Lecture 7: Linear Time Sorting

Time complexities of Comparison Sorts

Sorting Algorithms	Best Case	Average Case	Worst Case
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$ (Randomized Quick sort)	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

HOW FAST CAN WE SORT?

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, merge sort, quicksort, heapsort.

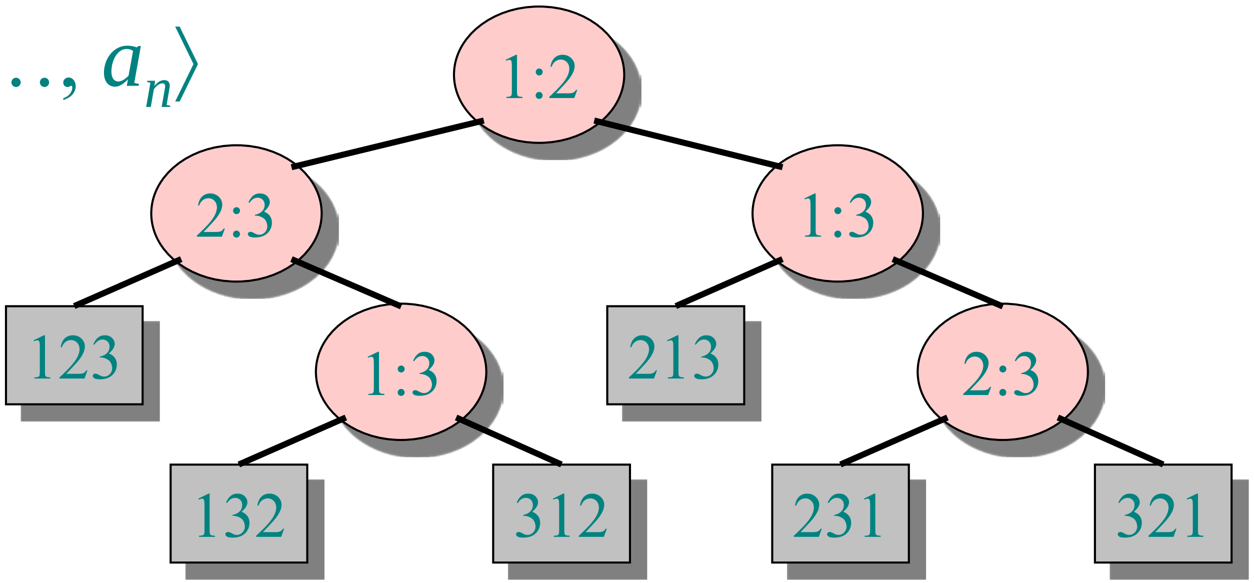
The best worst-case running time that we've seen for comparison sorting is $O(n \lg n)$.

Is $O(n \lg n)$ the best we can do?

Decision trees can help us answer this question.

DECISION-TREE EXAMPLE

Sort $\langle a_1, a_2, \ldots, a_n \rangle$

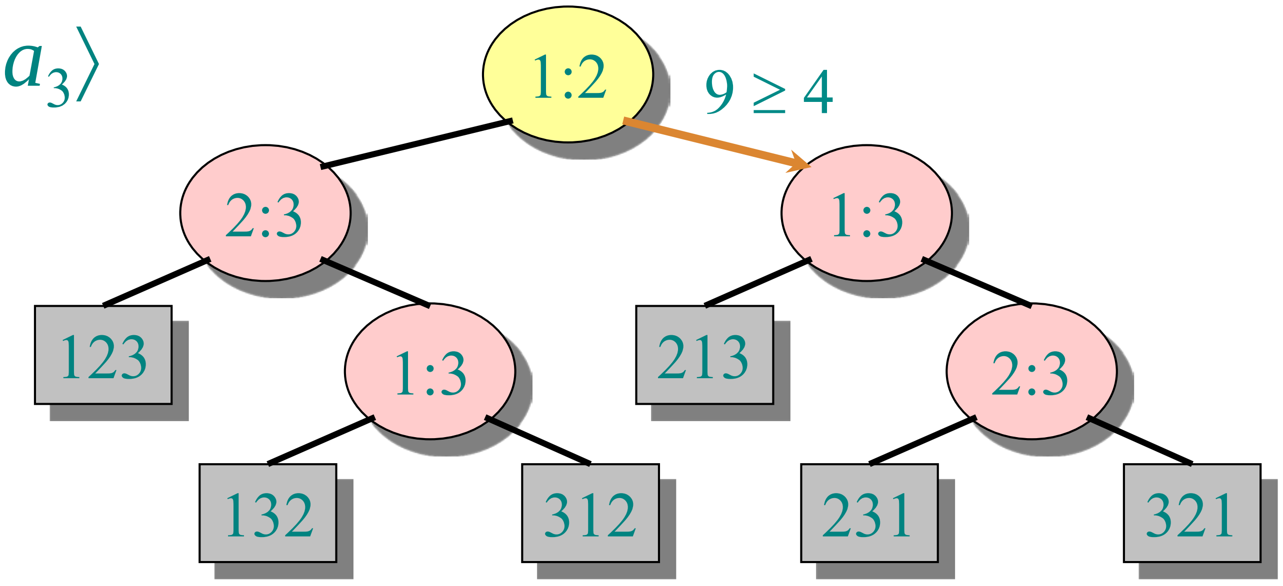


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

DECISION-TREE EXAMPLE

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle\, 9, 4, 6\, \rangle$:

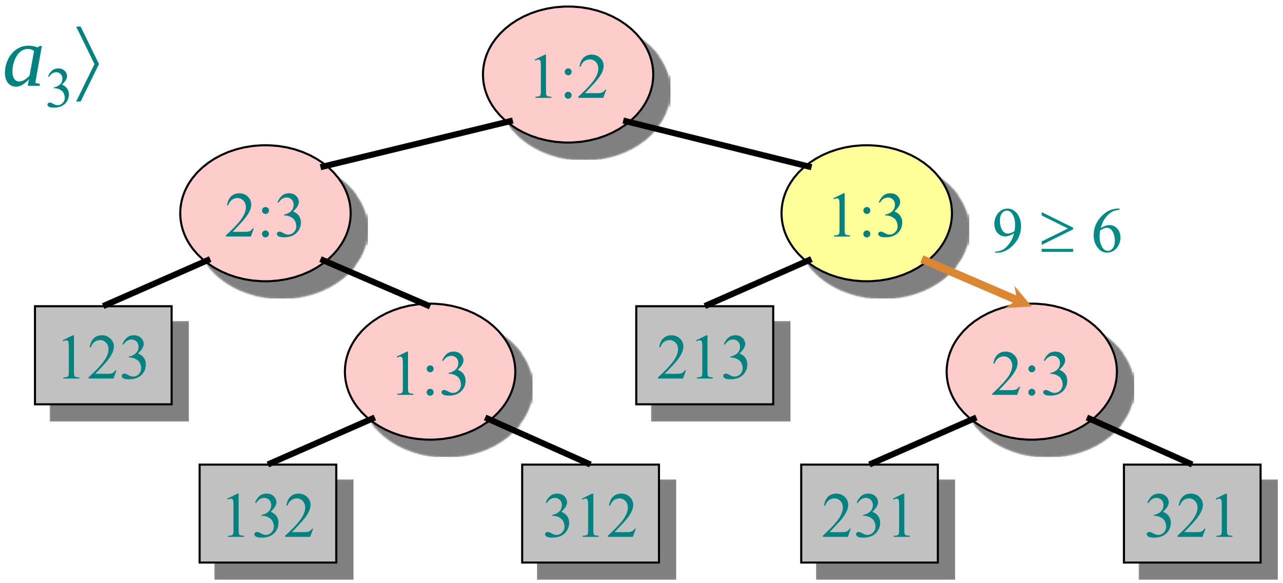


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

DECISION-TREE EXAMPLE

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle 9, 4, 6 \rangle$:

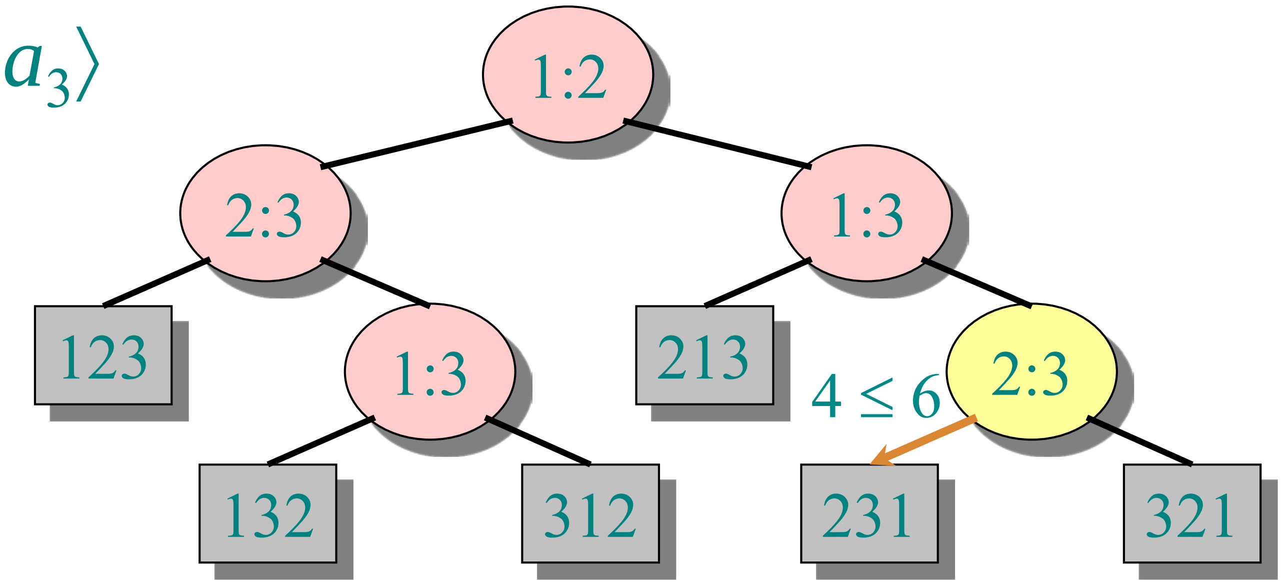


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

DECISION-TREE EXAMPLE

Sort $\langle a_1, a_2, a_3 \rangle = \langle 9, 4, 6 \rangle$:

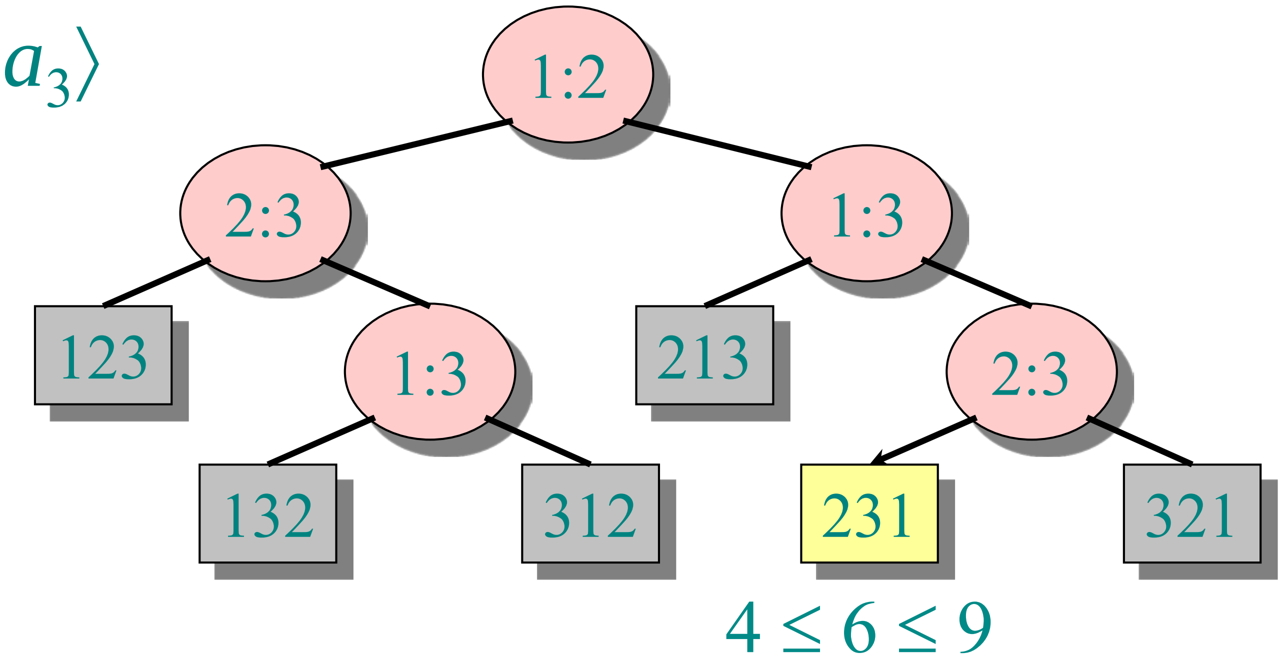


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

DECISION-TREE EXAMPLE

Sort $\langle a_1, a_2, a_3\rangle$
$= \langle\, 9, 4, 6 \,\rangle$:



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

DECISION-TREE MODEL

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

- One tree for each input size n.
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- The running time of the algorithm $=$ the length of the path taken.
- Worst-case running time $=$ height of tree.

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, $n! \leq 2^h$.

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

LOWER BOUND FOR COMPARISON SORTING

Corollary. Heapsort and merge sort are asymptotically optimal comparison sorting algorithms.

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***: $C[1 \,.\,.\, k]$.

COUNTING SORT

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

  do $C[i] \leftarrow 0$

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

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

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

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

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

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

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

COUNTING-SORT EXAMPLE

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

	1	2	3	4
C:				

B:					

LOOP 1

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

	1	2	3	4
C:	0	0	0	0

B:					

for $i \leftarrow 1$ **to** k
  do $C[i] \leftarrow 0$

LOOP 2

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

	1	2	3	4
C:	0	0	0	1

B:					

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

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

LOOP 2

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	0	1

B:

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

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

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

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

LOOP 2

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	1	2

B:

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

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

LOOP 2

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

	1	2	3	4
C:	1	0	2	2

B:					

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

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

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

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

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

do $C[i] \leftarrow C[i] + C[i-1]$ $\quad \triangleright\ C[i] = |\{\text{key} \le i\}|$

LOOP 3

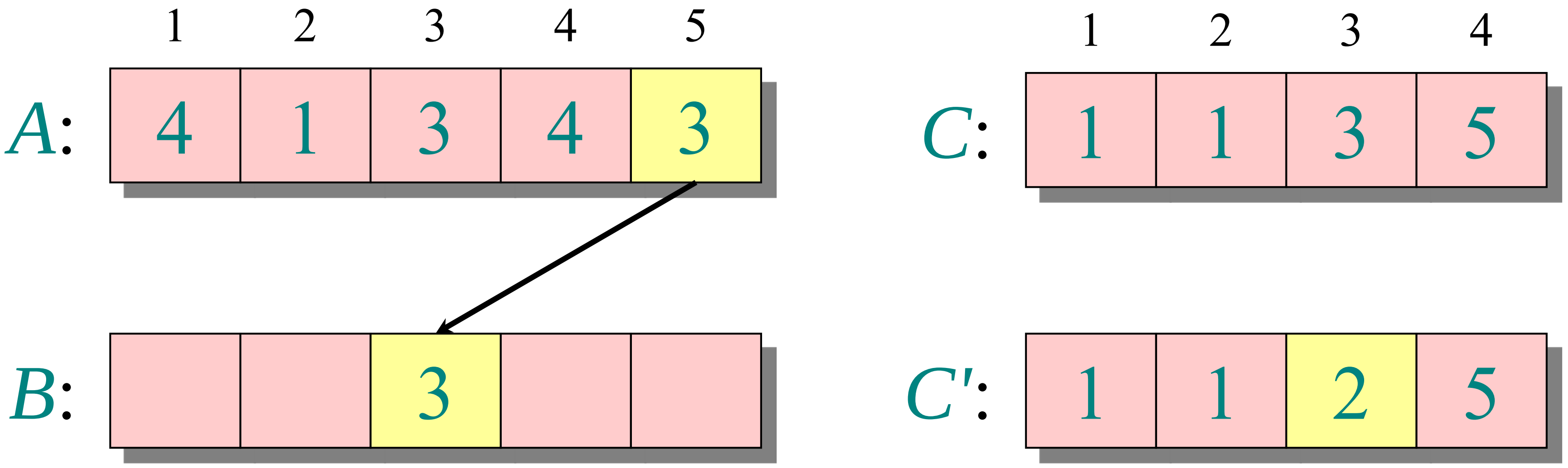
	1	2	3	4	5
A:	4	1	3	4	3
B:					

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

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

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

LOOP 4

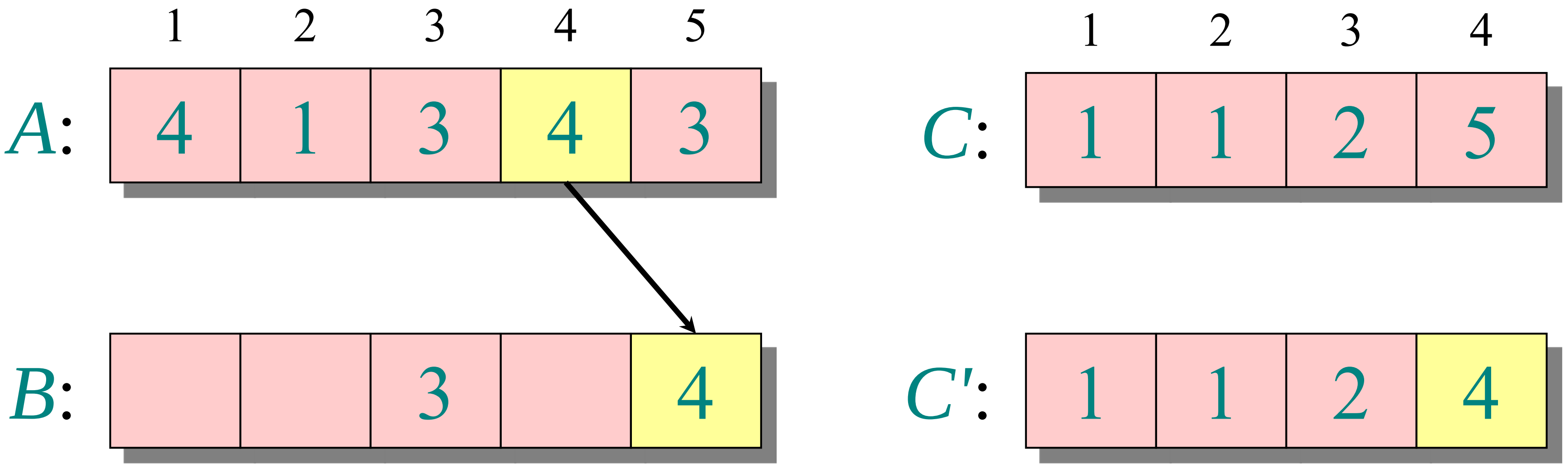


for $j \leftarrow n$ **downto** 1

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

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

LOOP 4

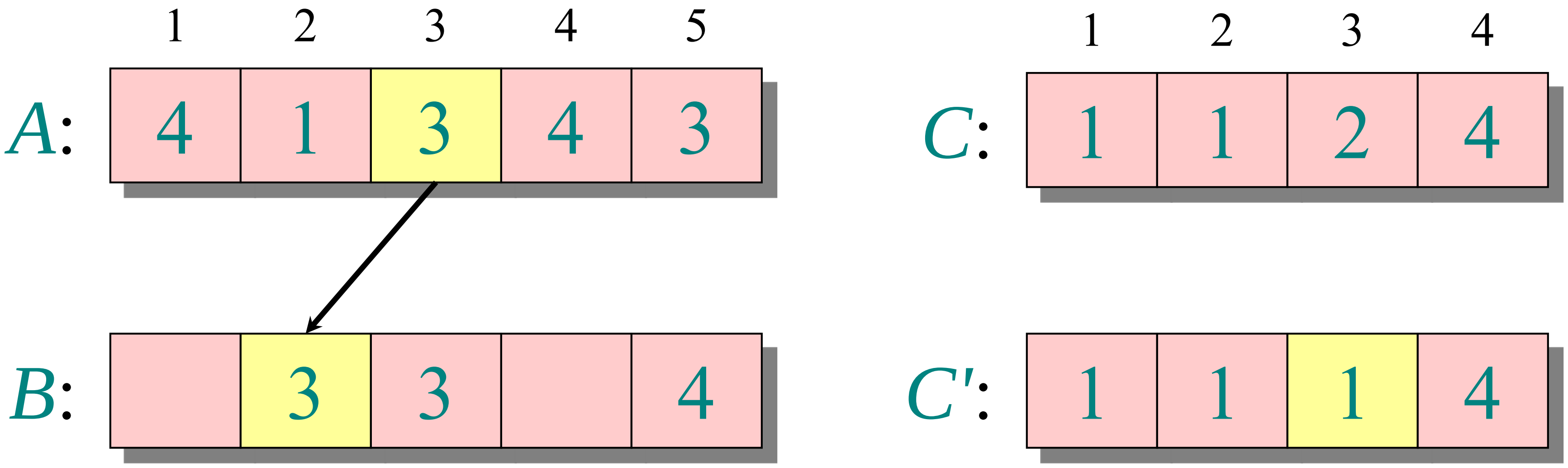


for $j \leftarrow n$ **downto** 1

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

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

LOOP 4

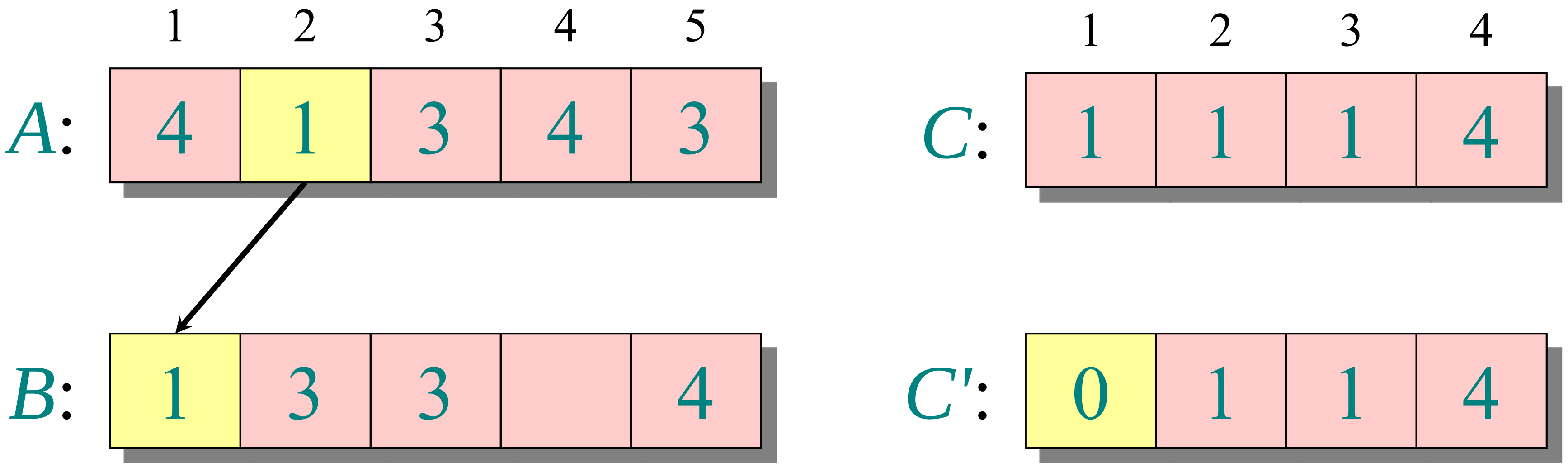


for $j \leftarrow n$ **downto** 1

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

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

LOOP 4



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

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

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

LOOP 4

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

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

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

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

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

ANALYSIS

$$
\begin{array}{ll}
\Theta(k) & \left\{\begin{array}{l} \textbf{for } i \leftarrow 1 \textbf{ to } k \\ \quad \textbf{do } C[i] \leftarrow 0 \end{array}\right. \\
\Theta(n) & \left\{\begin{array}{l} \textbf{for } j \leftarrow 1 \textbf{ to } n \\ \quad \textbf{do } C[A[j]] \leftarrow C[A[j]] + 1 \end{array}\right. \\
\Theta(k) & \left\{\begin{array}{l} \textbf{for } i \leftarrow 2 \textbf{ to } k \\ \quad \textbf{do } C[i] \leftarrow C[i] + C[i-1] \end{array}\right. \\
\Theta(n) & \left\{\begin{array}{l} \textbf{for } j \leftarrow n \textbf{ downto } 1 \\ \quad \textbf{do } B[C[A[j]]] \leftarrow \mathrm{A}[j] \\ \qquad\;\; C[A[j]] \leftarrow C[A[j]] - 1 \end{array}\right. \\
\hline
\Theta(n + k) &
\end{array}
$$

RUNNING TIME

If $k = O(n)$, then counting sort takes $\Theta(n)$ time.

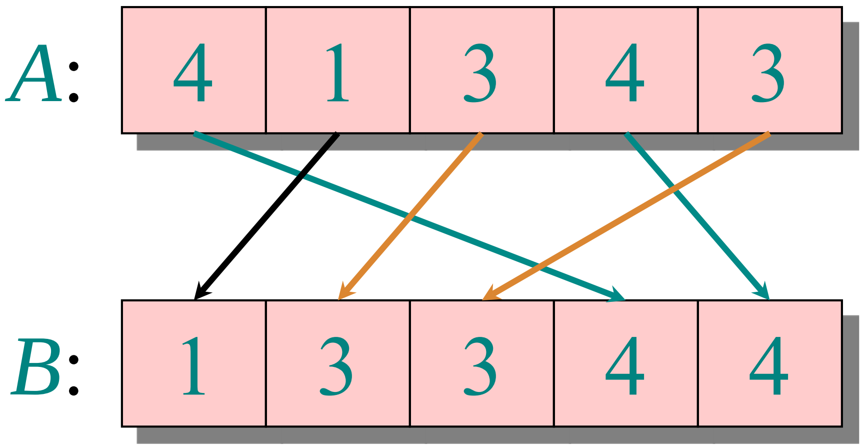
- But, sorting takes $\Omega(n \lg n)$ time!
- Where's the fallacy?

Answer:

- ***Comparison sorting*** takes $\Omega(n \lg n)$ time.
- Counting sort is not a ***comparison sort***.
- In fact, not a single comparison between elements occurs!

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

- Digit-by-digit sort.
- Original idea: sort on most-significant digit first (Bad!!!).
- Good idea: Sort on ***least-significant digit first*** with auxiliary ***stable*** sort.

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

Induction on digit position

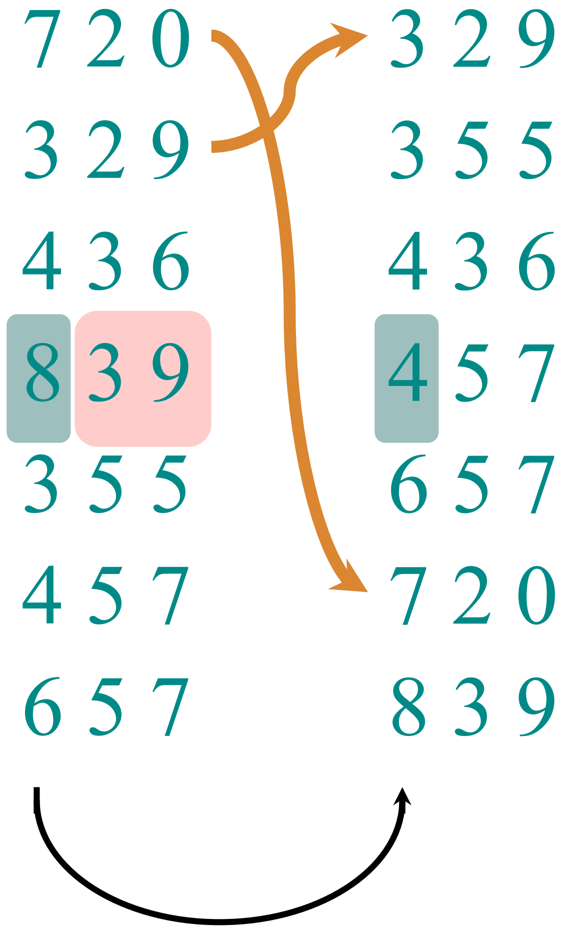
- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t

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

CORRECTNESS OF RADIX SORT

Induction on digit position

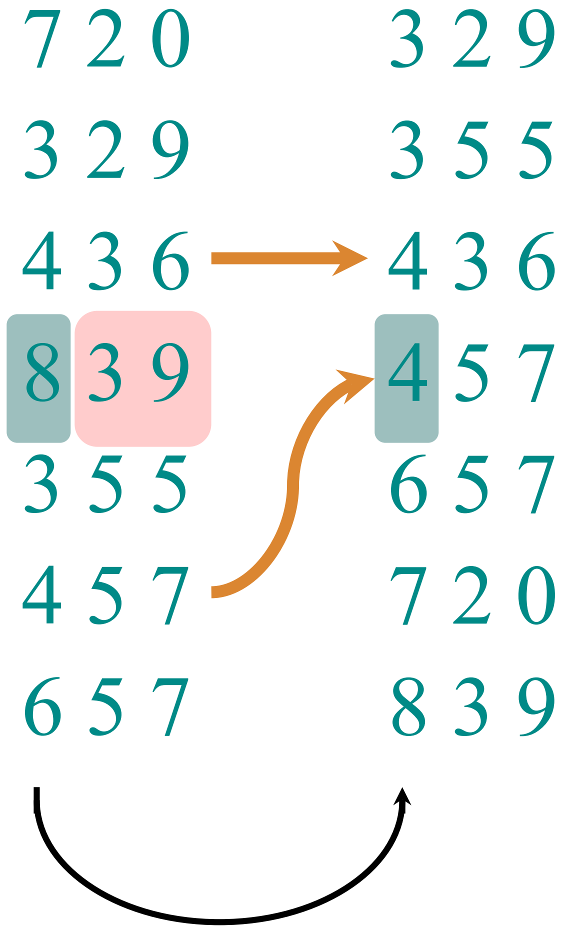
- Assume that the numbers are sorted by their low-order $t-1$ digits.
- Sort on digit t
  - Two numbers that differ in digit t are correctly sorted.



CORRECTNESS OF RADIX SORT

Induction on digit position

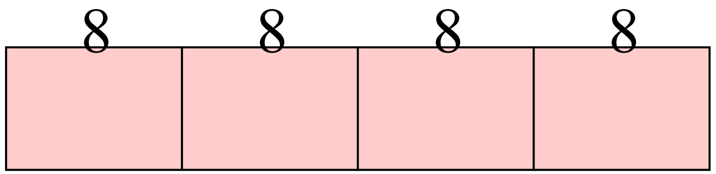
- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t
  - Two numbers that differ in digit t are correctly sorted.
  - Two numbers equal in digit t are put in the same order as the input $\Rightarrow$ correct order.



ANALYSIS OF RADIX SORT

- Assume counting sort is the auxiliary stable sort.
- 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 b/r equal 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 $r \gg \lg n$, the time grows exponentially.

CHOOSING r

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

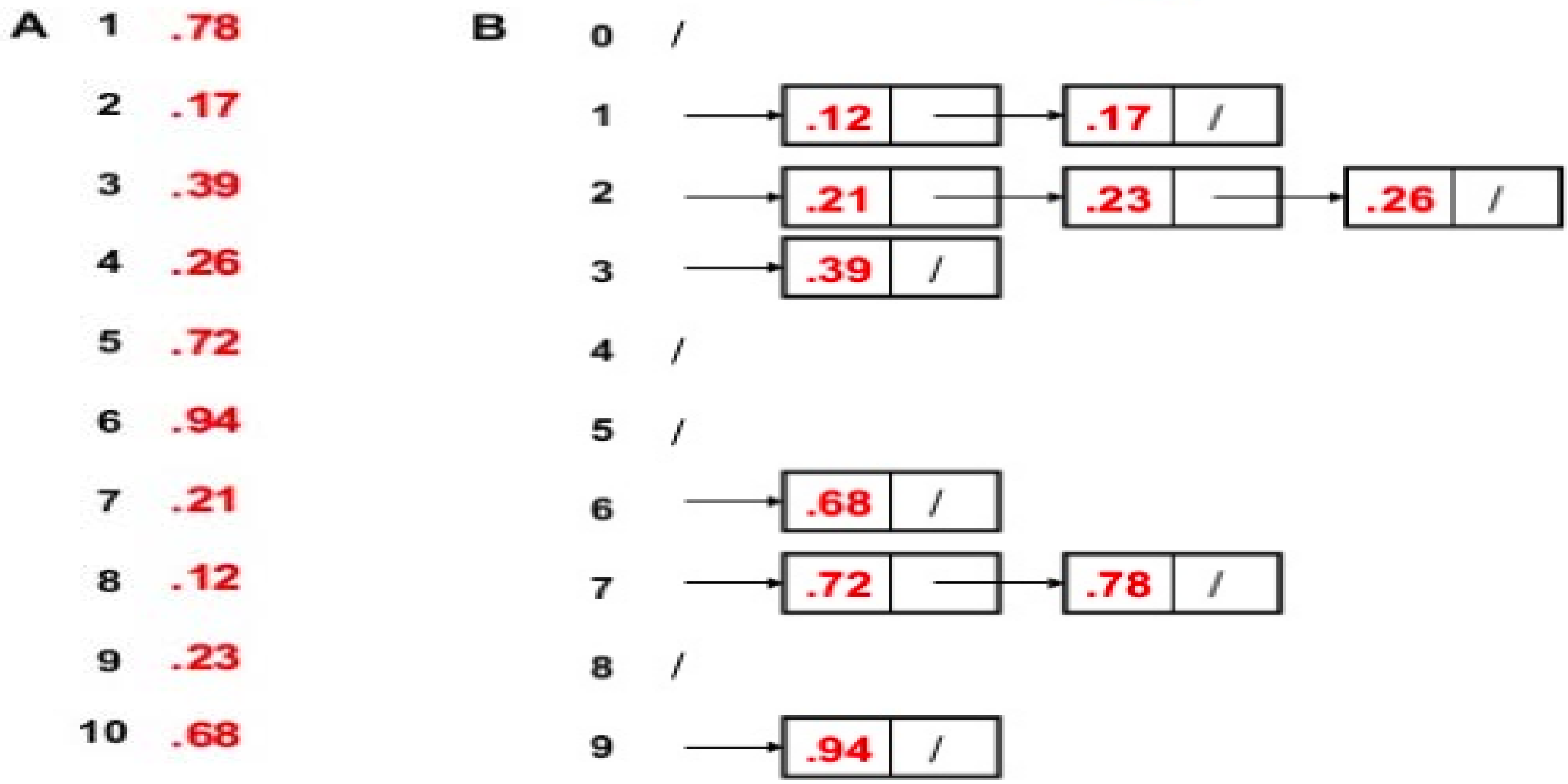
Minimize $T(n, b)$ by differentiating and setting to 0.

Or, just observe that we don't want $2^r \gg n$, and there's no harm asymptotically in choosing r as large as possible subject to this constraint.

Choosing $r = \lg n$ implies $T(n, b) = \Theta(bn/\lg n)$.

- For numbers in the range from 0 to $n^d - 1$, we have $b = d \lg n \Rightarrow$ radix sort runs in $\Theta(d\,n)$ time.

BUCKET SORT


A
1 .78
2 .17
3 .39
4 .26
5 .72
6 .94
7 .21
8 .12
9 .23
10 .68
B
0 /
1 .12 .17 /
2 .21 .23 .26 /
3 .39 /
4 /
5 /
6 .68 /
7 .72 .78 /
8 /
9 .94 /

BUCKET SORT

Idea :

- Divide the interval [0, n) into n equal – sized subintervals or buckets.
- Distribute the n input numbers into the buckets.

Since the inputs are assumed to be uniformly distributed over [0,1) , many numbers don't
fall into each bucket.

To produce the output , simply sort the numbers in each bucket and then go through the buckets , in order , listing the elements in each.

Pseudocode for Bucket Code

Bucket Sort (A)

1. n $\leftarrow$ length (A)
2. for i $\leftarrow$ 1 to n
3. do insert A[i] into list $B[\lfloor nA[i] \rfloor]$
4. for i $\leftarrow$ 0 to n-1
5. do sort list B[i] with insertion sort
6. Concatenate the list B[0] …………..B[n-1] together in order .

ANALYSIS OF RUNNING TIME

- Observe that all lines except line 5 takes O(n) time in worst case.
- We need to balenced that the total time taken by n calls to intersection sort in line 5

Let n_i be the random variables denoting the number of elements placed in bucket B[i]

So the running time of bucket sort is

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \, .$$

ANALYSIS CONT.

Taking expectations of both sides and using linearity of expectation, we have

$$
\begin{aligned}
\mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\
&= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}[O(n_i^2)] \qquad \text{(by linearity of expectation)} \\
&= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}[n_i^2]) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)
\end{aligned}
$$

ANALYSIS CONT.

we claim that

$$E(n_i^2) = 2 - (1/n) \quad \text{..................} \quad (2)$$

Define

$$X_{ij} = I\{A[j] \text{ falls in backet } i\}$$

for $i=0,1,\ldots,n-1, \quad j= 1,2,\ldots,n$

$$n_i = \sum_{j=1}^{n} X_{i_j}$$

ANALYSIS CONT.

To compute $\mathrm{E}[n_i^2]$, we expand the square and regroup terms:

$$
\begin{aligned}
\mathrm{E}[n_i^2] &= \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} X_{ij}X_{ik}\right] \\
&= \sum_{j=1}^{n}\mathrm{E}\left[X_{ij}^2\right] + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} \mathrm{E}[X_{ij}X_{ik}] \ ,
\end{aligned}
$$

ANALYSIS CONT.

As, Indicator variables X_{ij} is 1 with probabilty 1/n and 0 otherwise

so

$$E[X_{ij}^2] = 1 \cdot \frac{1}{n} + 0 \cdot \left(1 - \frac{1}{n}\right)$$

,

$$= \frac{1}{n} .$$

When $k \neq j$, the variables X_{ij} and X_{ik} are independent, and hence

$$\begin{aligned} E[X_{ij}X_{ik}] &= E[X_{ij}]E[X_{ik}] \\ &= \frac{1}{n} \cdot \frac{1}{n} \\ &= \frac{1}{n^2} . \end{aligned}$$

Substituting these two expected values in equation (8.3), we obtain

$$\begin{aligned} E[n_i^2] &= \sum_{j=1}^{n} \frac{1}{n} + \sum_{1 \le j \le n} \sum_{\substack{1 \le k \le n \\ k \neq j}} \frac{1}{n^2} \\ &= n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} \\ &= 1 + \frac{n-1}{n} \\ &= 2 - \frac{1}{n} , \end{aligned}$$

which proves (2)

ANALYSIS CONT.

Using the expected value in (1)

we can say that the running time of bucket sort is expected to be

$$T(n) = \Theta(n) + n.O(2-(1/n)) = \Theta(n)$$

thus, the entire bucket algorithm runs in *linear* expected time.