Chap 8 – Sorting in Linear Time

8.1 Lower bounds for sorting

8.2 Counting sort

8.3 Radix sort

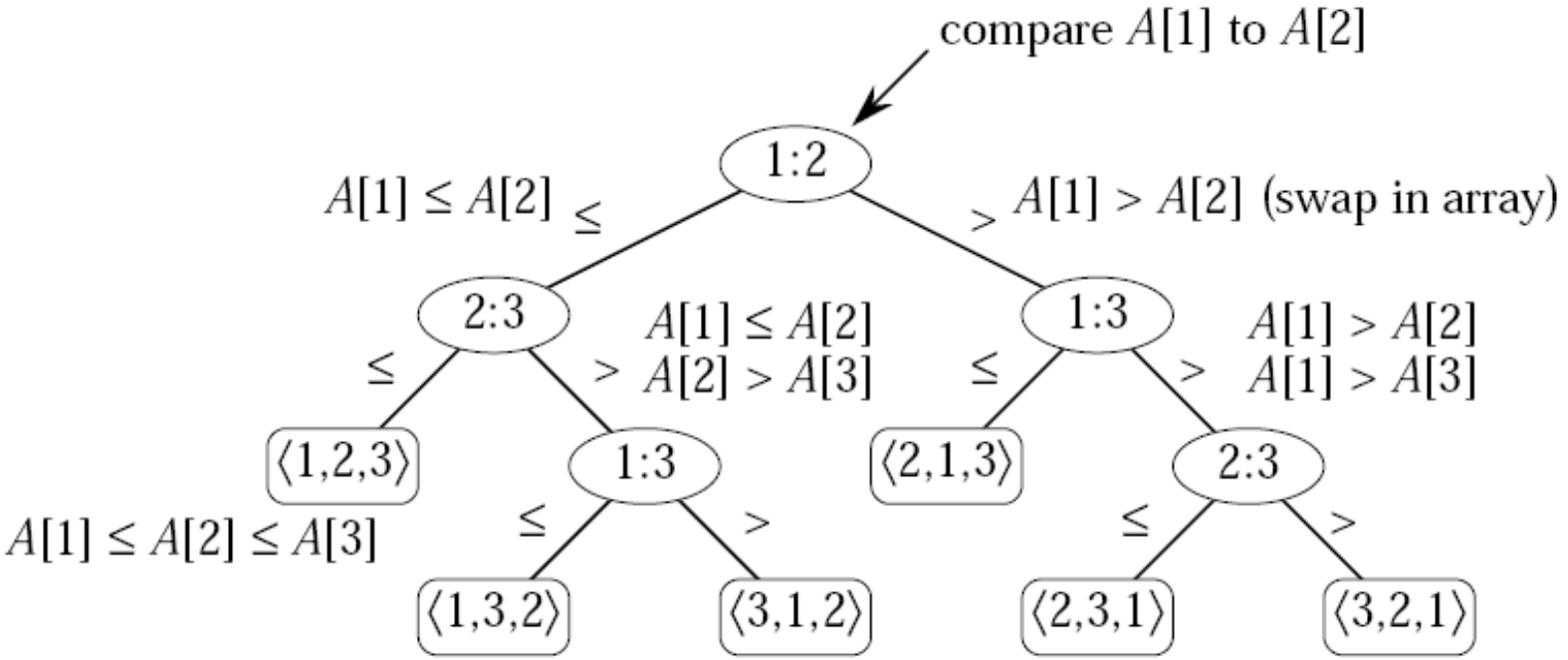
8.4 Bucket sort

8.1 Lower bounds for sorting

- Comparison sorts
  - Sorting algorithms that use comparisons of pairs of elements to gain order information.
  - All sorts seen so far are comparison sorts, e.g. insertion sort, mergesort, heap sort, quicksort.
- The decision-tree model
  - A decision tree is an abstraction of a comparison sort operating on an input of a given size.
  - For any comparison sort
    - There is one decision tree for each n.
    - The tree models all possible execution traces.

8.1 Lower bounds for sorting

- The decision-tree model
  - E.g. The decision tree for insertion sort on 3 elements.



8.1 Lower bounds for sorting

- Lower bounds for sorting
  - **LEMMA** Any binary tree of height h has $\leq 2^h$ leaves.

    Proof by induction

    Basis: $h = 0$

    The tree is just one node, which is a leaf. $1 \leq 2^0$ is true.

    Inductive step

    Extend tree of height h by making as many new leaves as possible. Each leaf becomes parent to two new leaves.

    Thus,

    # of leaves of height $h + 1$

    $= 2 \times$ # of leaves of height $h \leq 2 \cdot 2^h = 2^{h+1}$

8.1 Lower bounds for sorting

- Lower bounds for sorting
  - number of comparisons in the worst case
    = height of the decision tree
  - A decision tree must have $\geq n!$ leaves, assuming that all n elements are distinct.
    ∵ Every permutation appears at least once.
  - **THEOREM** Any comparison sort requires $\Omega(n \lg n)$ comparisons in the worst case.
    Let h = the height of a decision tree for sorting n distinct elements
    Then, $n! \leq$ # of leaves $\leq 2^h \Rightarrow h \geq \lg n! = \Omega(n \lg n)$

8.1 Lower bounds for sorting

- Lower bounds for sorting
  - **COROLLARY**

    Heapsort and merge sort are asymptotically optimal comparison sorts.
  - Q: Which of heapsort and merge sort does fewer comparisons in the worst case?

    A: Merge sort.

    ∵ Heapifying takes $2 \lg n$ comparisons (finding the max among the parent and two children takes 2 comparisons)
  - Q: Is merge sort optimal in the # of comparisons?

    A: No

8.1 Lower bounds for sorting

- Lower bounds for sorting
  - Let $S(n) =$ the minimum # of comparisons needed in the worst case. Then,

    $S(n) =$ the minimum height among all decision trees

    $\geq \lceil \lg n! \rceil$
  - Q: Is $S(n) = \lceil \lg n! \rceil$ for all $n \geq 1$?

    A: Unknown for most values of n.
  - Let $M(n) =$ the # of comparisons done by merge sort in the worst case. Then,

    $M(1) = 0$

    $M(n) = M(\lfloor n/2 \rfloor) + M(\lceil n/2 \rceil) + n - 1, \quad n > 1$

8.1 Lower bounds for sorting

- Lower bounds for sorting

n	1	2	3	4	5	6	7	8	9	10
$\lceil \lg n! \rceil$	0	1	3	5	7	10	13	16	19	22
$M(n)$	0	1	3	5	8	11	14	17	21	25

  - Observe that

    $M(n) = S(n) = \lceil \lg n! \rceil,\ 1 \le n \le 4$

    $M(n) \ge S(n) \ge \lceil \lg n! \rceil,\ 5 \le n \le 10$
  - Q: In particular, does $S(5) = 7$ or 8?

    A: $S(5) = 7$

    Thus, merge sort isn't optimal for $n = 5$.

8.1 Lower bounds for sorting

- Lower bounds for sorting: $S(5) = 7$
  - Compare $x_1 : x_2, x_3 : x_4$
  - Compare the two larger numbers
  - Insert b_3 and b_2, in that order, into the main chain using binary insertion
    - Inserting b_3 takes 2 comparisons, and resulting in

$a_1 \rightarrow a_2$ (main chain)

b_1 b_2 $b_3\ (= x_5)$

$b_3 \rightarrow b_1 \rightarrow a_1 \rightarrow a_2$

b_2

$b_1 \rightarrow b_3 \rightarrow a_1 \rightarrow a_2$

b_2

$b_1 \rightarrow a_1 \rightarrow b_3 \rightarrow a_2$

b_2

$b_1 \rightarrow a_1 \rightarrow a_2 \rightarrow b_3$

b_2

    - Inserting b_2 takes 2 more comparisons.

8.1 Lower bounds for sorting

- Lower bounds for sorting
  - Merge insertion sort

    A generalization of the preceding $n = 5$ case.

    Let

    $F(n) =$ the # of comparisons done by merge insertion sort in the worst case.

    It is known that

    $F(n) = S(n) = \lceil \lg n! \rceil,\ 1 \le n \le 11, n = 20,21$

8.2 Counting sort

- Counting sort

  Depend on a *key assumption*: $0 \le A[i] \le k, \forall 1 \le i \le n$

  COUNTING-SORT(A, B, n, k) Sort A into B

  let $C[0..k]$ be a new array

  for $i = 0$ **to** k $C[i] = 0$

  for $j = 1$ **to** n

  $C[A[j]] = C[A[j]] + 1$ $C[i] = (\#\text{ of elements} = i)$

  for $i = 1$ **to** k

  $C[i] = C[i] + C[i-1]$ $C[i] = (\#\text{ of elements} \le i)$

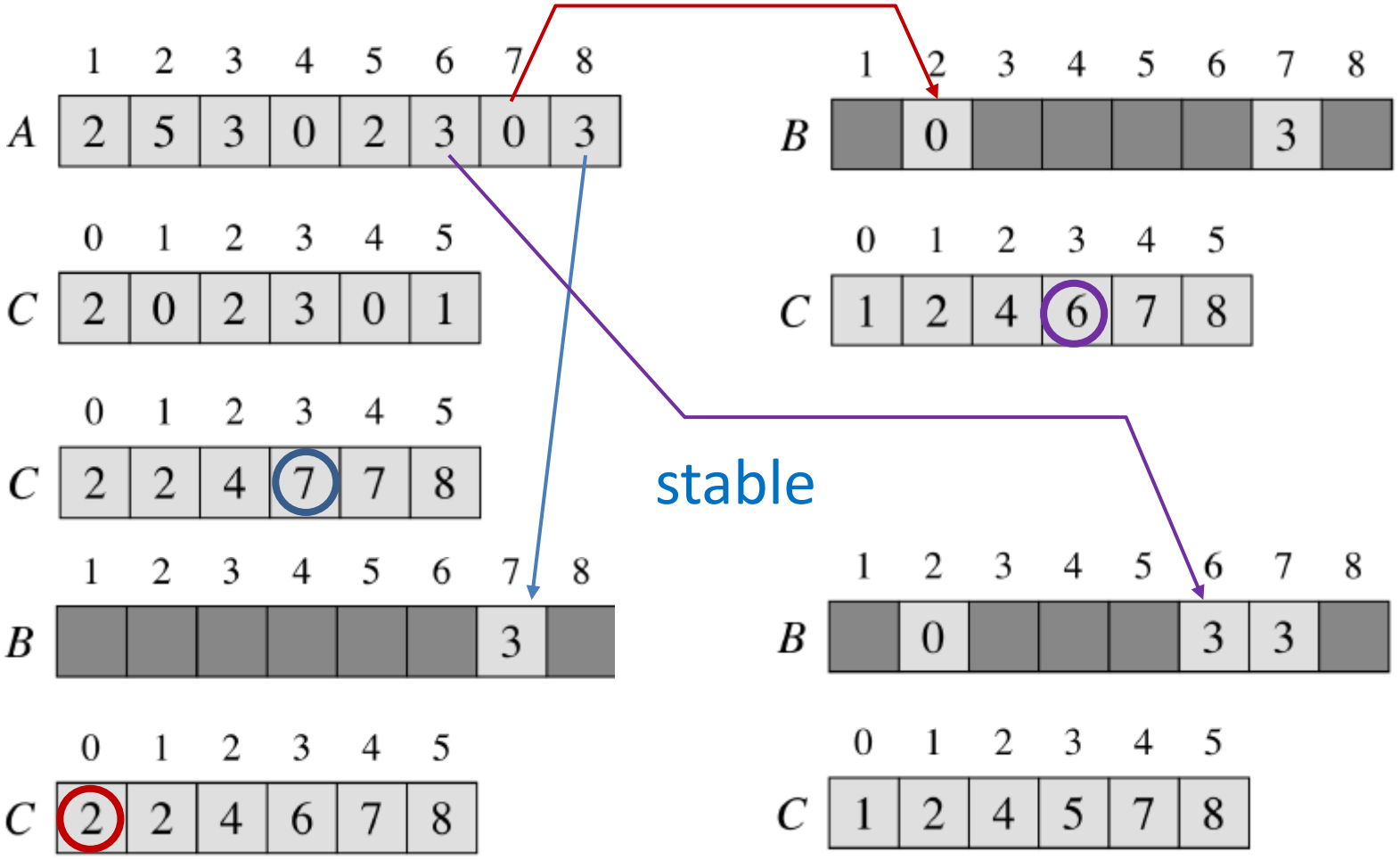
  for $j = n$ **downto** 1

  $B[C[A[j]]] = A[j]$

  $C[A[j]] = C[A[j]] - 1$

8.2 Counting sort

- Counting sort: An example



8.2 Counting sort

- Properties of counting sort
  - Counting sort is stable

    That is, input order is maintained among equal values.
  - Time complexity: $\Theta(n+k) = \Theta(\max(n,k))$

    If $k = O(n)$, then time $= \Theta(n)$.
  - How big a k is practical?
    - Good for sorting 32-bit values? No.
    - 16-bit? Probably not.
    - 8-bit? Maybe, depending on n.
    - 4-bit? Probably (unless n is really small).
  - Counting sort will be used in radix sort.

8.3 Radix sort

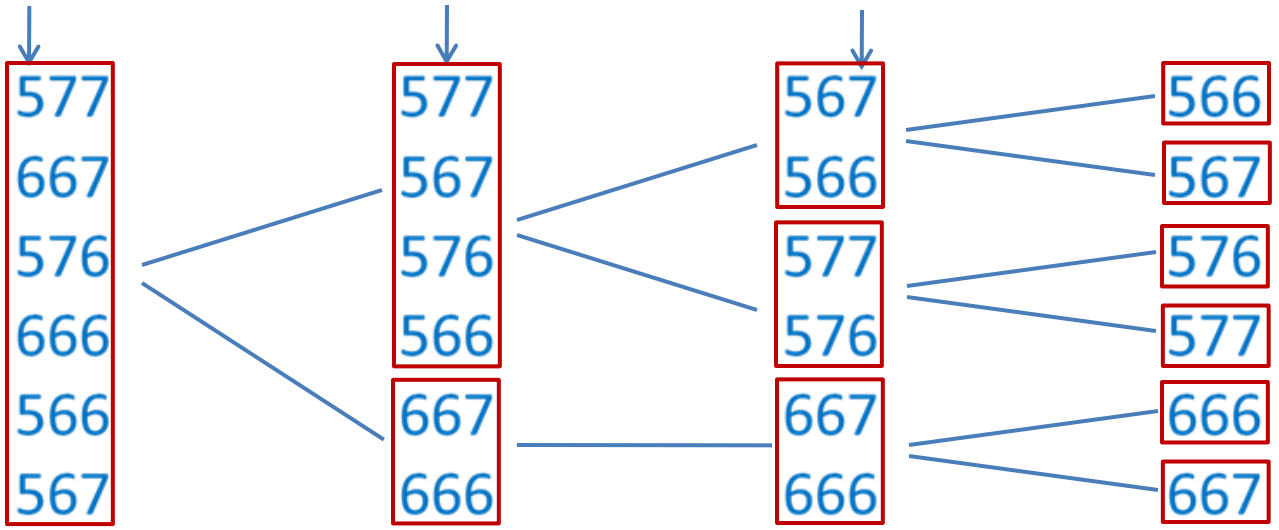
- MSD radix sort

  Assume the elements are d-digit numbers $x = x_d \ldots x_2 x_1$

  MSD radix sort – Sort the most significant digit first

  for $i = d$ **downto** 1

  for each pile having the same digit x_{i+1}, sort on digit x_i



8.3 Radix sort

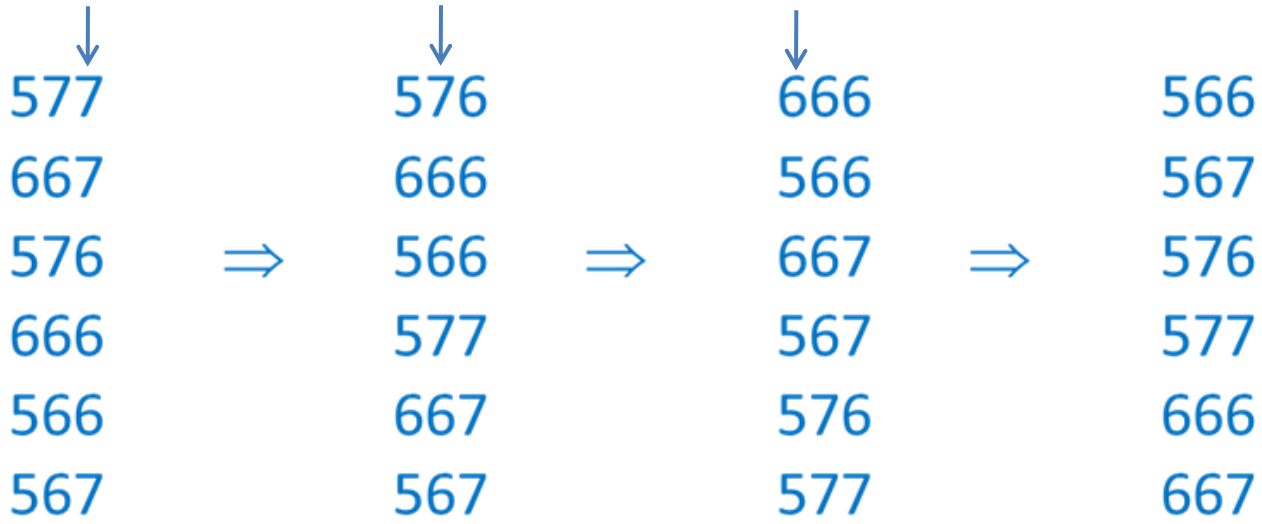
- LSD radix sort

  LSD radix sort – Sort the least significant digit first

  RADIX-SORT(A, d)

  for $i = 1$ **to** d

  use a stable sort to sort array A on digit i



8.3 Radix sort

- Correctness of LSD radix sort

  By induction on number of passes (i.e. i in pseudocode)

  Assuming that digits $1, 2, \ldots, i$ are sorted, show that a stable sort on digit $i + 1$ leaves digits $1, 2, \ldots, i, i + 1$ sorted.

  Case 1

  If 2 digits in position $i + 1$ are different, ordering by position $i + 1$ is correct, and positions $1, 2, \ldots, i$ are irrelevant.

  Case 2

  If 2 digits in position $i + 1$ are equal, numbers are already in the right order (by inductive hypothesis).

  The stable sort on digit $i + 1$ leaves them in the right order.

8.3 Radix sort

- Running time of LSD radix sort

  Assume that we use counting sort as the intermediate sort.

  - $\Theta(n+k)$ per pass (digits in range $0..k$)
  - d passes
  - $\Theta(d(n+k))$ in total
  - If $k = O(n)$, then time $= \Theta(dn)$

  How to break each key into digits?

  - n elements, b bits per element
  - Break each element into r-bit digits. Have $d = \lceil b/r \rceil$
  - Use counting sort, $k = 2^r - 1$
  - Time $= \Theta\left(\frac{b}{r}(n+2^r)\right)$

8.3 Radix sort

- Running time of LSD radix sort

  How to choose r?

  - Balance b/r and $n + 2^r$
  - Choosing $r \approx \lg n$ gives us

    $$\Theta\left(\frac{b}{\lg n}(n + 2^{\lg n})\right) = \Theta\left(\frac{b}{\lg n}(n + n)\right) = \Theta(n/\lg n)$$

  - If $r < \lg n$, then $b/r > b/\lg n$ and $n + 2^r$ doesn't improve
  - If $r > \lg n$, then $n + 2^r$ gets big,

    e.g. $r = 2\lg n \Rightarrow n + 2^{2\lg n} = n + \left(2^{\lg n}\right)^2 = n + n^2$
  - So, to sort 2^{16} 32-bit numbers, use $r = \lg 2^{16} = 16$ bits

    $\lceil b/r \rceil = \lceil 32/16 \rceil = 2$ passes

8.3 Radix sort

- Radix sort vs Merges sort/Quicksort

  To sort $2^{16} = (65536)$ 32-bit numbers

  - Radix sort
    - 4 passes over the data (2 counting sort, each 2 passes)
    - need 2 auxiliary 65536-element array
  - Merge sort/quicksort
    - $\lg 2^{16} = 16$ passes over the data
    - Mergesort needs 1 auxiliary 65536-element array

8.3 Radix sort

- Radix sort on other data types
  - Radix sort can be used with any data that can be compared bitwise, e.g. unsigned integers, signed integers, floating numbers, strings, etc.
  - Example – 2's complement integers

    Sort the last pass on signed integers, and all the other passes on unsigned integers.

    For r-bit digits, modify the last pass as follows:

    $C[A[j]] \Rightarrow C[A[j] + 2^{r-1}]$

    so that

    $C[-2^{r-1}..2^{r-1}-1] \rightarrow C[0..2^r-1]$

8.3 Radix sort

- Radix sort on other data types
  - Example – 2's complement integers (Cont'd)

0100		1 0100		1000	-8
1010		0 0000		1010	-6
0000		-1 1100		1100	-4
1110	⇒	-2 1000	⇒	1110	-2
0010		-2 1010		0000	0
1100		-1 1110		0010	2
0110		0 0010		0100	4
1000		1 0110		0110	6

+2

8.3 Radix sort

- Radix sort on other data types
  - Example – strings

    cats cat bog bogs café

    Step 1 – Sort the strings by their lengths by radix sort

    cat bog cats bogs café

    One way to do this step is to sort on the weights

    $\text{strlen}(A[i]) \times n + i, 0 \le i \le n-1$

index	0	1	2	3	4
array A	**cats**	**cat**	**bog**	**bogs**	**café**
weight	20	16	17	23	24
sorted weight	16	17	20	23	24
sorted weight mod 5	1	2	0	3	4

8.3 Radix sort

- Radix sort on other data types
  - Example – strings (Cont'd)

    Step 2 – Sort the characters backward in decreasing length, putting the shorter strings before the longer strings

	cat		café		café		bog
	bog		bog		cat		bogs
cats	café	⇒	bogs	⇒	cats	⇒	café
bogs	cats		cat		bog		cat
café	bogs		cats		bogs		cats

8.4 Bucket sort

- Bucket sort
  - Assumes the input is generated by a random process that distributes elements uniformly over $[0,1)$.
  - Idea
    - Divide $[0,1)$ into n equal-sized buckets.
    - Distribute the n input values into the buckets.
    - Sort each bucket.
    - Then go through buckets in order, listing elements in each one.

8.4 Bucket sort

- Bucket sort

  Input: $A[1..n]$, where $0 \le A[i] < 1$ for all i.

  BUCKET-SORT(A, n)

  let $B[0..n-1]$ be a new array of linked lists

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

  make $B[i]$ an empty list

  for $i = 1$ **to** n

  insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$

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

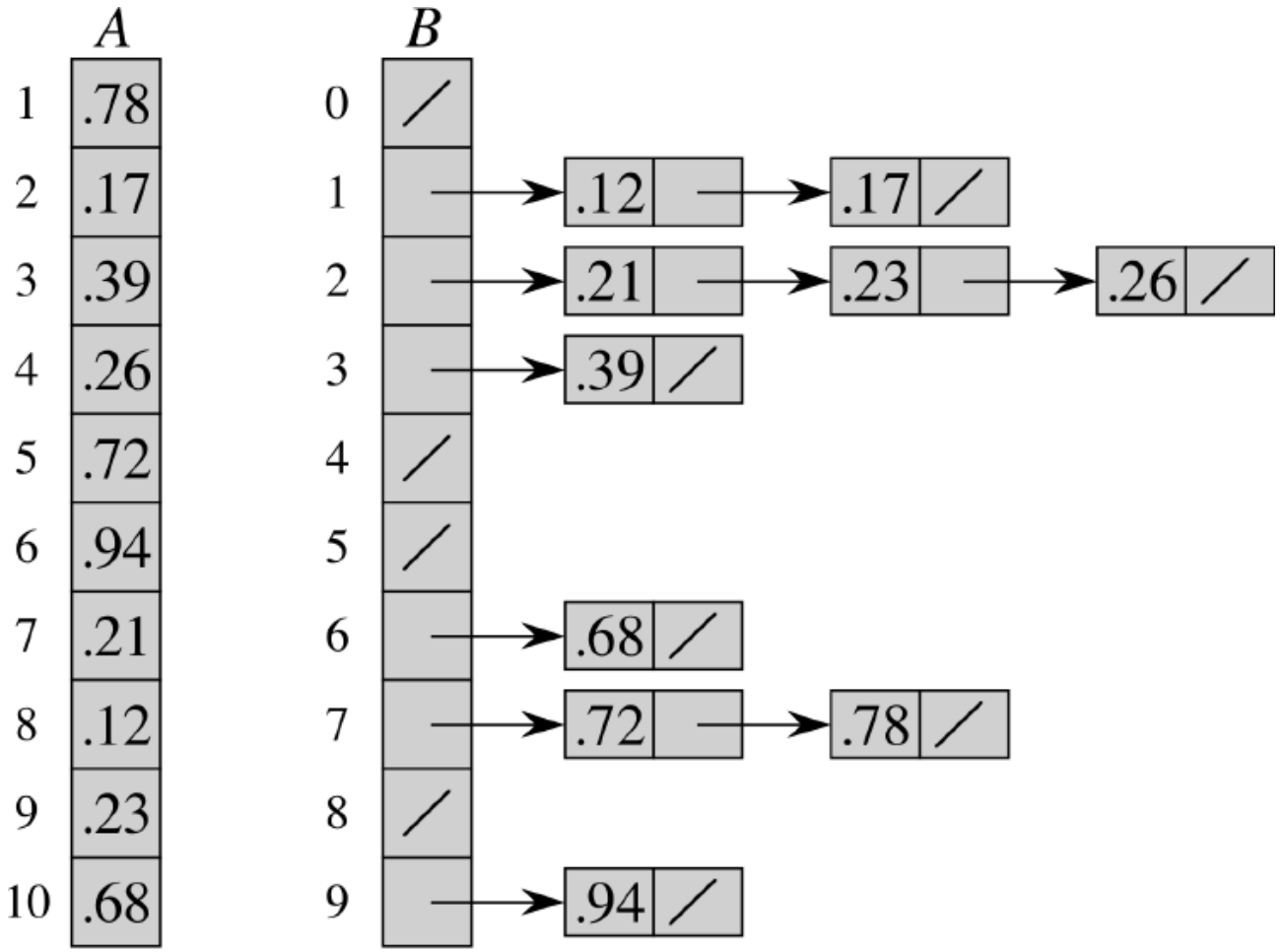
  sort list $B[i]$ with insertion sort

  concatenate lists $B[0], B[1], \ldots, B[n-1]$ together in order

  return the concatenated list

8.4 Bucket sort

- Bucket sort: An example



8.4 Bucket sort

- Correctness of bucket sort

  Consider $A[i],\ A[j]$

  Assume without loss of generality that $A[i] \leq A[j]$

  Then, $\lfloor n \cdot A[i] \rfloor \leq \lfloor n \cdot A[j] \rfloor$

  Case 1: $\lfloor n \cdot A[i] \rfloor = \lfloor n \cdot A[j] \rfloor$

  $A[i]$ is placed into the same bucket as $A[j]$.

  Insertion sort fixes up.

  Case 2: $\lfloor n \cdot A[i] \rfloor < \lfloor n \cdot A[j] \rfloor$

  $A[i]$ is placed into a bucket with a lower index.

  Concatenation of lists fixes up.

8.4 Bucket sort

- Running time of bucket sort

  Analysis
  - Relies on no bucket getting too many values.
  - All lines of algorithm except insertion sorting take $O(n)$ altogether.
  - Intuitively, if each bucket gets a constant number of elements, it takes $O(1)$ time to sort each bucket
    $\Rightarrow O(n)$ sorting time for all buckets.
  - We **expect** each bucket to have few elements, since the average is 1 element per bucket.
  - But we need to do a careful analysis.

8.4 Bucket sort

- Probabilistic analysis

  Define random variables

  $T(n)$ = the running time of Bucket sort on n elements

  n_i = the number of elements placed in bucket $B[i]$

  Then,

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

  Big-O, not Θ, $\because$ may not be the worst case of insertion sort

  CLAIM

$$\mathrm{E}[n_i^2] = 2 - 1/n \text{ for } i = 0, 1, \ldots, n-1$$

8.4 Bucket sort

- Probabilistic analysis

$$\mathrm{E}[T(n)] = \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}[O(n_i^2)] \quad \because \text{linearity of expectation}$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}[n_i^2]) \quad \because \mathrm{E}[aX] = a\mathrm{E}[X]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(2 - 1/n)$$

$$= \Theta(n) + O(n) = \Theta(n)$$

8.4 Bucket sort

- Probabilistic analysis

  Proof of **CLAIM** $\mathrm{E}[n_i^2] = 2 - 1/n$ for $i = 0, 1, \ldots, n-1$

  Define indicator random variables

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

  Then,

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

  Since $A[j]$ is distributed uniformly over $[0,1)$, $i = \lfloor n \cdot A[j] \rfloor$ is distributed uniformly over $[0, n)$.

  Therefore,

  $\mathrm{E}[X_{ij}] = \Pr\{A[j] \text{ falls in bucket } i\} = 1/n$

8.4 Bucket sort

- Probabilistic analysis

  We also have

$$\mathrm{E}[X_{ij}^2] = 0^2 \cdot \Pr\{A[j] \text{ doesn't fall in bucket } i\} + 1^2 \cdot \Pr\{A[j] \text{ does fall in bucket } i\}$$
$$= 0 \cdot (1 - 1/n) + 1 \cdot (1/n)$$
$$= \frac{1}{n}$$

  and

  X_{ij} and X_{ik}, $j \neq k$, are independent random variables

$$\Rightarrow \mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$$
$$= \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$$

8.4 Bucket sort

- Probabilistic analysis

$$\mathrm{E}[n_i^2] = \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right]$$

$$= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} X_{ij}X_{ik}\right]$$

$$= \sum_{j=1}^{n} \mathrm{E}[X_{ij}^2] + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \mathrm{E}[X_{ij}X_{ik}]$$

$$= \sum_{j=1}^{n} \frac{1}{n} + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \frac{1}{n^2} = n\cdot\frac{1}{n} + 2\binom{n}{2}\frac{1}{n^2} = 2 - \frac{1}{n}$$