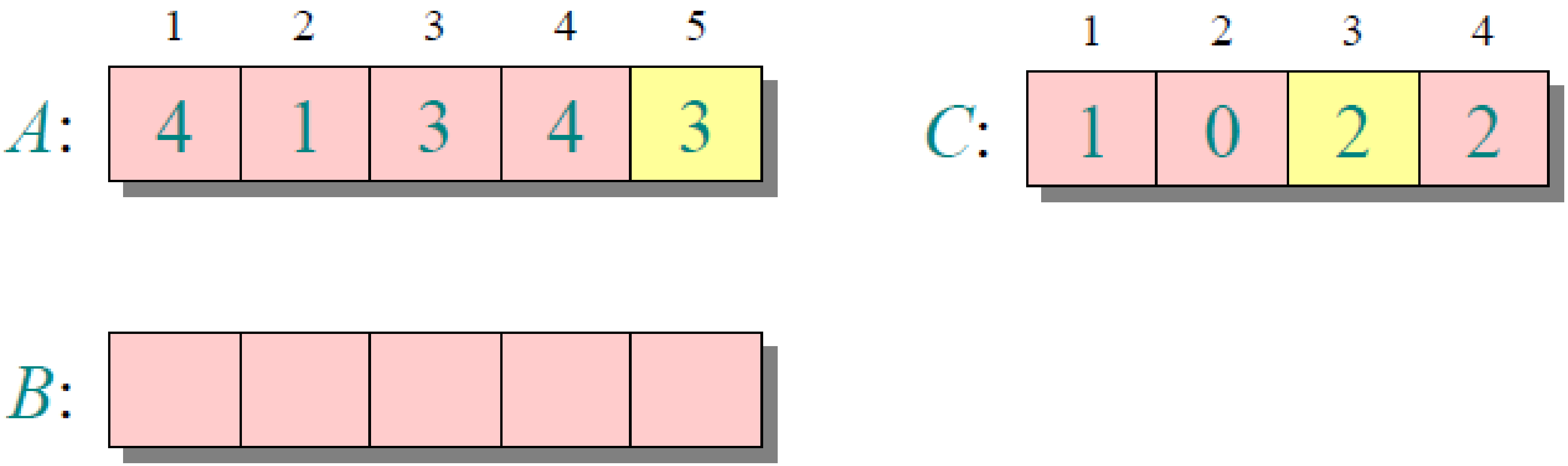
Loop 2



for $j \leftarrow 1$ **to** n

$$C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright\ C[i] = |\{\text{key} = i\}|$$

Loop 2



for $j \leftarrow 1$ **to** n

$$C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright\ C[i] = |\{\text{key} = i\}|$$

Loop 3

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	2	2

for $i \leftarrow 2$ **to** k

$\quad C[i] \leftarrow C[i] + C[i\text{–}1] \quad \rhd\ C[i] = |\{\text{key} \leq i\}|$

Loop 3

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	3	2

for $i \leftarrow 2$ **to** k

$C[i] \leftarrow C[i] + C[i–1]$ $\quad \triangleright\ C[i] = |\{\text{key} \leq i\}|$

Loop 3

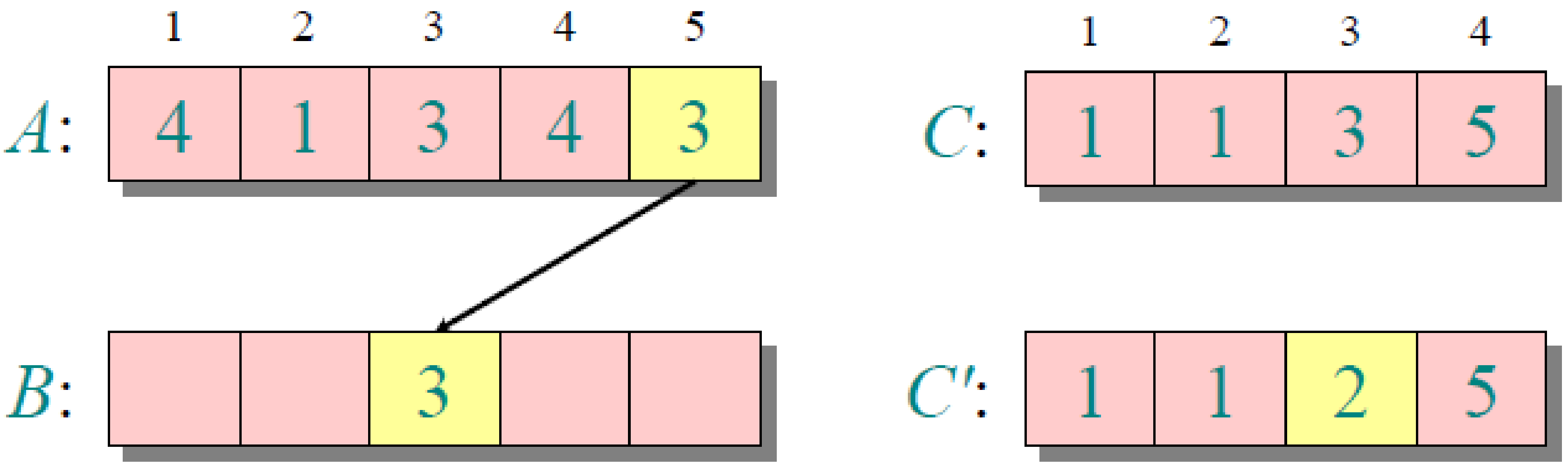
	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	3	5

for $i \leftarrow 2$ **to** k

$C[i] \leftarrow C[i] + C[i–1]$ $\rhd$ $C[i] = |\{\text{key} \leq i\}|$

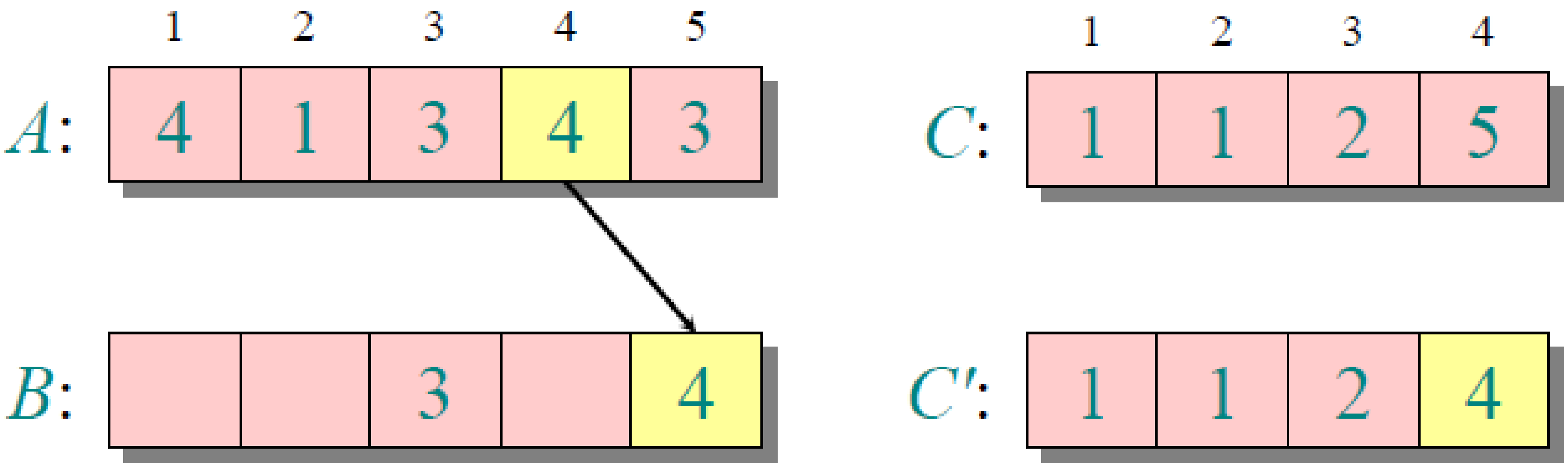
Loop 4



for $j \leftarrow n$ **downto** 1

$$B[C[A[j]]] \leftarrow A[j]$$
$$C[A[j]] \leftarrow C[A[j]] - 1$$

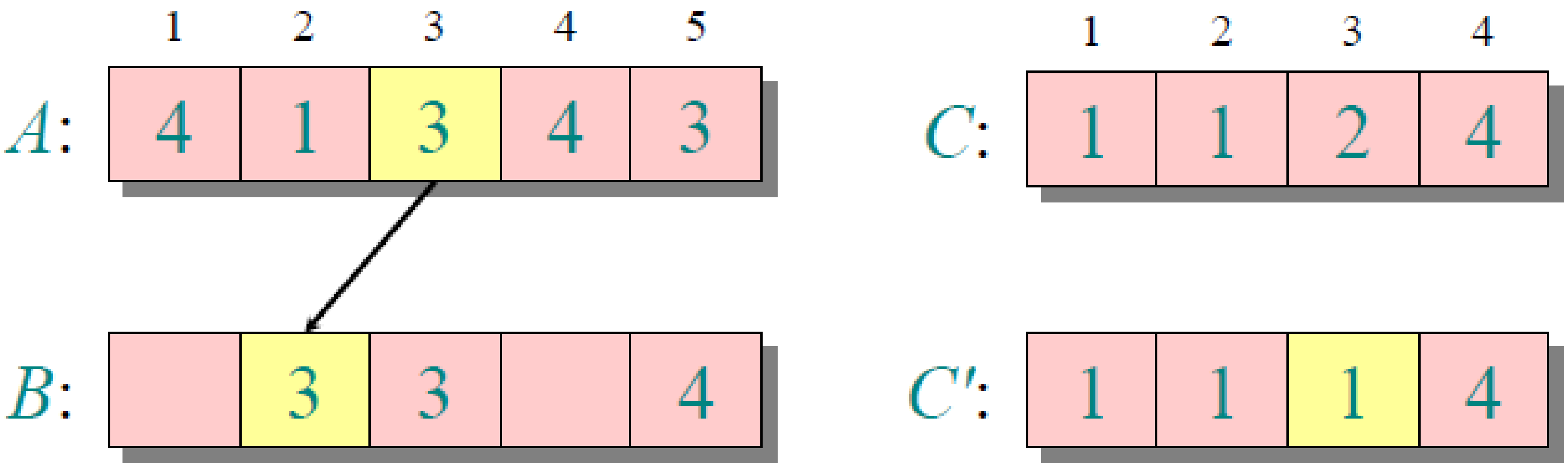
Loop 4



for $j \leftarrow n$ **downto** 1

$$B[C[A[j]]] \leftarrow A[j]$$
$$C[A[j]] \leftarrow C[A[j]] - 1$$

Loop 4

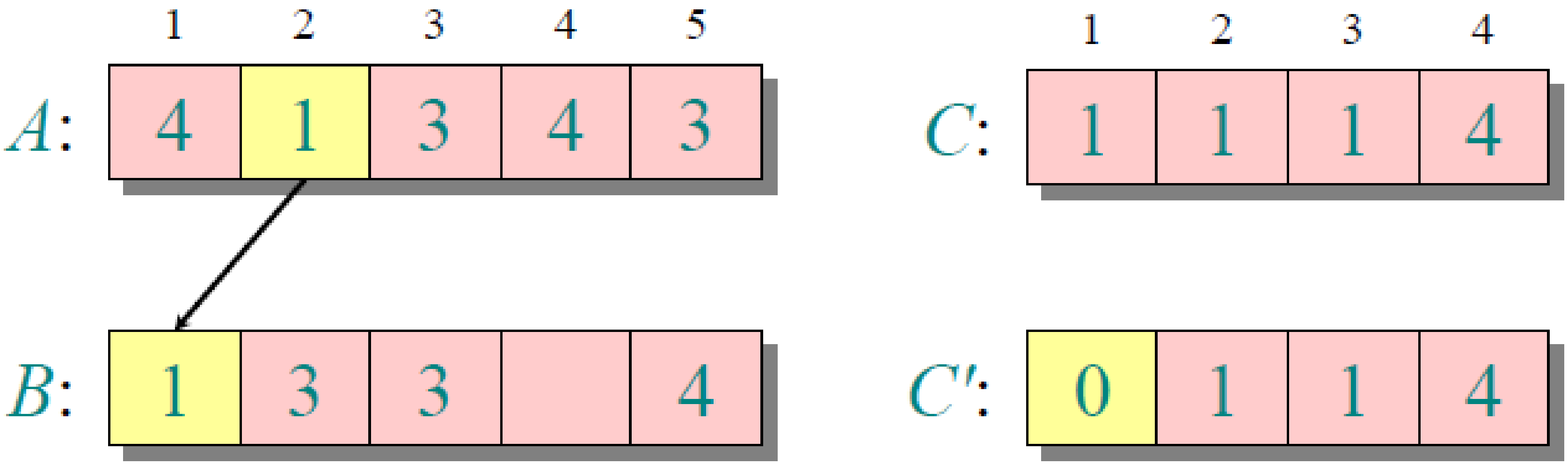


for $j \leftarrow n$ **downto** 1

$B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4

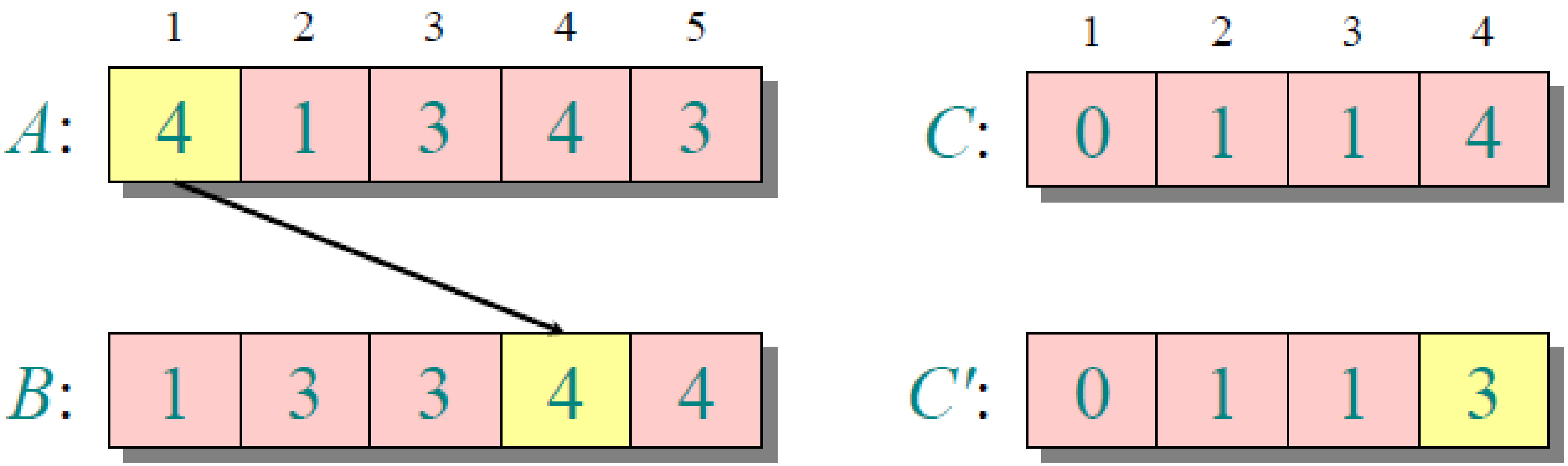


for $j \leftarrow n$ **downto** 1

$$B[C[A[j]]] \leftarrow A[j]$$

$$C[A[j]] \leftarrow C[A[j]] - 1$$

Loop 4



for $j \leftarrow n$ **downto** 1

$$B[C[A[j]]] \leftarrow A[j]$$

$$C[A[j]] \leftarrow C[A[j]] - 1$$

Analysis

$$\Theta(k) \left\{ \begin{array}{l} \textbf{for } i \leftarrow 1 \textbf{ to } k \\ \qquad C[i] \leftarrow 0 \end{array} \right.$$

$$\Theta(n) \left\{ \begin{array}{l} \textbf{for } j \leftarrow 1 \textbf{ to } n \\ \qquad C[A[j]] \leftarrow C[A[j]] + 1 \end{array} \right.$$

$$\Theta(k) \left\{ \begin{array}{l} \textbf{for } i \leftarrow 2 \textbf{ to } k \\ \qquad C[i] \leftarrow C[i] + C[i-1] \end{array} \right.$$

$$\Theta(n) \left\{ \begin{array}{l} \textbf{for } j \leftarrow n \textbf{ downto } 1 \\ \qquad B[C[A[j]]] \leftarrow A[j] \\ \qquad C[A[j]] \leftarrow C[A[j]] - 1 \end{array} \right.$$

$$\Theta(n + k)$$

Running time

- If $k = O(n)$, then counting sort takes $\Theta(n)$ time.
- We have seen that ***comparison sorting*** takes $\Omega(n \lg n)$ time.

Q. How is this possible?

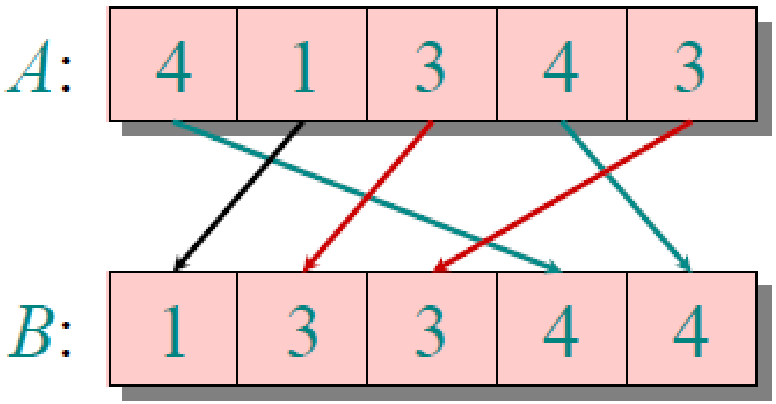
A. Counting sort is not a ***comparison sort***.

In fact, not a single comparison between elements occurs!

Note: Counting sort is not a good choice if $k >> n$.

Stable sorting

Counting sort is a ***stable*** sort: It preserves the input order among equal elements.



Exercise: What other sorts have this property?

Radix sort

- ***Origin***: Herman Hollerith's card-sorting machine for the 1890 U.S. Census. (See details in Appendix)
- Digit-by-digit sort.
- Hollerith's original(bad) idea: sort the most significant digit first. Causing many bins!

- Good idea: Sort on ***least-significant digit first*** with an auxiliary ***stable*** sort.

Counting sort is a good choice

3 2 9
4 5 7
6 5 7
8 3 9
4 3 6
7 2 0
3 5 5

Operation of Radix sort

3 2 9	7 2 0	7 2 0	3 2 9
4 5 7	3 5 5	3 2 9	3 5 5
6 5 7	4 3 6	4 3 6	4 3 6
8 3 9	4 5 7	8 3 9	4 5 7
4 3 6	6 5 7	3 5 5	6 5 7
7 2 0	3 2 9	4 5 7	7 2 0
3 5 5	8 3 9	6 5 7	8 3 9

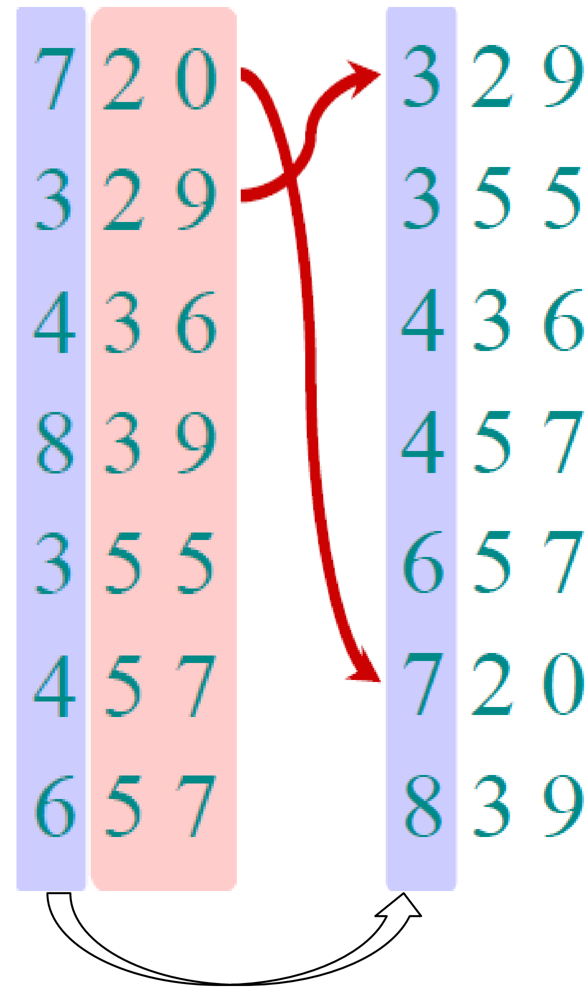
Correctness of Radix sort

- Assume that the numbers are sorted by their low-order $t-1$ digits.
- Sort on digit t

720	329
329	355
436	436
839	457
355	657
457	720
657	839

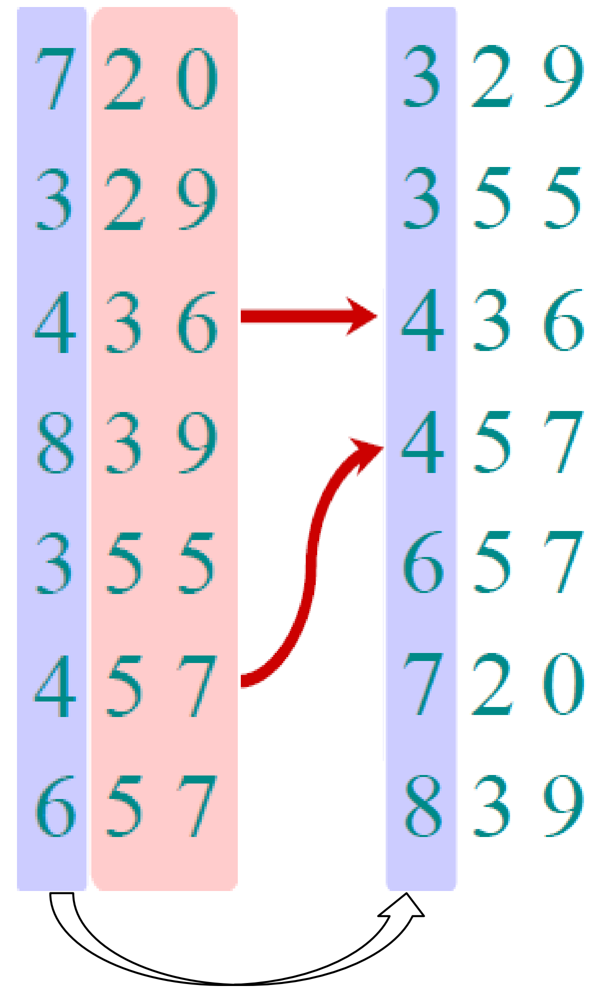
Correctness of Radix sort

- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t
  - ➢ Two numbers that differ in digit t are correctly sorted.



Correctness of Radix sort

- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t
  - Two numbers that differ in digit t are correctly sorted.
  - Two numbers equal in digit t are put in the same order as the input $\Rightarrow$ correct order.
    (requires an auxiliary stable sort, it may be counting sort but there are other options as well)

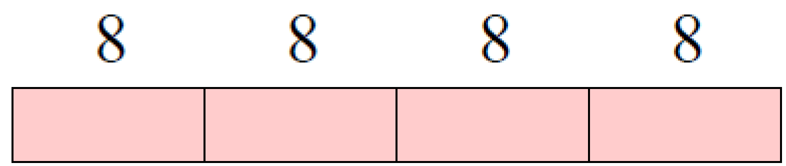


Analysis of Radix sort

- Assume counting sort is our auxiliary stable sort.
- So, sorting each digit takes $\Theta(n)$ since k is small (takes values between 0 and 9)
- Then, we need to pass through n numbers for each digit. (Three passes in the previous example).
- If the number of digits is low, it seems good.
- Let’s make a more formal analysis.

Analysis continued

- Sort n computer words of b bits each.
- Each word can be viewed as having b/r base-2^r digits.



Example: 32-bit word

$r = 8 \Rightarrow b/r = 4$ passes of counting sort on base-2^8 digits;

or $r = 16 \Rightarrow b/r = 2$ passes of counting sort on base-2^{16} digits.

How many passes should we make?

Analysis continued

- ***Recall:*** Counting sort takes $\Theta(n + k)$ time to sort n numbers in the range from 0 to $k - 1$.
- If each b-bit word is broken into r-bit pieces, each pass of counting sort takes $\Theta(n + 2^r)$ time.
- Since there are b/r passes, we have

$$T(n,b) = \Theta\left(\frac{b}{r}\left(n + 2^r\right)\right)$$

- Choose r to minimize $T(n, b)$:
  Increasing r means fewer passes,
  but as $r >> \lg n$, the time grows exponentially.

Analysis continued

$$T(n,b) = \Theta\left(\frac{b}{r}\left(n + 2^r\right)\right)$$

- We could minimize $T(n,b)$ by differentiating and setting to 0. (A calculus approach)
- But it is more intuitive to use the observation that we don't want $r > \lg n$.
- Choosing $r = \lg n$ implies $T(n,b) = \Theta(bn/\lg n)$.
- Numbers are in 0 to 2^b range. Lets substitute 2^b with n^d . (observe $d << b$). Then, we have $b = d \lg n \Rightarrow$ radix sort runs in $\Theta(dn)$ time.

Analysis result

- Note that $\Theta(dn)$ is the same with $\Theta(bn/\lg n)$ but it is easier to grasp the complexity with $\Theta(dn)$. (This assumes we select the best r and remember $2^b = n^d$).
- Counting sort handles numbers from 0 to n in linear time. Radix sort handles number from 0 to n^d in linear time, which is better.
- As long as $d < \lg n$, radix sort beats *comparison sorting*.

Example comparison with merge sort

Lets say there n many 32-bit numbers (b=32):
Assume that we select $r \sim \lg n$ (an optimum choice)

	Radix Sort	Merge Sort ($n \lg n$)
n=2000	$2^{32}=2000^d$ $d\sim3$ $\Theta(3n)$	2000 lg2000 $\sim\Theta(11n)$
n=256	$2^{32}=256^d$ $d=4$ $\Theta(4n)$	256 lg256 $\Theta(8n)$
n=32	$2^{32}=32^d$ $d\sim6$ $\Theta(6n)$	32 lg32 $\Theta(5n)$
n=2000 b=16	$2^{16}=2000^d$ $d\sim1.5$ $\Theta(1.5n)$	2000 lg2000 $\sim\Theta(11n)$

Conclusion

In practice, radix sort is fast for large size inputs when numbers are small.

Downside: Unlike quicksort, counting sort is not memory friendly. Radix sort (using counting sort) is usually slower than a well-tuned quicksort in practice.

Chapter 8 of the textbook is on linear time sorting. Please try to solve exercises in 8.1 and 8.2.

Appendix: Punched-card technology

- The 1880 U.S. Census took almost 10 years to process.
- While a lecturer at MIT, Herman Hollerith (1860-1929) prototyped punched-card technology.
- His machines, including a “card sorter,” allowed the 1890 census total to be reported in 6 weeks.
- He founded the Tabulating Machine Company in 1911, which later merged with other companies to form International Business Machines (IBM).

Appendix: Punched cards

- Punched card = data record.
- Algorithm = machine + human operator.
- An operator inserts a card into the press.
- Pins on the press reach through the punched holes to make electrical contact with cups beneath the card.
- Whenever a particular digit value is punched, the lid of the corresponding sorting bin lifts. The operator deposits the card into the bin and closes the lid.
- When all cards have been processed, the front panel is opened, and the cards are collected in order.
- Check http://www.oz.net/~markhow/writing/holl.htm
- Punch-card creator http://www.facade.com/legacy/punchcard/