Algorithms

Searching and Sorting

Ming-Hsien Tsai

Academia Sinica
National Taiwan University

Based on: U. Manber. Introduction to Algorithms - A Creative Approach. Addison-Wesley, 1989.
Also based on slides made by Professor Yih-Kuen Tsay.

Searching a Sorted Sequence

Problem

Let $x_1, x_2, \ldots, x_n$ be a sequence of real numbers such that $x_1 \leq x_2 \leq \cdots \leq x_n$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

Idea: cut the search space in half by asking only one question

$$\begin{cases} T(1) = O(1) \\ T(n) = T(n/2) + O(1),\ n \geq 2 \end{cases}$$

Time complexity: O(log n)

Binary Search

```
Algorithm Binary_Search (X, n, z);
begin
    Position := Find(z, 1, n);
end
```

```
function Find (z, Left, Right) : integer;
begin
    if Left = Right then
        if X[Left] = z then Find := Left
        else Find := 0
    else
        Middle := ⌈(Left + Right) / 2⌉ ;

        if z < X[Middle] then Find := Find(z, Left, Middle - 1)
        else Find := Find(z, Middle, Right)
end
```

Searching a Cyclically Sorted Sequence

Problem

Given a cyclically sorted list, find the position of the minimal element in the list (we assume, for simplicity, that this position is unique)

Position:	1	2	3	4	5	6	7	8
List: [	5	6	7	0	1	2	3	4]

Position:	1	2	3	4	5	6	7	8
List: [	0	1	2	3	4	5	6	7]

Searching a Cyclically Sorted Sequence

Problem

Given a cyclically sorted list, find the position of the minimal element in the list (we assume, for simplicity, that this position is unique)

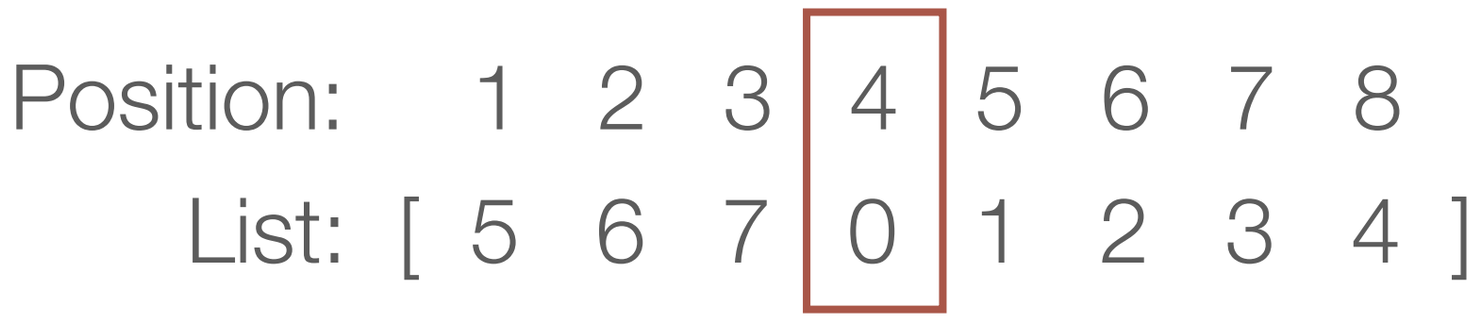
Position:	1	2	3	4	5	6	7	8
List: [	5	6	7	0	1	2	3	4]

Position:	1	2	3	4	5	6	7	8
List: [	0	1	2	3	4	5	6	7]

Searching a Cyclically Sorted Sequence

Problem

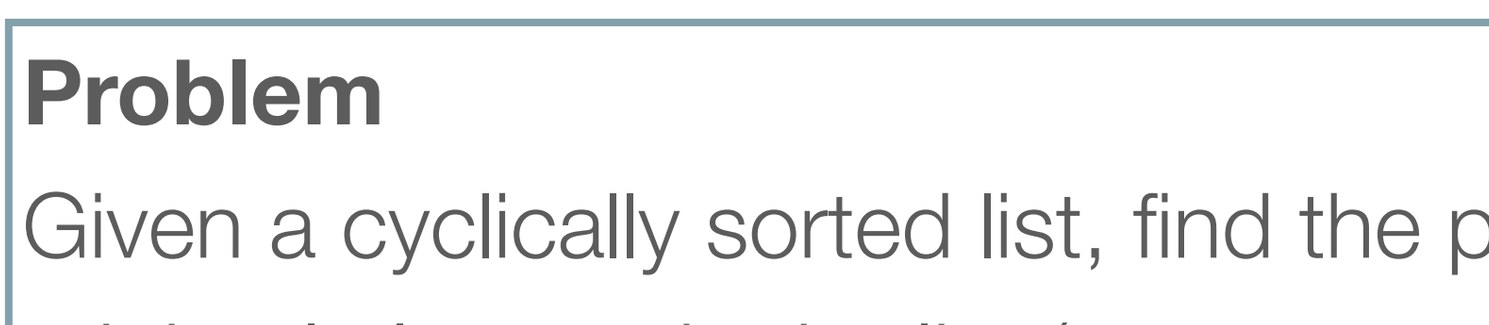
Given a cyclically sorted list, find the position of the minimal element in the list (we assume, for simplicity, that this position is unique)



Searching a Cyclically Sorted Sequence

Problem

Given a cyclically sorted list, find the position of the minimal element in the list (we assume, for simplicity, that this position is unique)



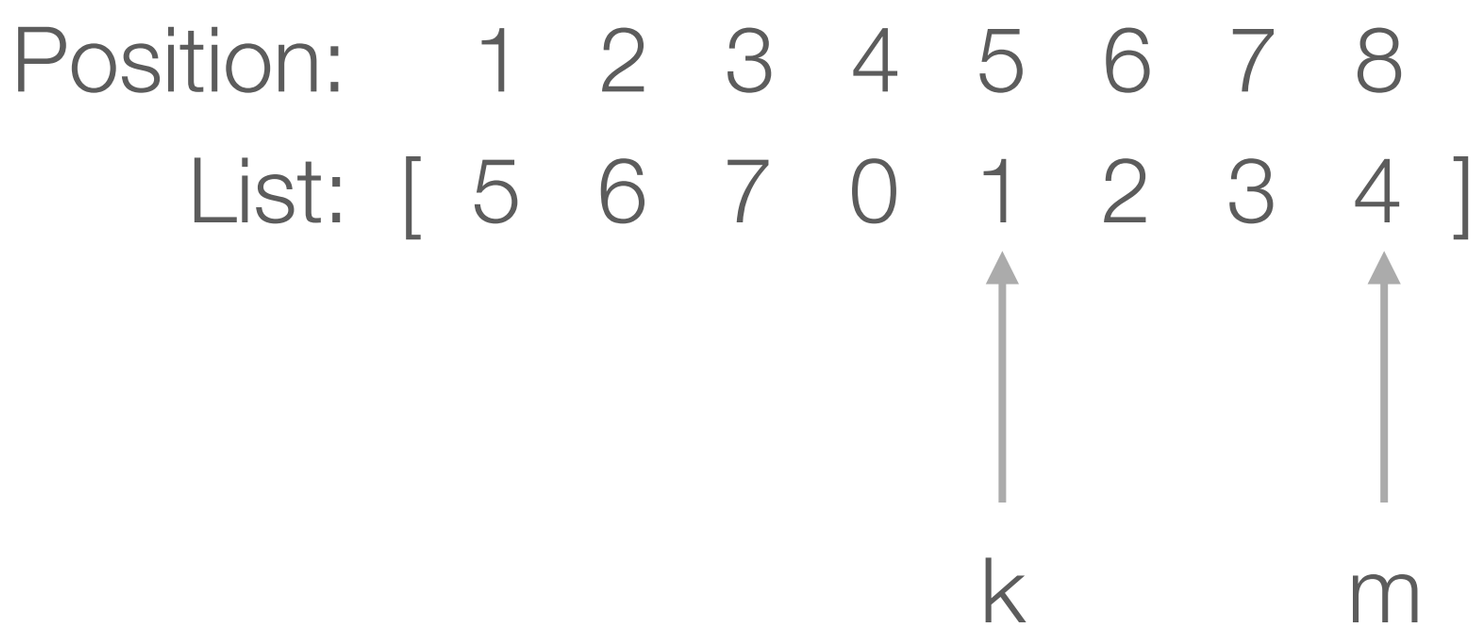
To cut the search space in half, what question should we ask?

Searching a Cyclically Sorted Sequence (cont'd)

Assume x_i is the minimal element in the sequence.

Take two numbers x_k and x_m, such that $k < m$.

Case 1: $x_k < x_m$, i cannot be in the range $k < i \leq m$

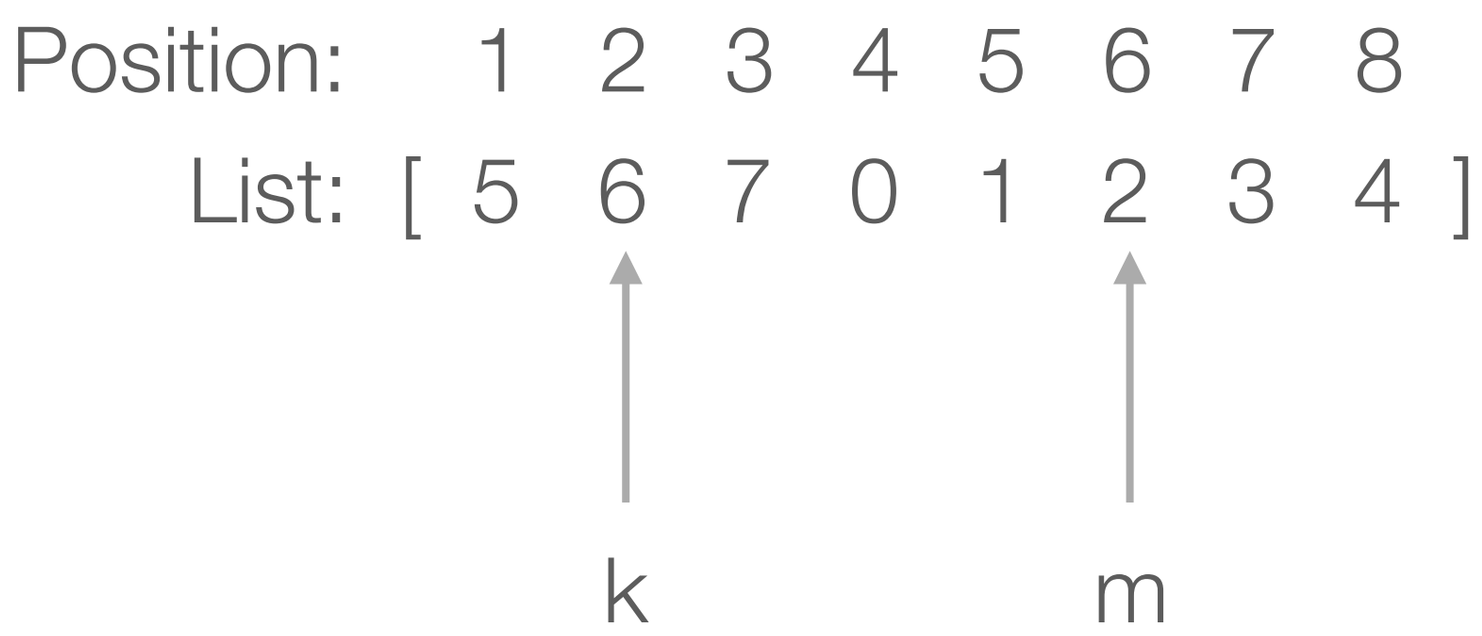


Searching a Cyclically Sorted Sequence (cont'd)

Assume x_i is the minimal element in the sequence.

Take two numbers x_k and x_m, such that $k < m$.

Case 2: $x_k \geq x_m$, i must be in the range $k < i \leq m$

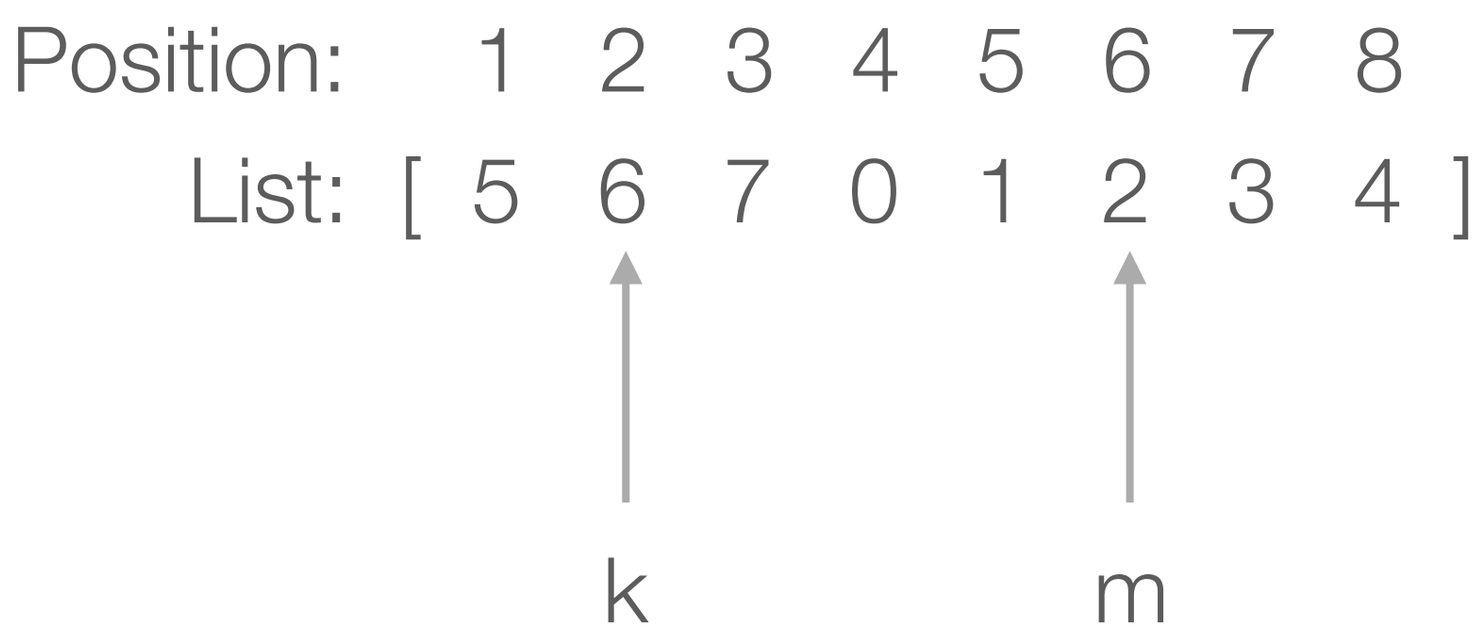


Searching a Cyclically Sorted Sequence (cont'd)

Assume x_i is the minimal element in the sequence.

Take two numbers x_k and x_m, such that $k < m$.

Case 2: $x_k \geq x_m$, i must be in the range $k < i \leq m$



How to choose x_k and x_m?

Cyclic Binary Search

```
Algorithm Cyclic_Binary_Search (X, n);
begin
    Position := Cyclic_Find(1, n);
end
```

```
function Cyclic_Find (Left, Right) : integer;
begin
    if Left = Right then Cyclic_Find := Left
   else
        Middle := ⌊(Left + Right) / 2⌋ ;
        if X[Middle] < X[Right] then Cyclic_Find := Cyclic_Find(Left, Middle)
        else Cyclic_Find := Cyclic_Find(Middle + 1, Right)
end
```

Searching for a Special Index

Problem

Give a sorted sequence of ***distinct*** integers $a_1, a_2, \ldots, a_n$, determine whether there exists an index i such that $a_i = i$

Position: 1 2 3 4 5 6 7 8

List: [-1 1 2 4 5 6 8 9]

Position: 1 2 3 4 5 6 7 8

List: [-1 1 2 5 6 9 9 10]

Searching for a Special Index

Problem

Give a sorted sequence of ***distinct*** integers $a_1, a_2, \ldots, a_n$, determine whether there exists an index i such that $a_i = i$

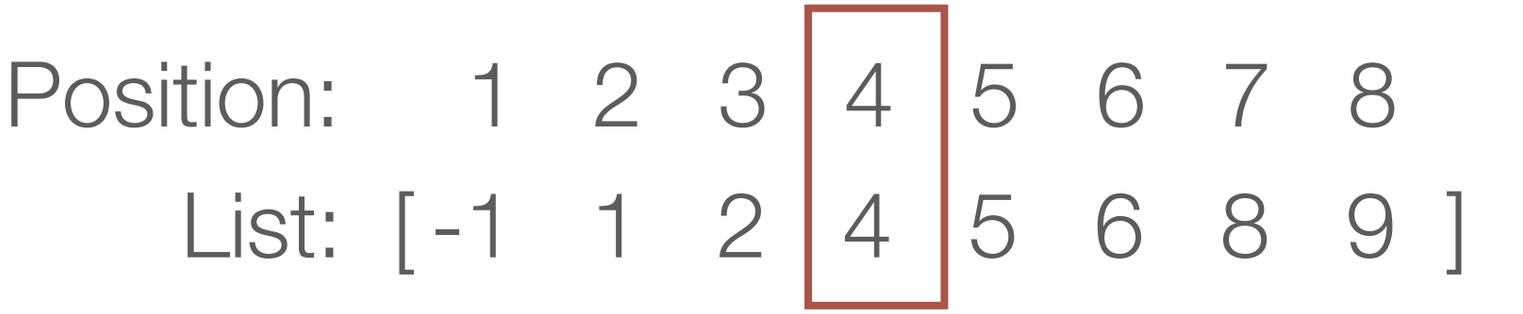
Position:	1	2	3	4	5	6	7	8
List: [	-1	1	2	4	5	6	8	9]

Position:	1	2	3	4	5	6	7	8
List: [	-1	1	2	5	6	9	9	10]

Searching for a Special Index

Problem

Give a sorted sequence of ***distinct*** integers $a_1, a_2, \ldots, a_n$, determine whether there exists an index i such that $a_i = i$



Can we cut the search space in half by asking only one question?

Searching for a Special Index (cont'd)

Position:	1	2	3	4	5	6	7	8	
List: [	0	1	2	4	6	7	8	9	]

$a_3 = 2 < 3$

Position:	1	2	3	4	5	6	7	8	
List: [	0	1	2	4	6	7	8	9	]

$a_5 = 6 > 5$

Special Binary Search

```
Algorithm Cyclic_Binary_Search (A, n);
begin
    Position := Special_Find(1, n);
end
```

```
function Special_Find (Left, Right) : integer;
begin
    if Left = Right then
        if A[Left] = Left then Special_Find := Left
        else Special_Find := 0
    else
        Middle := ⌊(Left + Right) / 2⌋ ;

        if A[Middle] < Middle then Special_Find := Special_Find(Middle + 1, Right)
        else Special_Find := Special_Find(Left, Middle)
end
```

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

How do we choose x_i and x_j?

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

How do we choose x_i and x_j?

x_1

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

How do we choose x_i and x_j?

x_1 x_2

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

How do we choose x_i and x_j?

x_1 x_2 x_{2^2}

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

How do we choose x_i and x_j?

x_1 x_2 x_{2^2} x_{2^3}

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

How do we choose x_i and x_j?

x_1 x_2 x_{2^2} x_{2^3} x_{2^4}

Searching Sequences of Unknown Size

Problem

Let $x_1, x_2, \ldots$ be a sequence of real numbers such that $x_i \leq x_{i+1}$ for all $i \geq 1$. Give a real number z, we want to find whether z appears in the sequence, and, if it does, to find an index i such that $x_i = z$.

What subsequence should we search in?

Between some x_i and x_j where $x_i \leq z \leq x_j$

How do we choose x_i and x_j?

$$x_1 \quad x_2 \quad x_{2^2} \quad x_{2^3} \quad x_{2^4} \quad \ldots \quad x_{2^a} \quad x_{2^{a+1}}$$

Stuttering Subsequence

Problem

Given two sequences $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$, find the maximal value of i such that B^i is a subsequence of A

If B = xyzzx, then B^2 = xxyyzzzzxx, B^3 = xxxyyyzzzzzzxxx

B is a ***subsequence*** of A if we can embed B inside A in the same order but with possible holes

For example, B^2 = xxyyzzzzxx is a subsequence of xxzzyyyyxxzzzzzxxx

Stuttering Subsequence (cont'd)

Problem

Given two sequences $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$, find the maximal value of i such that B^i is a subsequence of A

If B^j is a subsequence of A, then B^i is a subsequence of A, for $1 \leq i \leq j$

Stuttering Subsequence (cont'd)

Problem

Given two sequences $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$, find the maximal value of i such that B^i is a subsequence of A

If B^j is a subsequence of A, then B^i is a subsequence of A, for $1 \leq i \leq j$

Does the value of i have a boundary?

Stuttering Subsequence (cont'd)

Problem

Given two sequences $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$, find the maximal value of i such that B^i is a subsequence of A

If B^j is a subsequence of A, then B^i is a subsequence of A, for $1 \le i \le j$

Does the value of i have a boundary?

The maximum value of i cannot exceed $\lfloor n/m \rfloor$ (or B^i would be longer than A)

Stuttering Subsequence (cont'd)

- Two ways to find the maximum i:
  - Sequential search (try 1, 2, 3, etc. sequentially)
    - Time complexity: O(nj) where j is the maximum value of i
  - Binary search (between 1 and $\lfloor n/m \rfloor$)
    - Time complexity: O(n log(n / m))

Stuttering Subsequence (cont'd)

- A general technique to find a maximum i that satisfies some property
  - Find an algorithm that determines whether a given i satisfies that property
  - Use binary search if we have an upper bound for i
  - Otherwise, use the doubling scheme

Interpolation Search

- In binary search, the search space is always cut in half
- When we find that z should be in between x_i and x_j where x_i is very close to z, shall we still cut the search space in half?

Interpolation Search

- In binary search, the search space is always cut in half
- When we find that z should be in between x_i and x_j where x_i is very close to z, shall we still cut the search space in half?

pages from 1 to 800

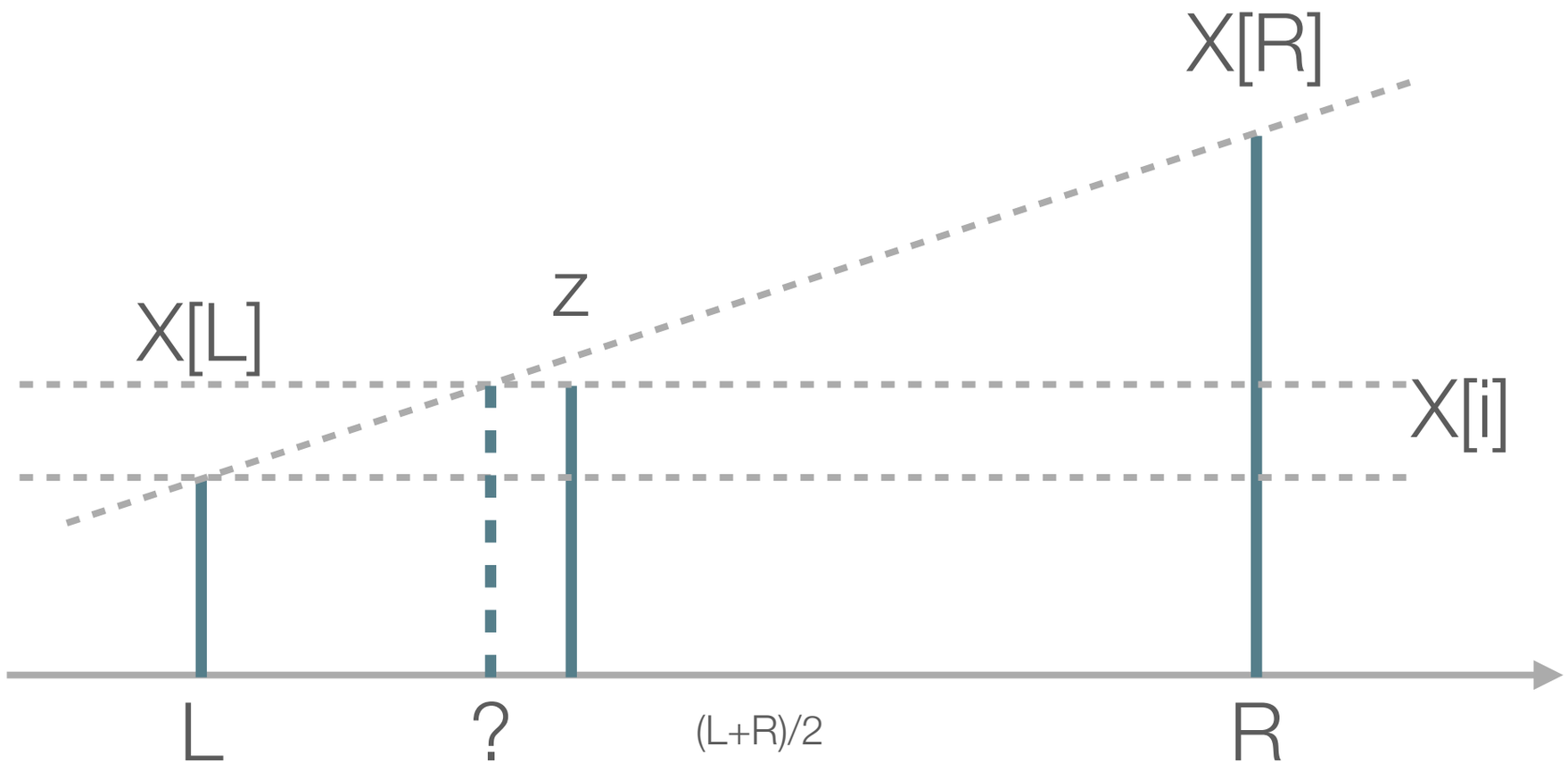
Interpolation Search

- In binary search, the search space is always cut in half
- When we find that z should be in between x_i and x_j where x_i is very close to z, shall we still cut the search space in half?

pages from 1 to 800

We can use ***interpolation*** to cut the search space by an amount that seems the most likely to succeed

Interpolation Search (cont'd)



$$\frac{R - L}{X[R] - X[L]} = \frac{? - L}{z - X[L]} \Rightarrow ? = L + \frac{(z - X[L])(R - L)}{X[R] - X[L]}$$

Interpolation Search (cont'd)

```
Algorithm Interpolation_Search (X, n, z);
begin
    if z < X[1] or z > X[n] then Position := 0
    else Position := Int_Find(z, 1, n);
end
```

```
function Int_Find (z, Left, Right) : integer;
begin
    if X[Left] = z then Int_Find := Left
    else if Left = Right or X[Left] = X[Right] then Int_Find := 0
    else
        Next_Guess := ⌈Left + (z - X[Left])(Right - Left) / (X[Right] - X[Left])⌉ ;

        if z < X[Next_Guess] then Int_Find := Int_Find(z, Left, Next_Guess - 1)
        else Int_Find := Int_Find(z, Next_Guess, Right)
end
```

Interpolation Search (cont'd)

- Interpolation search is efficient for inputs consisting of relatively uniformly distributed elements

- The average number of comparisons performed by interpolation search is O(log log n)

- But interpolation search is not much better than binary search in practice

  - Unless n is very large, the value of log n is small enough

  - Interpolation search requires more elaborate arithmetic

Sorting

Problem

Given n numbers x_1, x_2, …, x_n, arrange them in increasing order. In other words, find a sequence of distinct indices $1 \le i_1, i_2, \ldots, i_n \le n$, such that $x_{i_1} \le x_{i_2} \le \cdots \le x_{i_n}$

A sorting algorithm is called ***in-place*** if no additional work space is used besides the initial array that holds the elements

Using Binary Search Trees

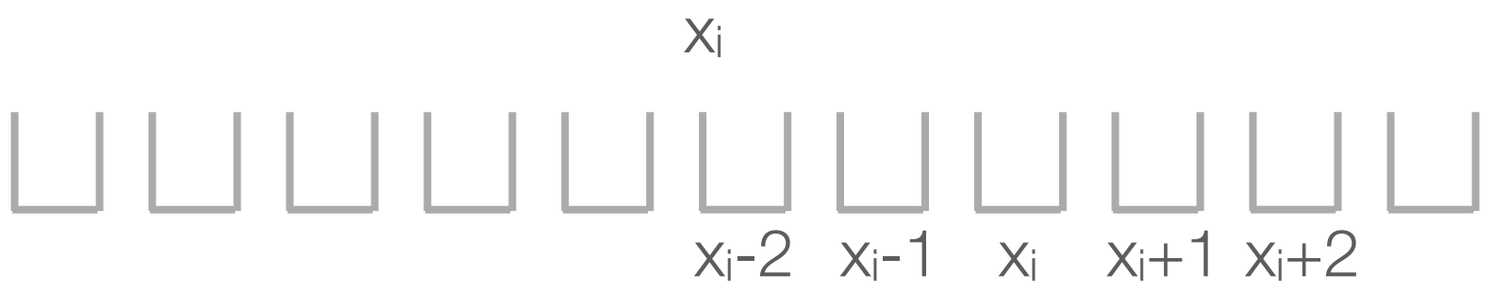
- Binary search trees, such as AVL trees, may be used for sorting

    1. Create an empty tree
    2. Insert the numbers one by one to the tree
    3. Traverse the tree and output the numbers

- What is the time complexity? Suppose we use an AVL tree

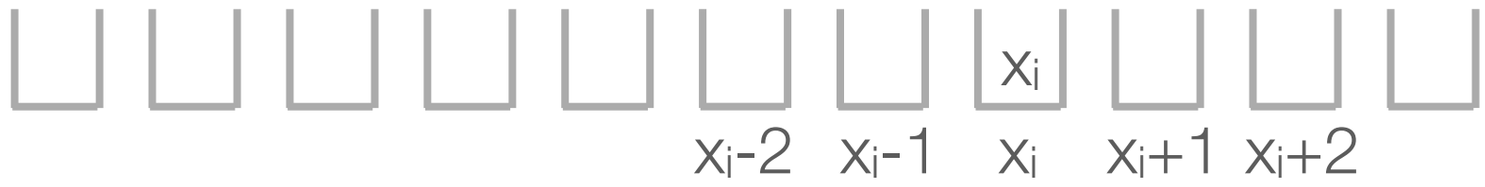
Bucket Sort

- Inputs: n integers $x_1, x_2, \ldots, x_n$ where $1 \le x_i \le m$ for $1 \le i \le n$
- Bucket sort:
  - Allocate m buckets
  - For each i, put x_i in the bucket corresponding to its value
  - Scan the buckets in order and collect all the integers
- Complexity: O(m + n)



Bucket Sort

- Inputs: n integers x_1, x_2, …, x_n where $1 \le x_i \le m$ for $1 \le i \le n$
- Bucket sort:
  - Allocate m buckets
  - For each i, put x_i in the bucket corresponding to its value
  - Scan the buckets in order and collect all the integers
- Complexity: O(m + n)



Radix Sort

- Observe how letters are delivered

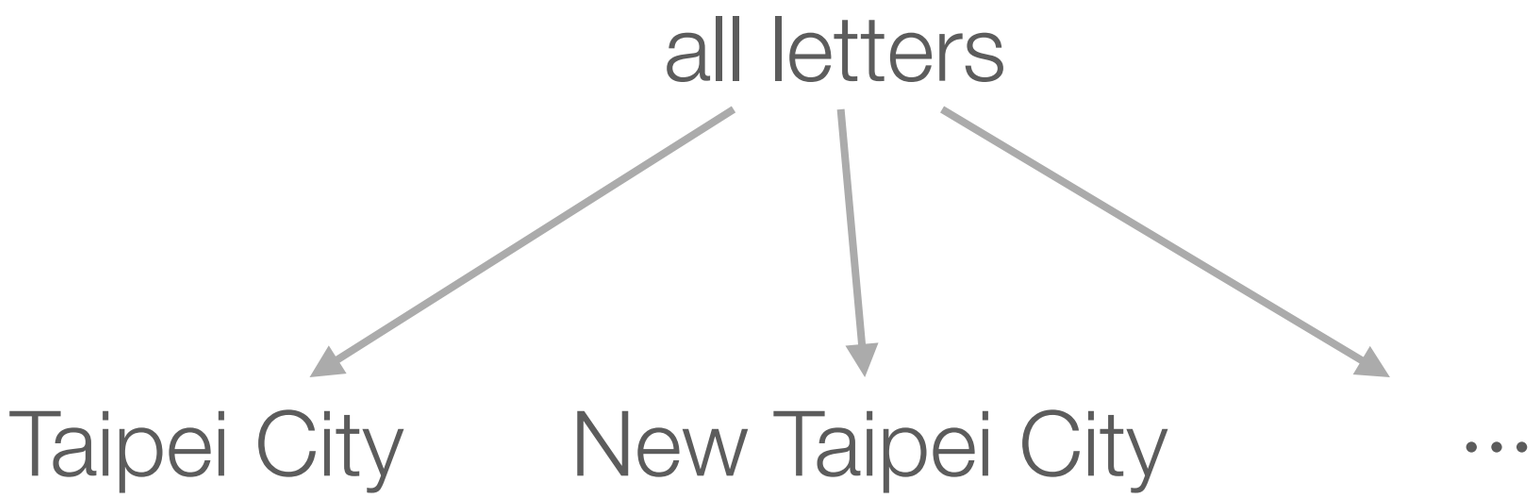
Radix Sort

- Observe how letters are delivered

all letters

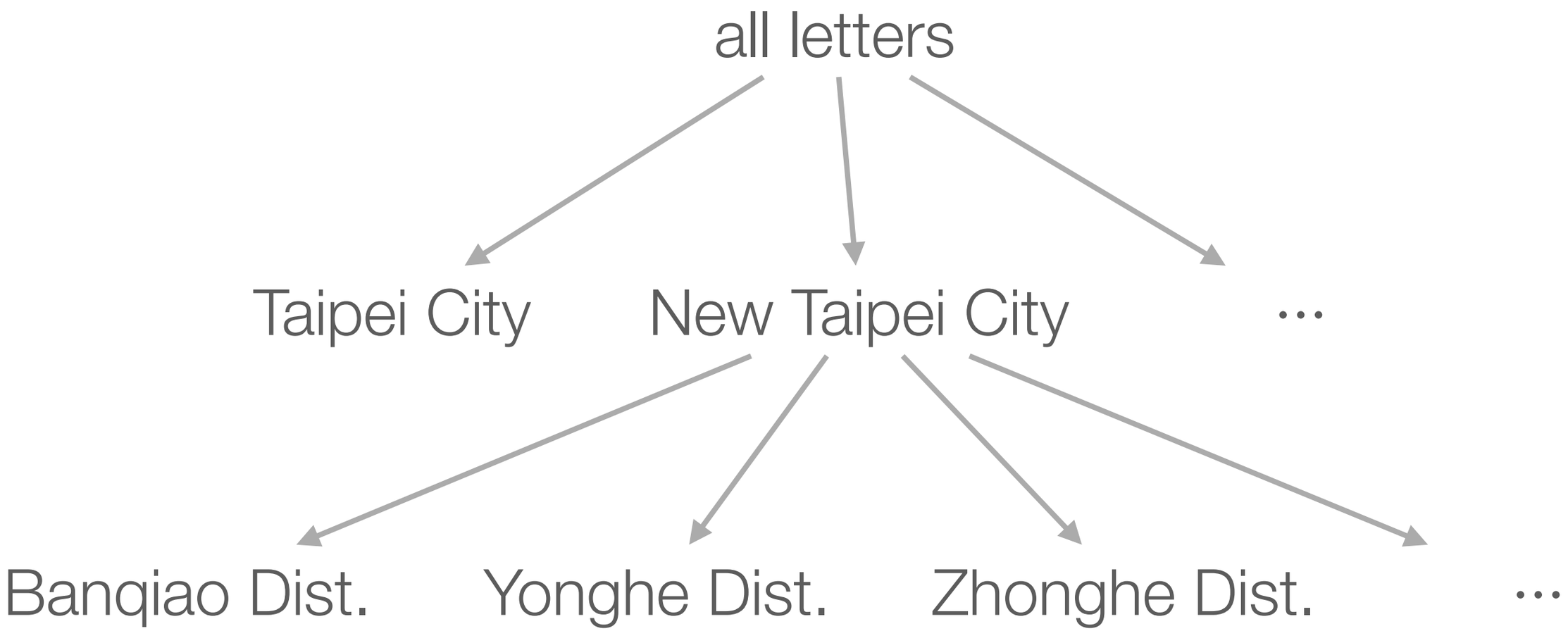
Radix Sort

- Observe how letters are delivered



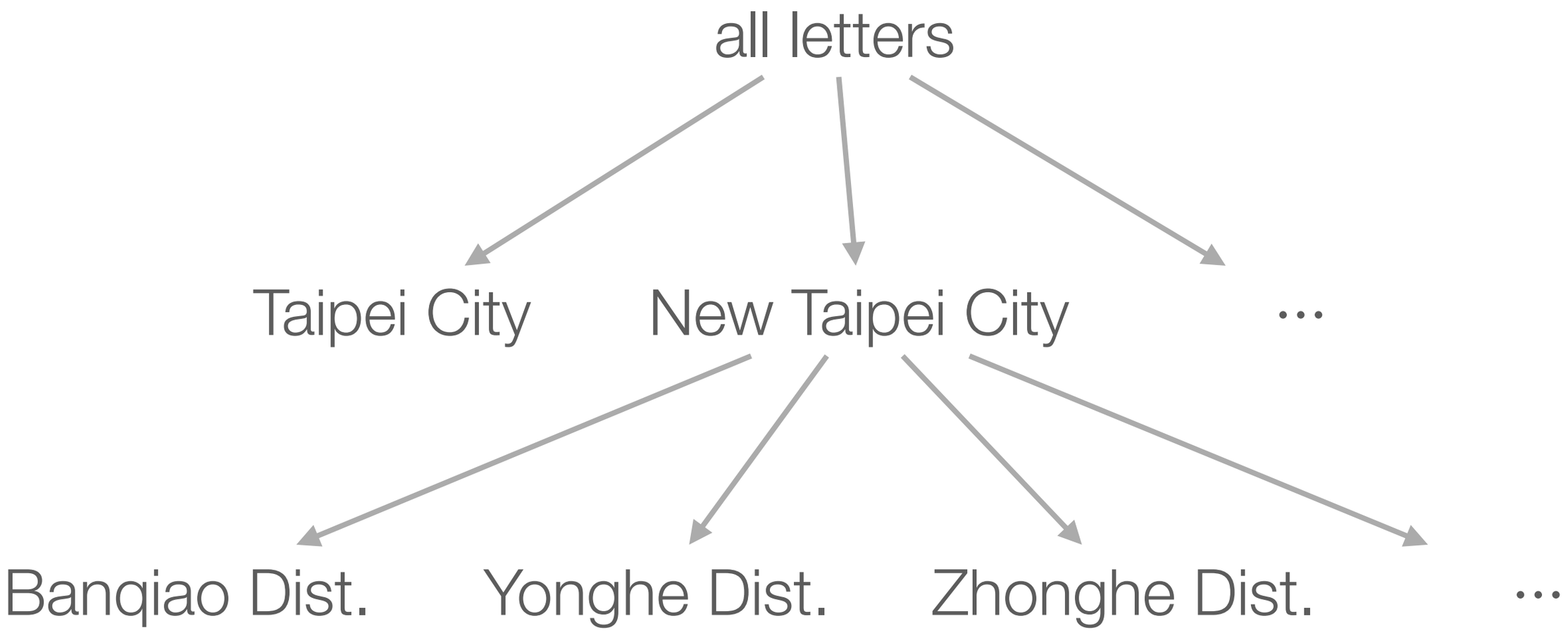
Radix Sort

- Observe how letters are delivered



Radix Sort

- Observe how letters are delivered



Can we sort numbers in a similar way?

Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD

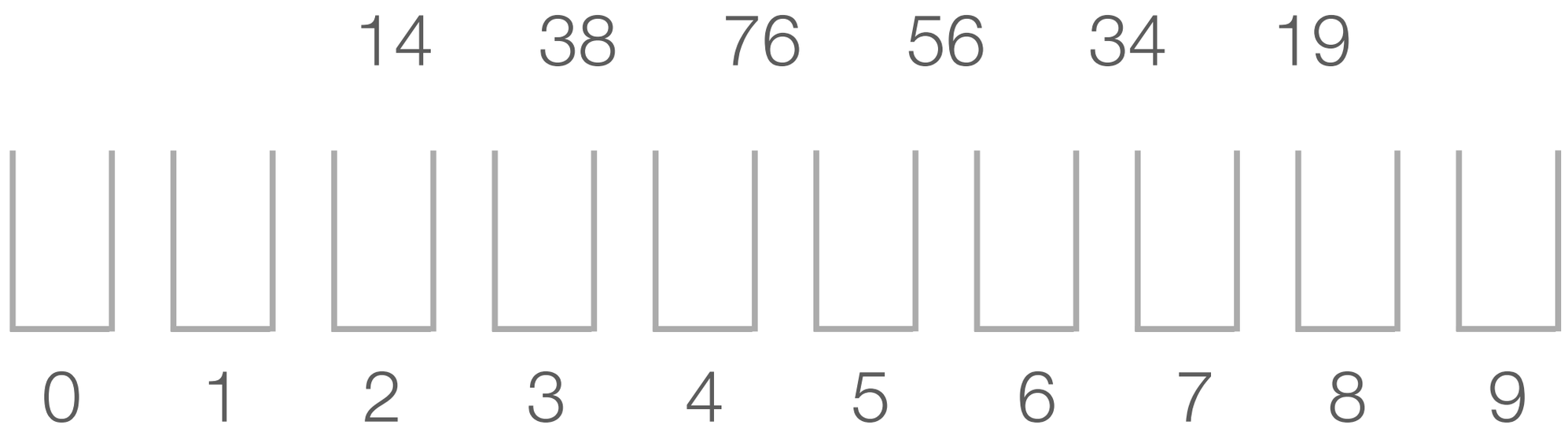
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD

14 38 76 56 34 19

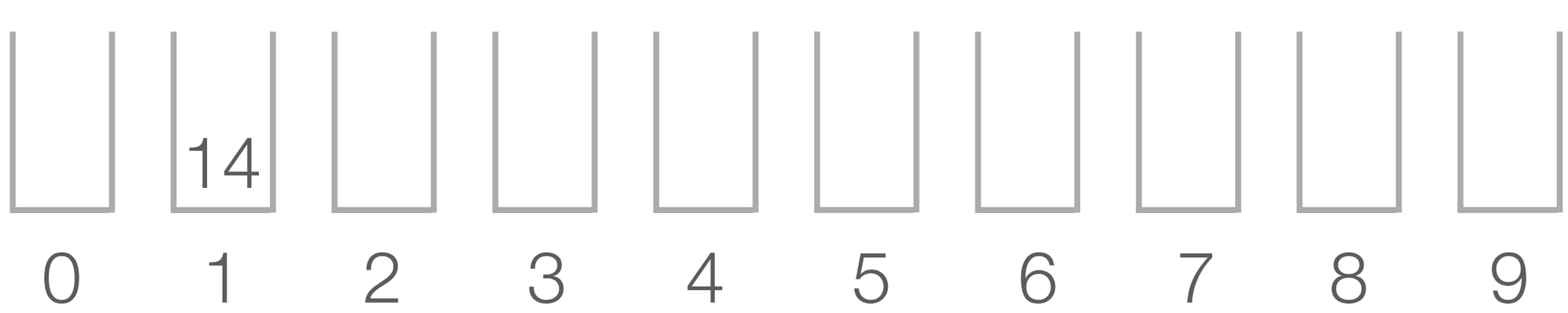
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



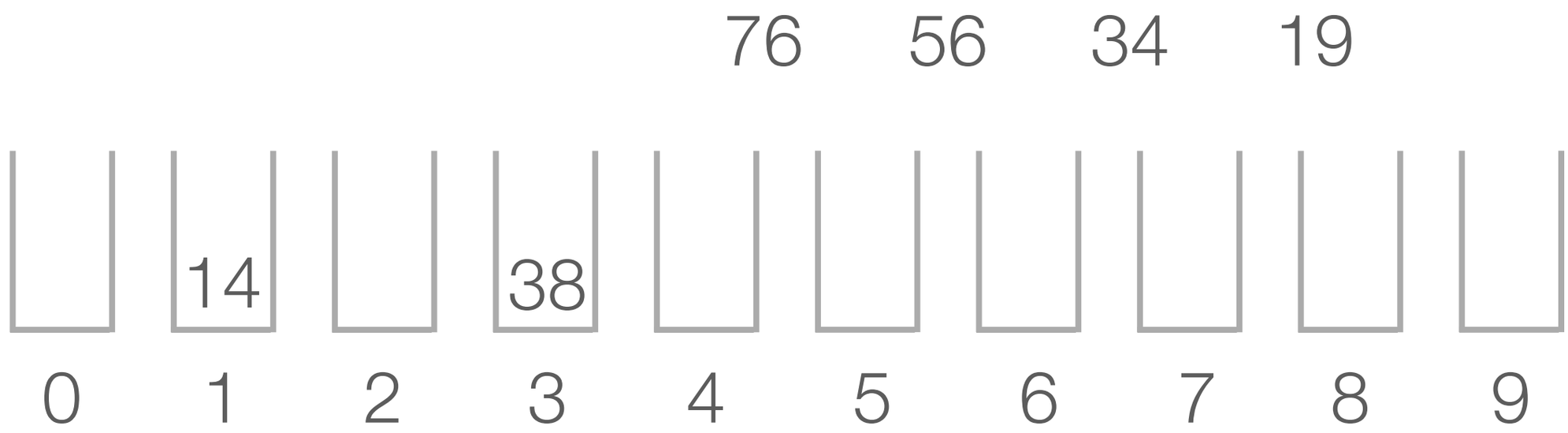
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



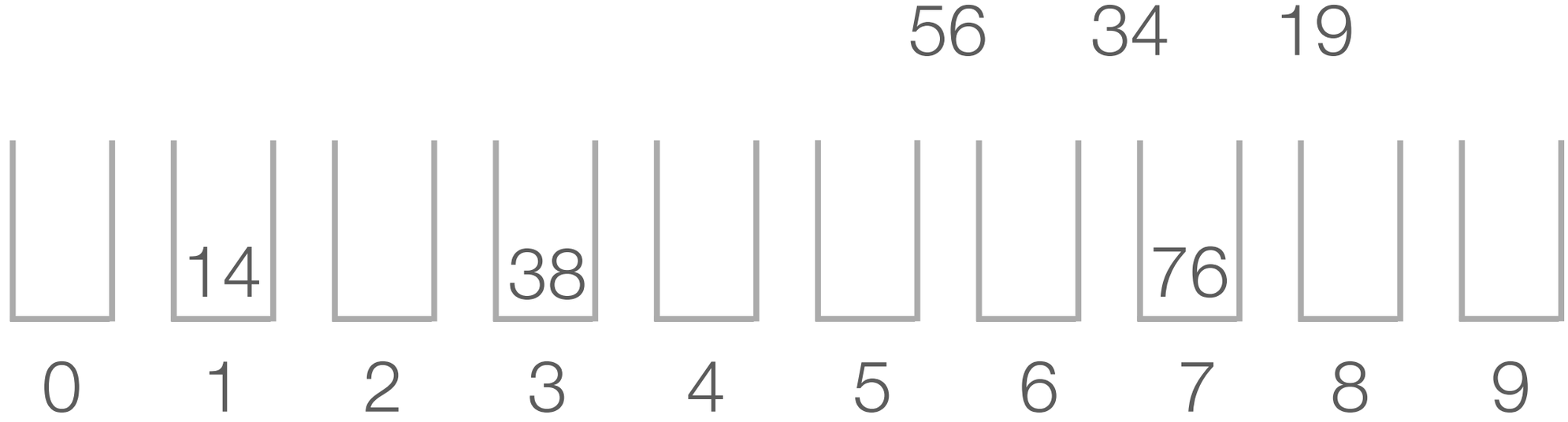
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



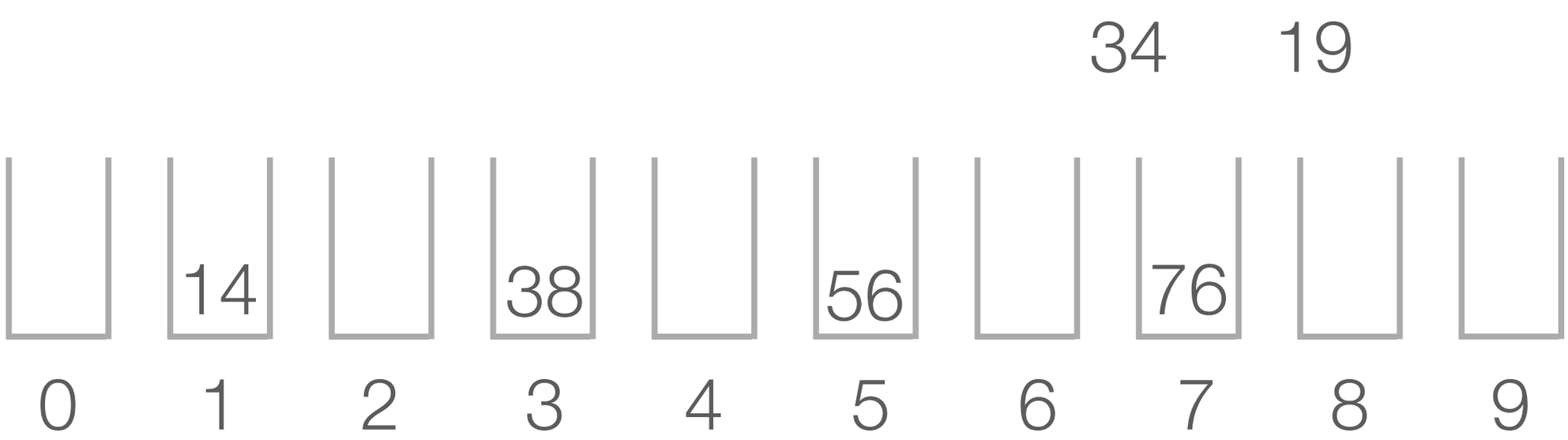
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



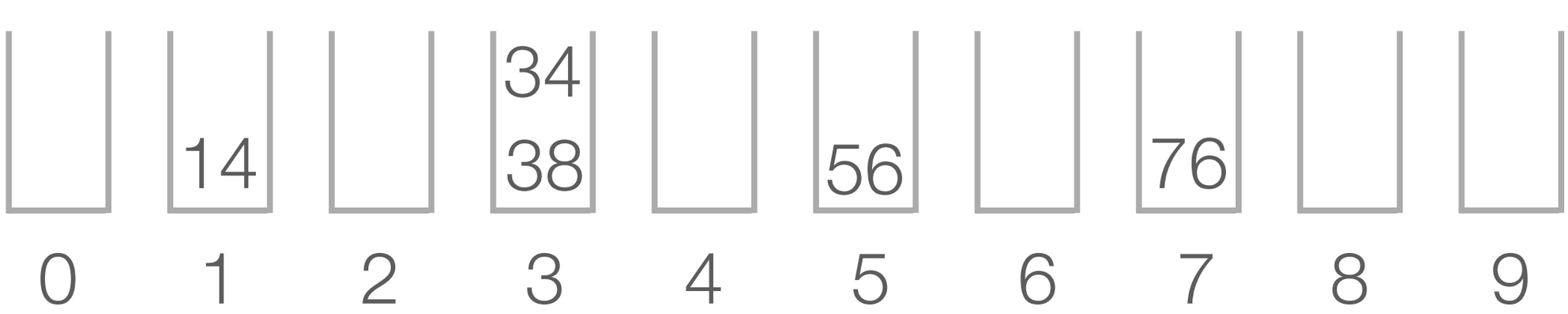
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



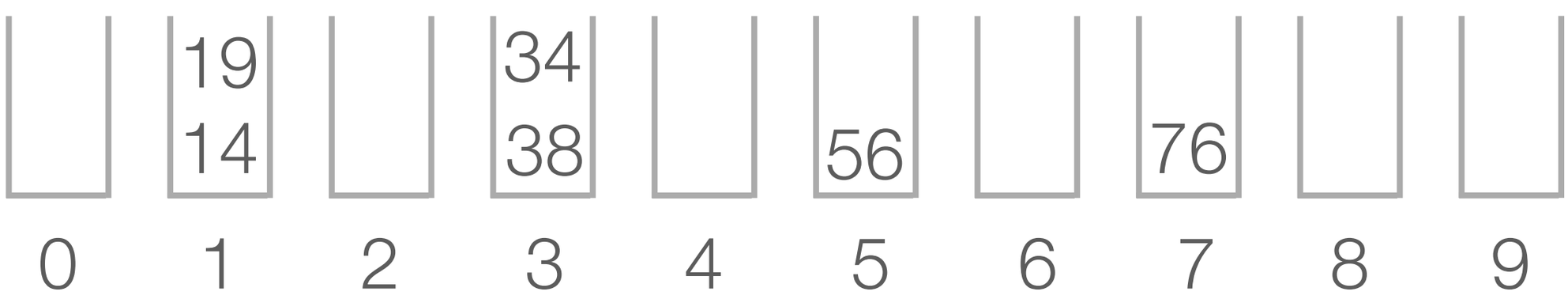
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



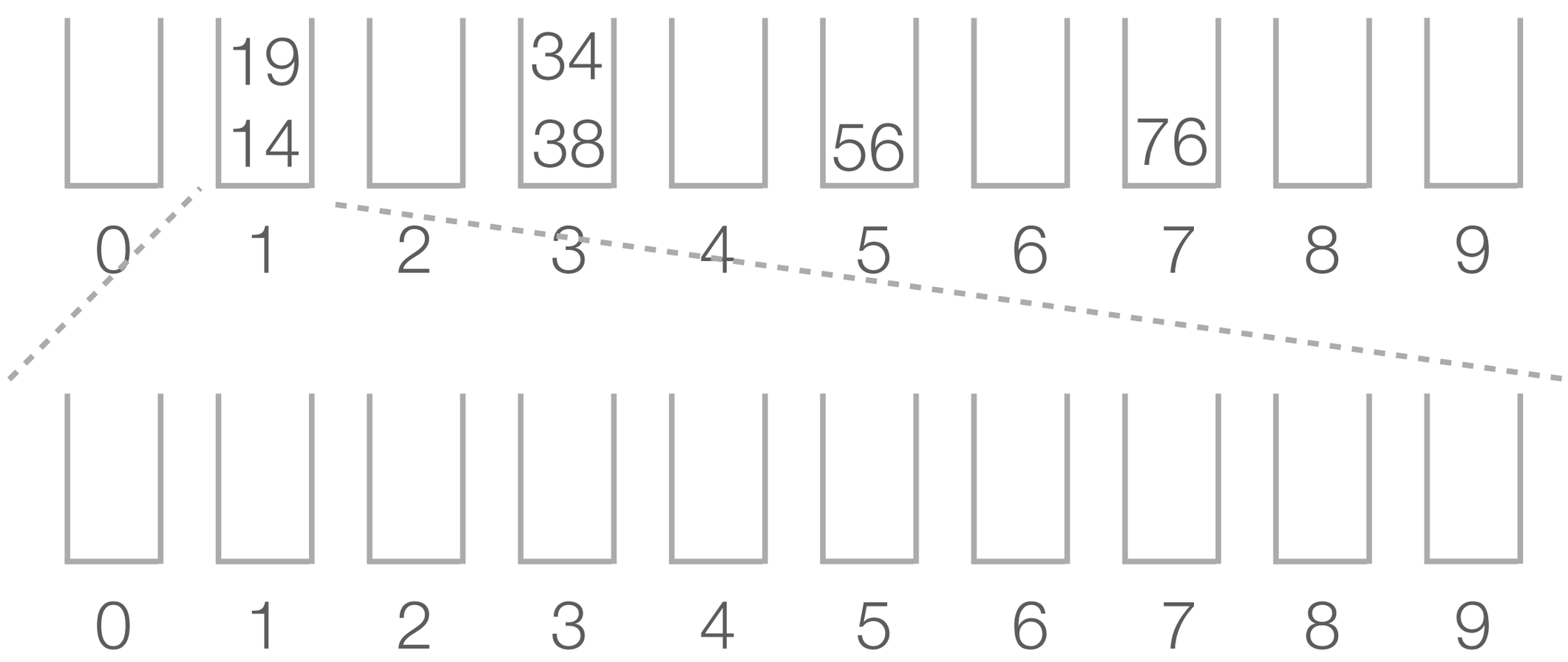
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



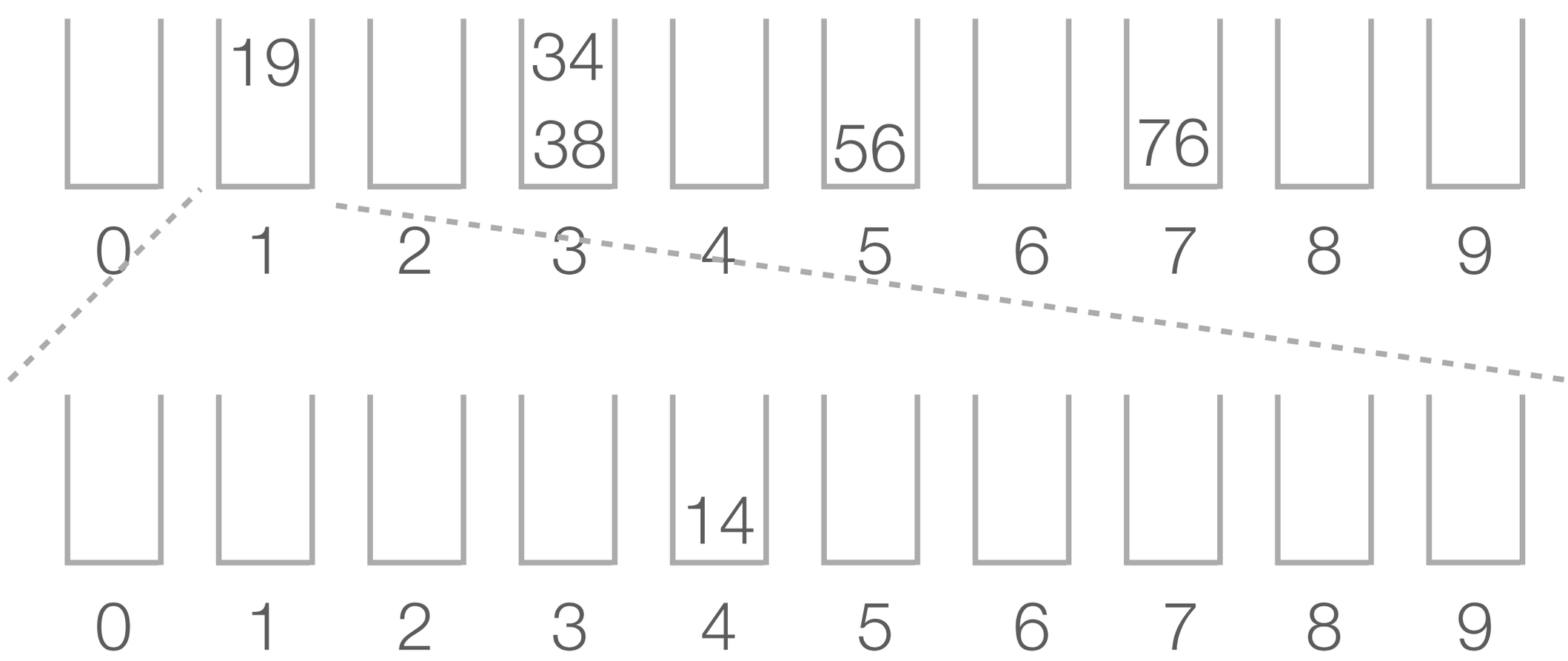
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



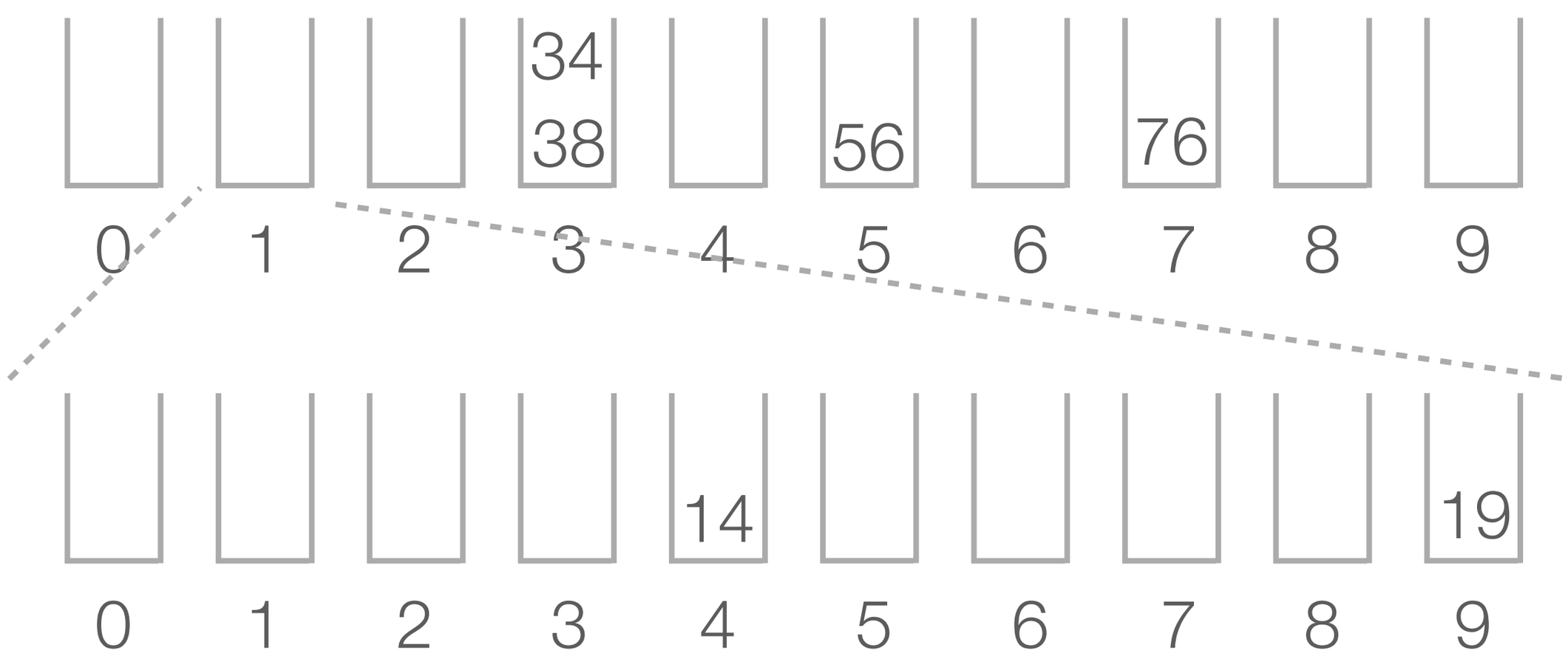
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



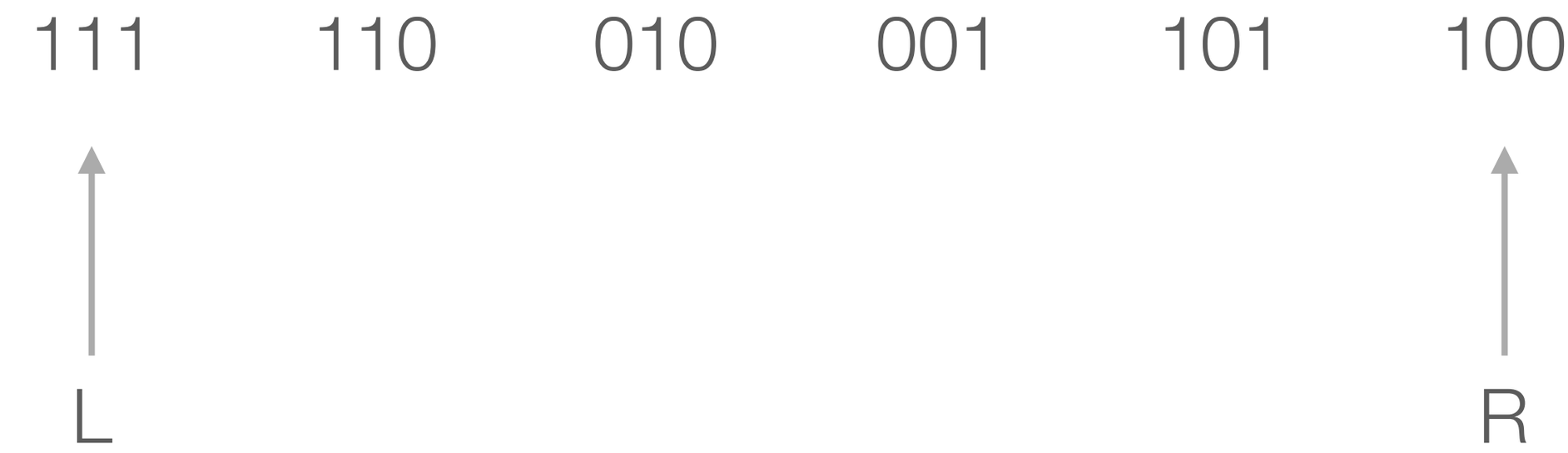
Radix Sort (cont'd)

- First idea
  - Sort digit-by-digit from MSD



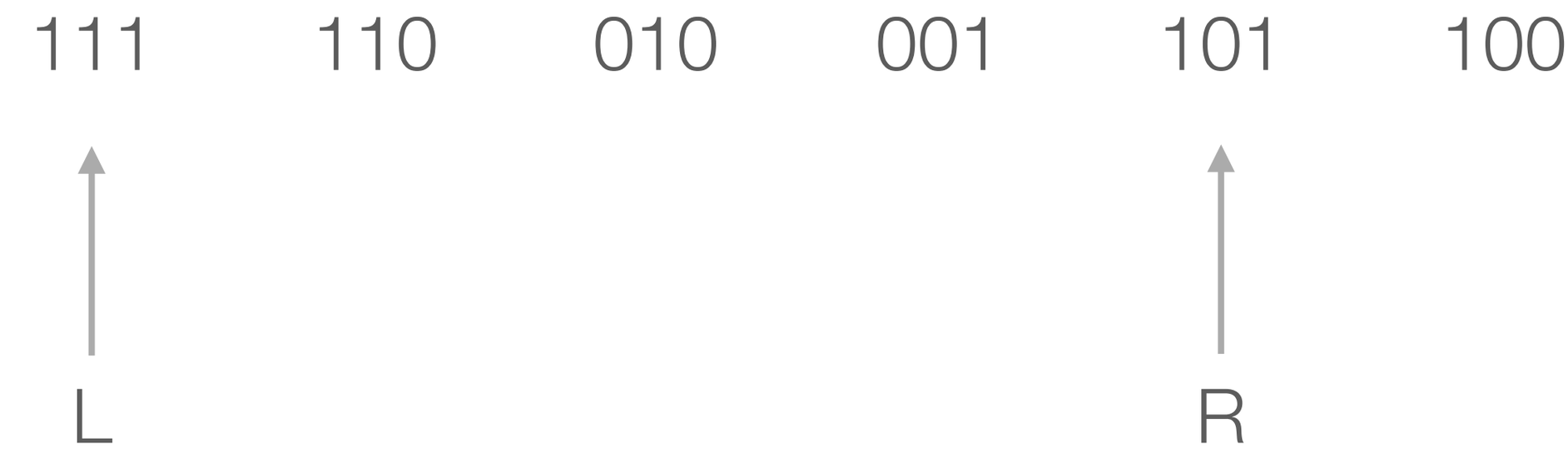
Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket



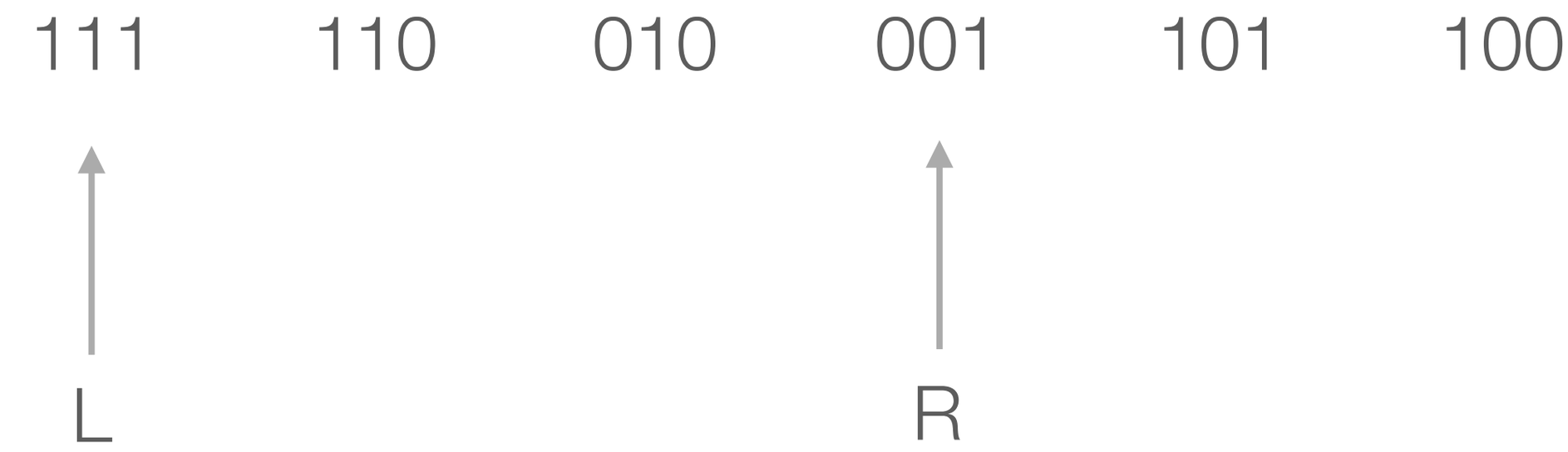
Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket



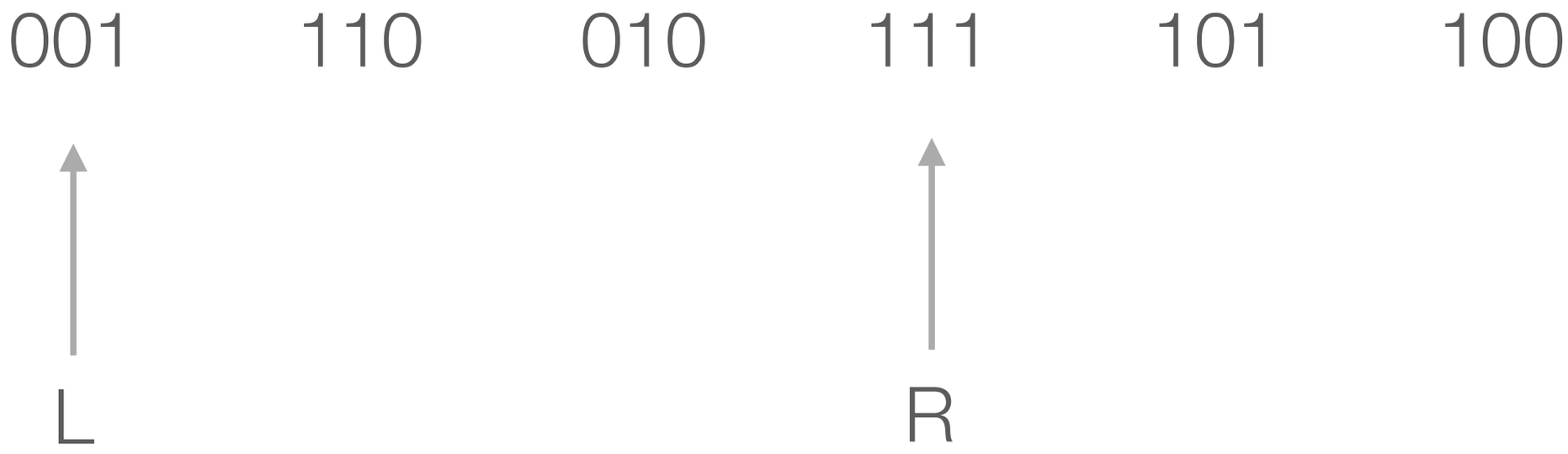
Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket



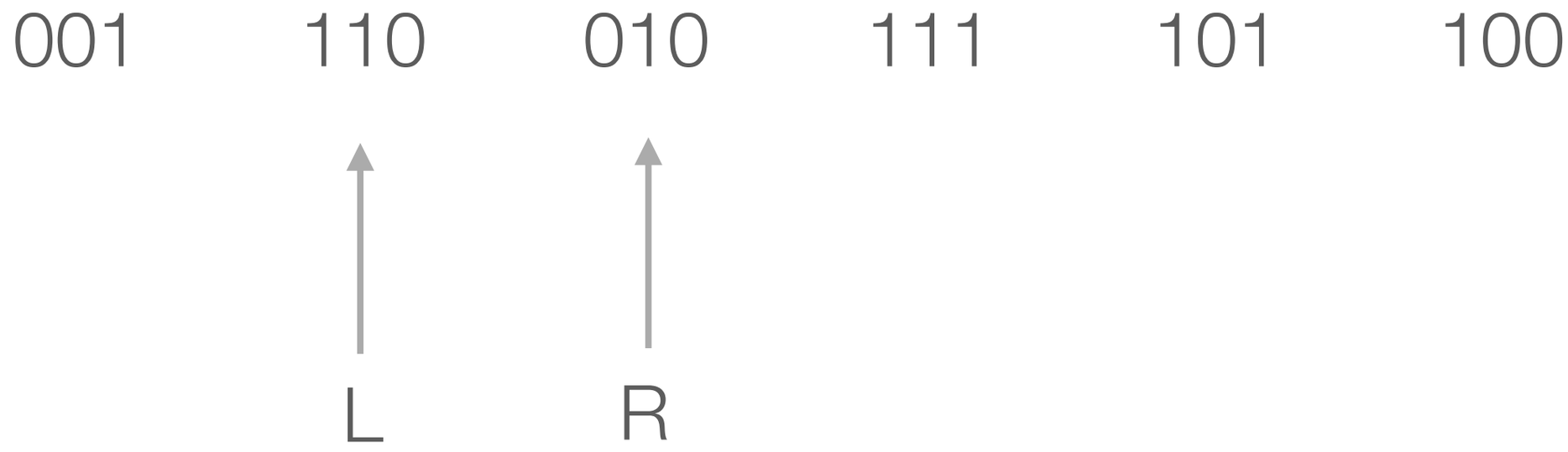
Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket



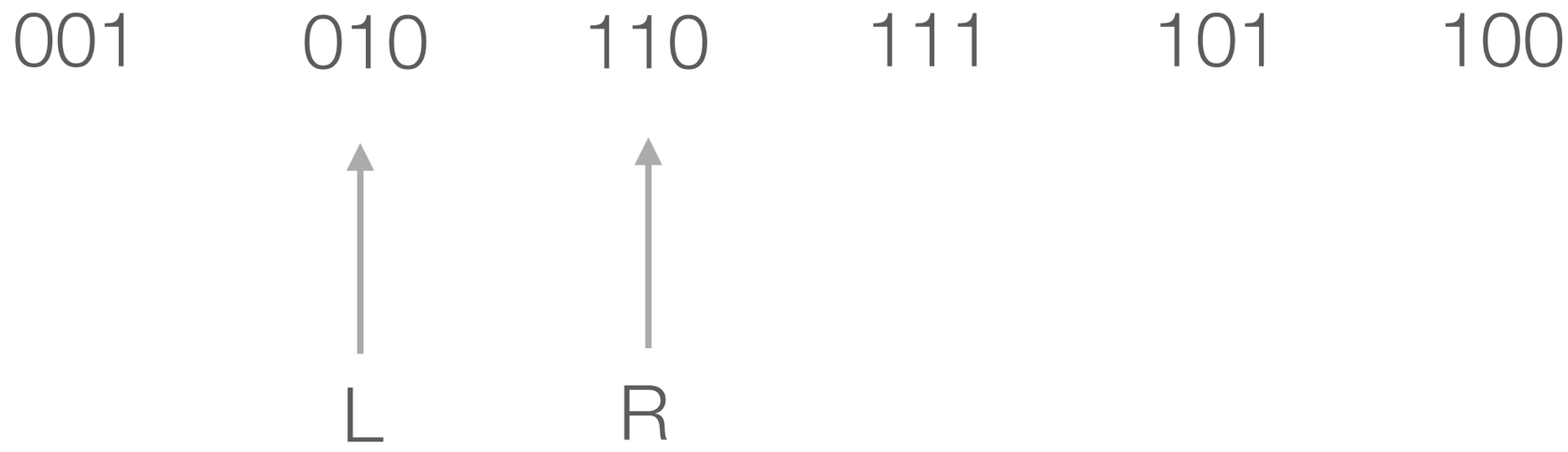
Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket



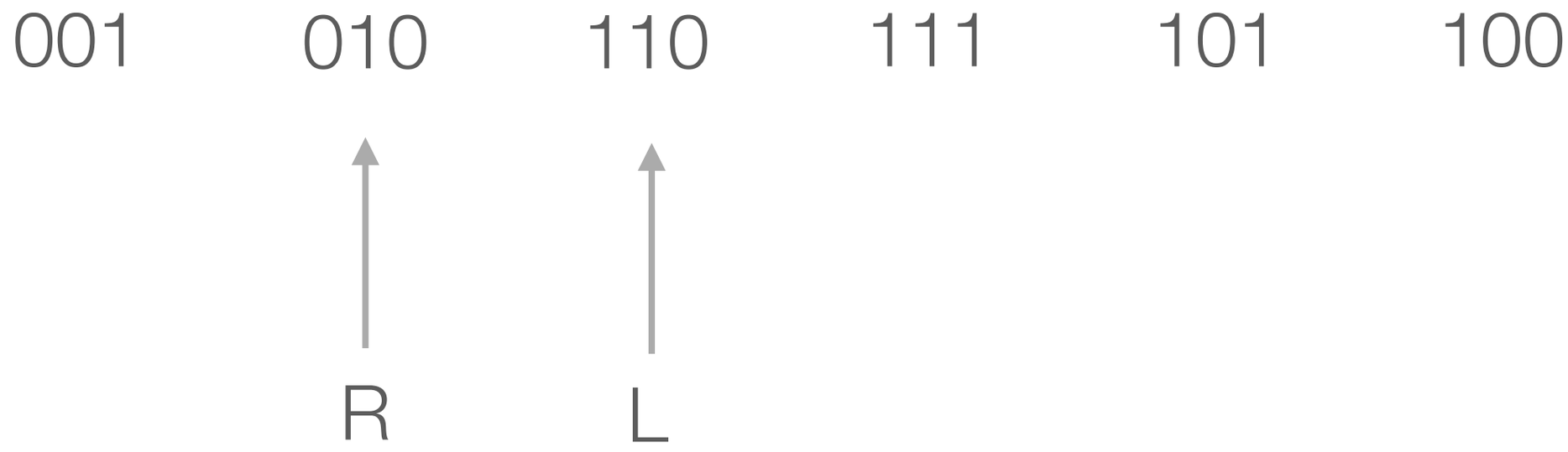
Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket



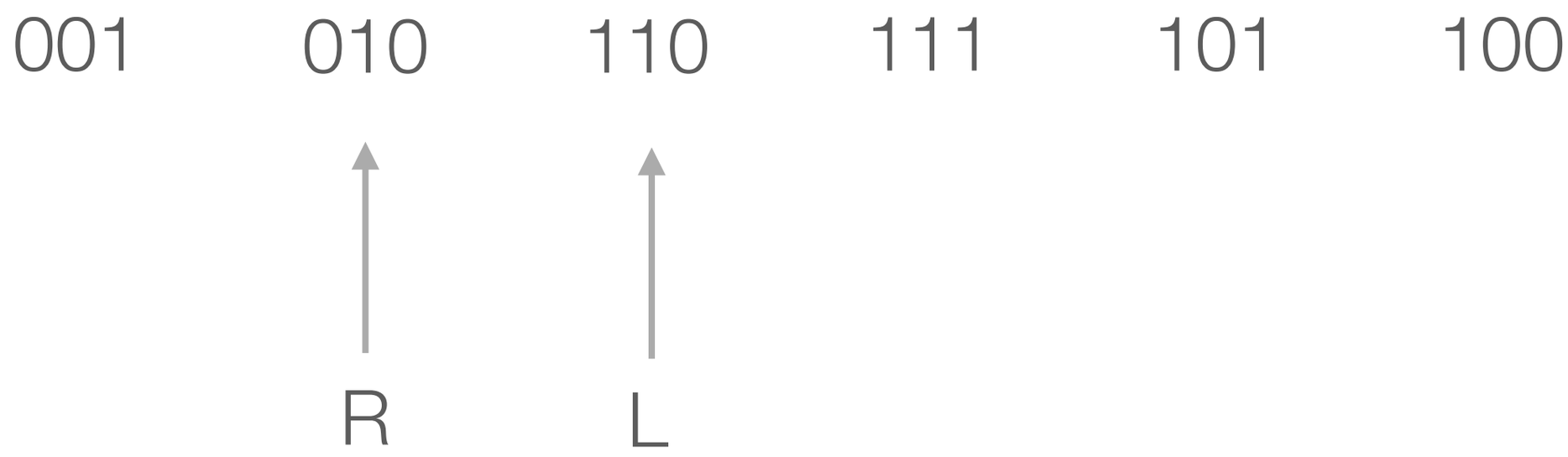
Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket



Radix Sort (cont'd)

- In-place binary MSD radix sort
  - Divide into 0s bucket and 1s bucket

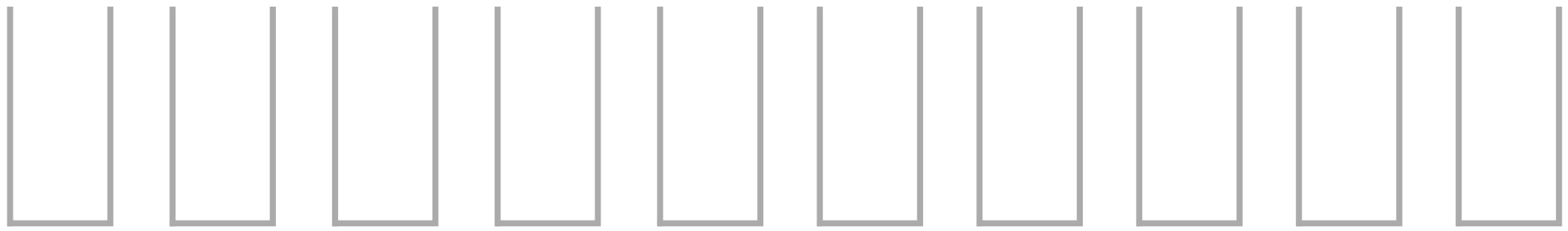


Recursively sort the 0s bucket and the 1s bucket based on the second bit and so on

Radix Sort (cont'd)

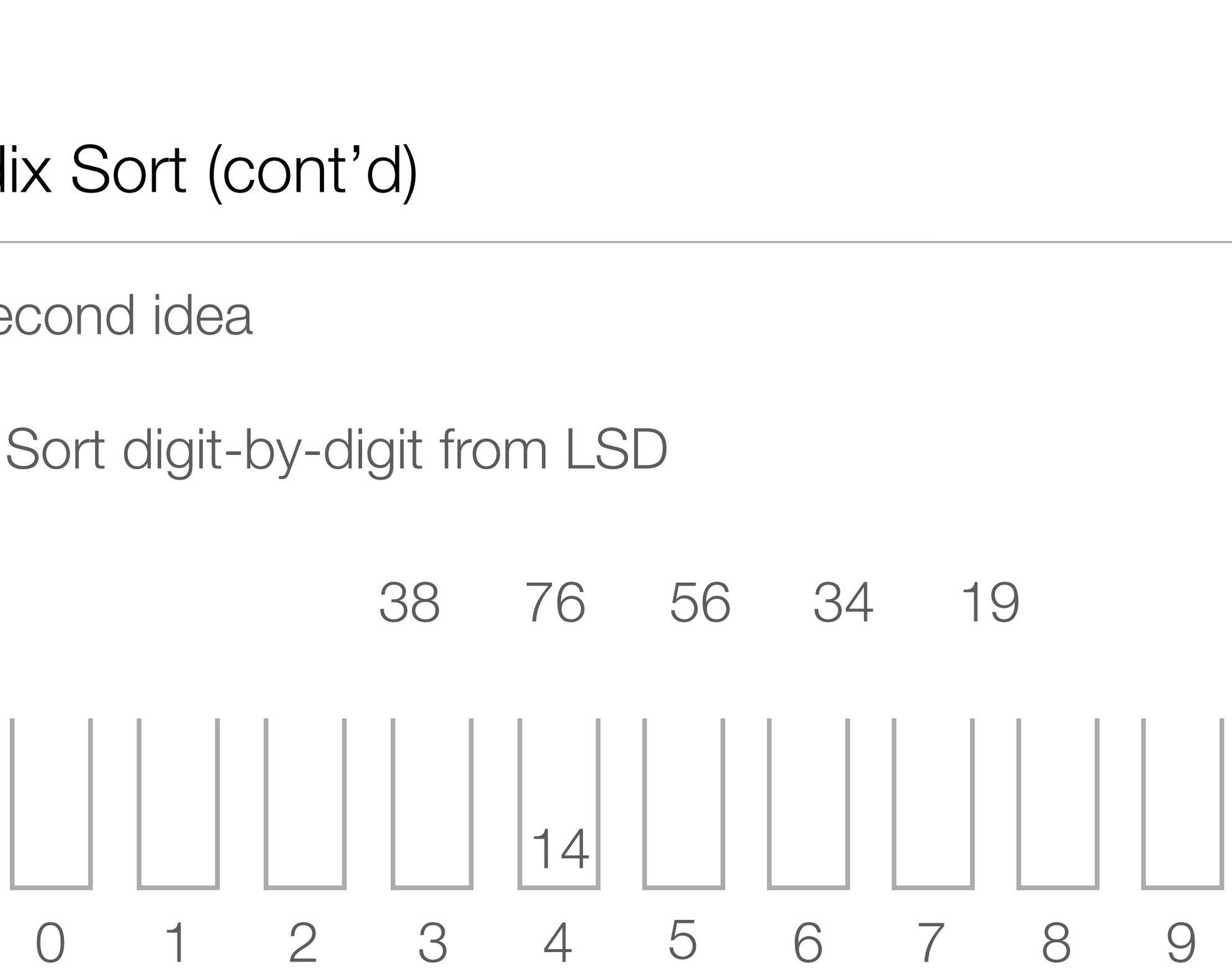
- Second idea
  - Sort digit-by-digit from LSD

14 38 76 56 34 19



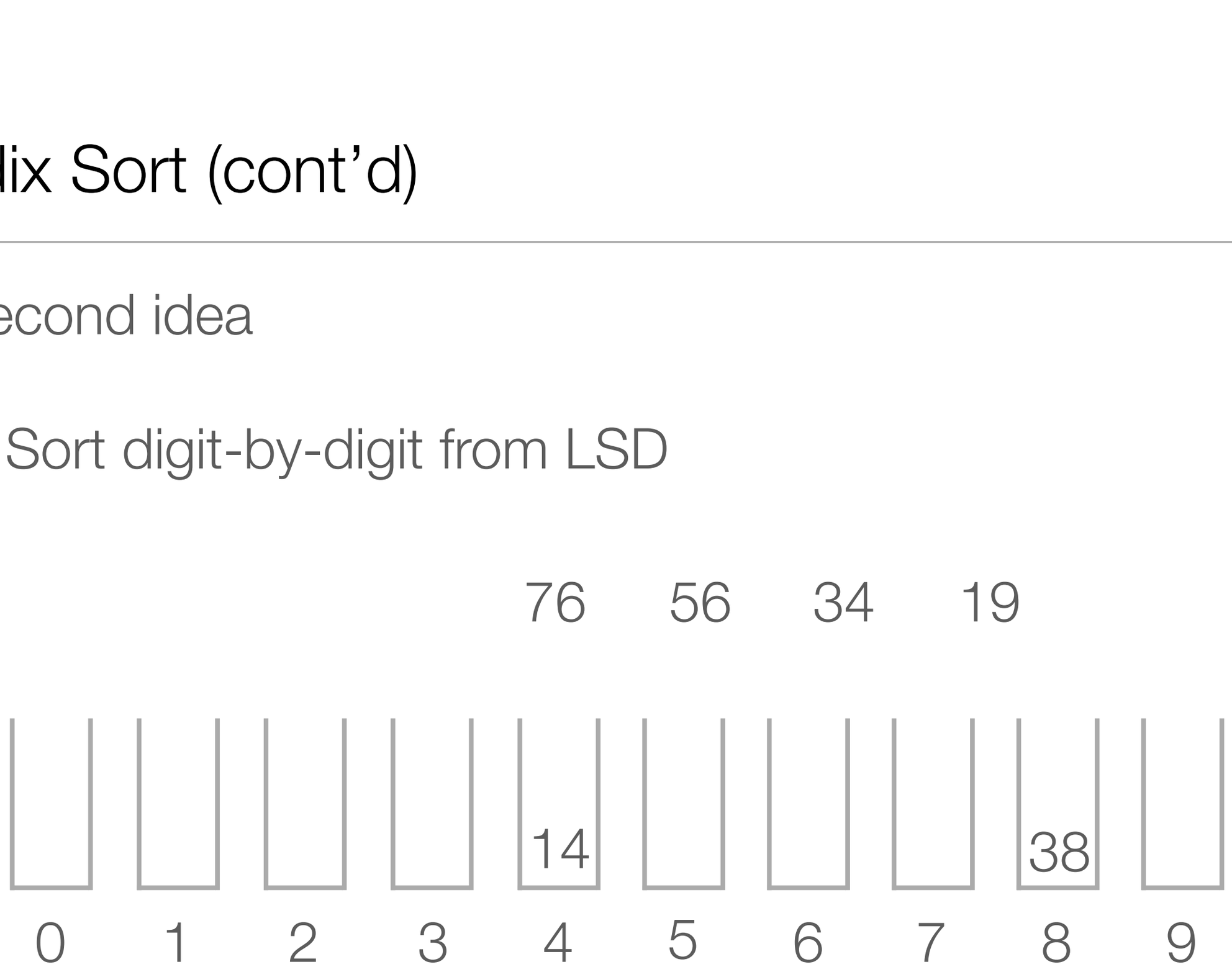
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



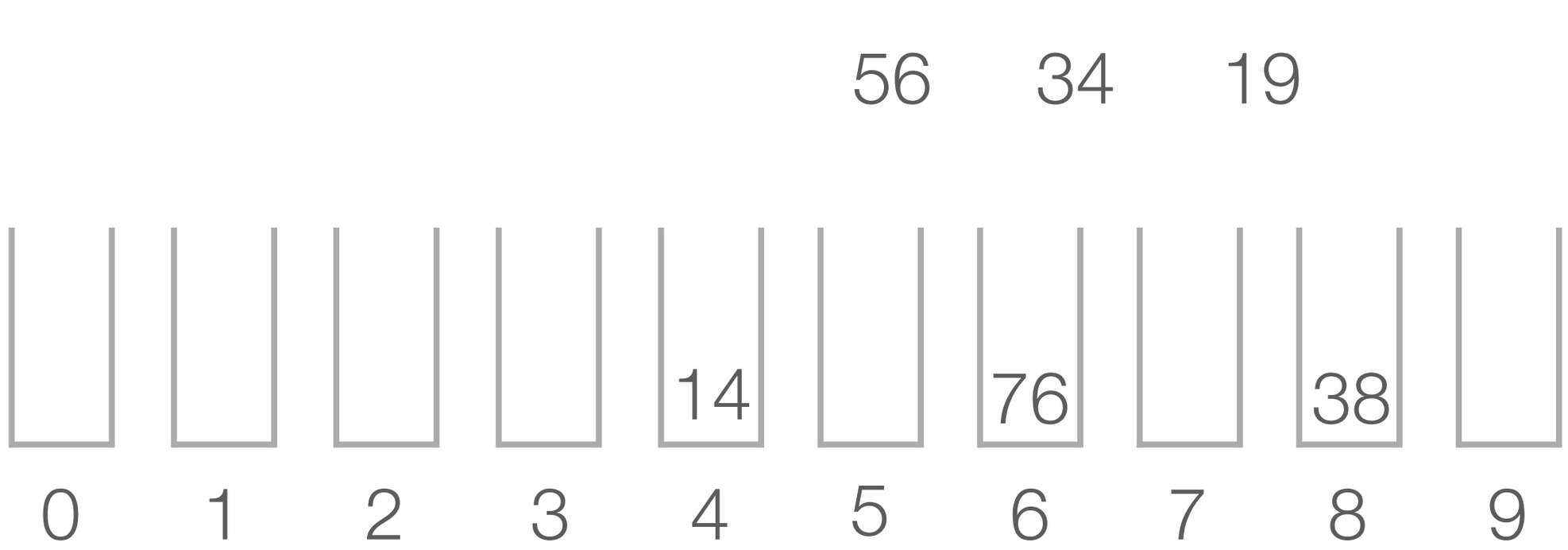
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



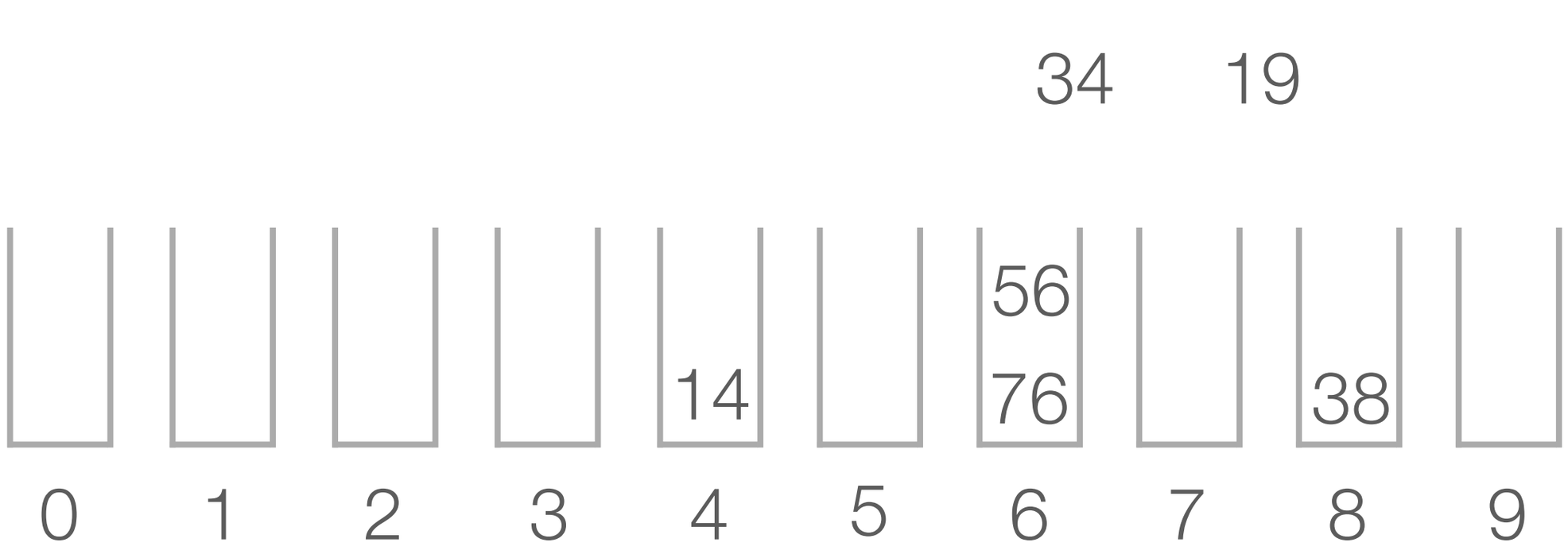
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



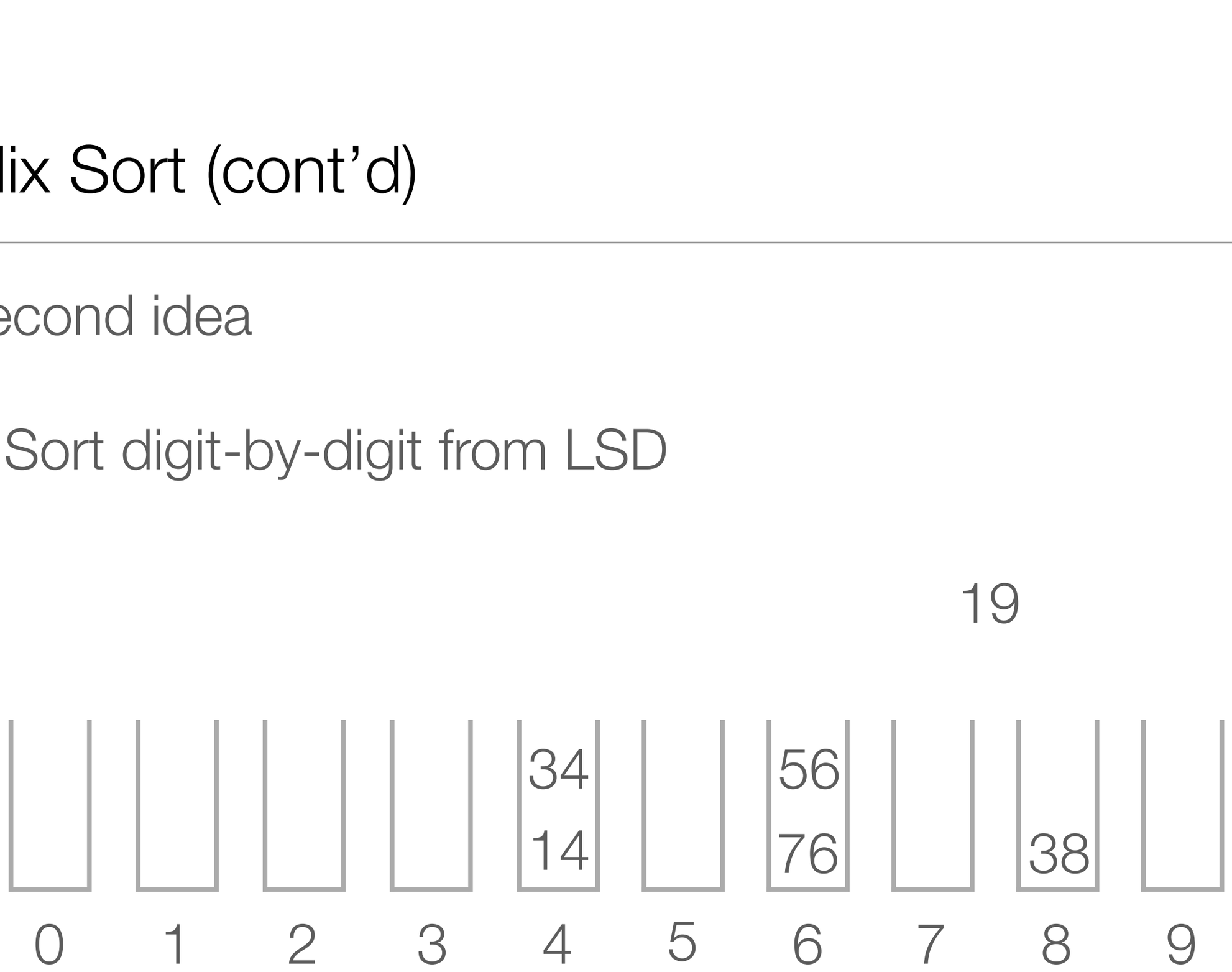
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



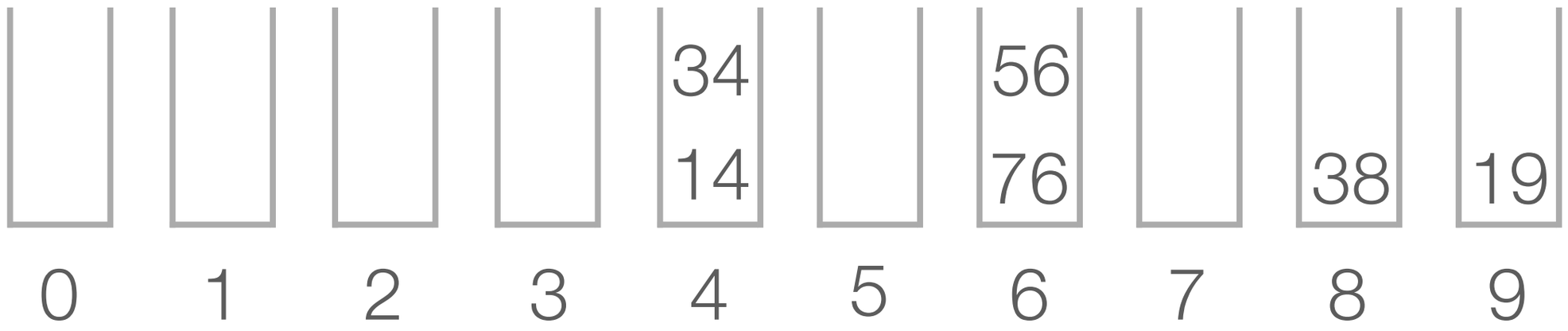
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



Radix Sort (cont'd)

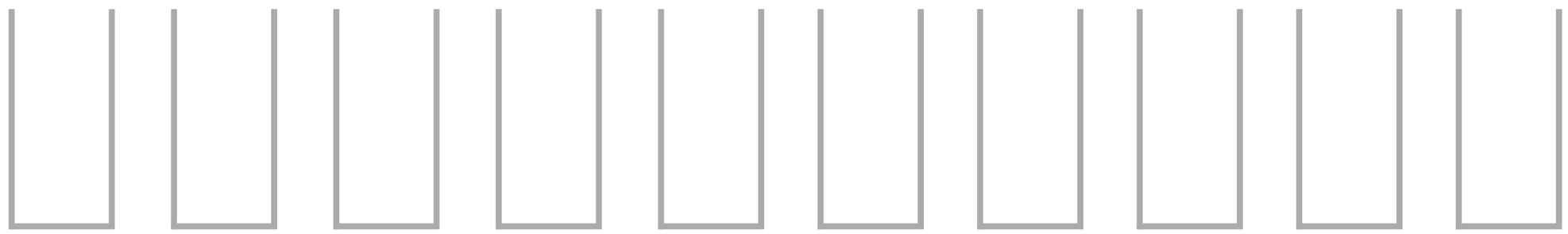
- Second idea
  - Sort digit-by-digit from LSD



Radix Sort (cont'd)

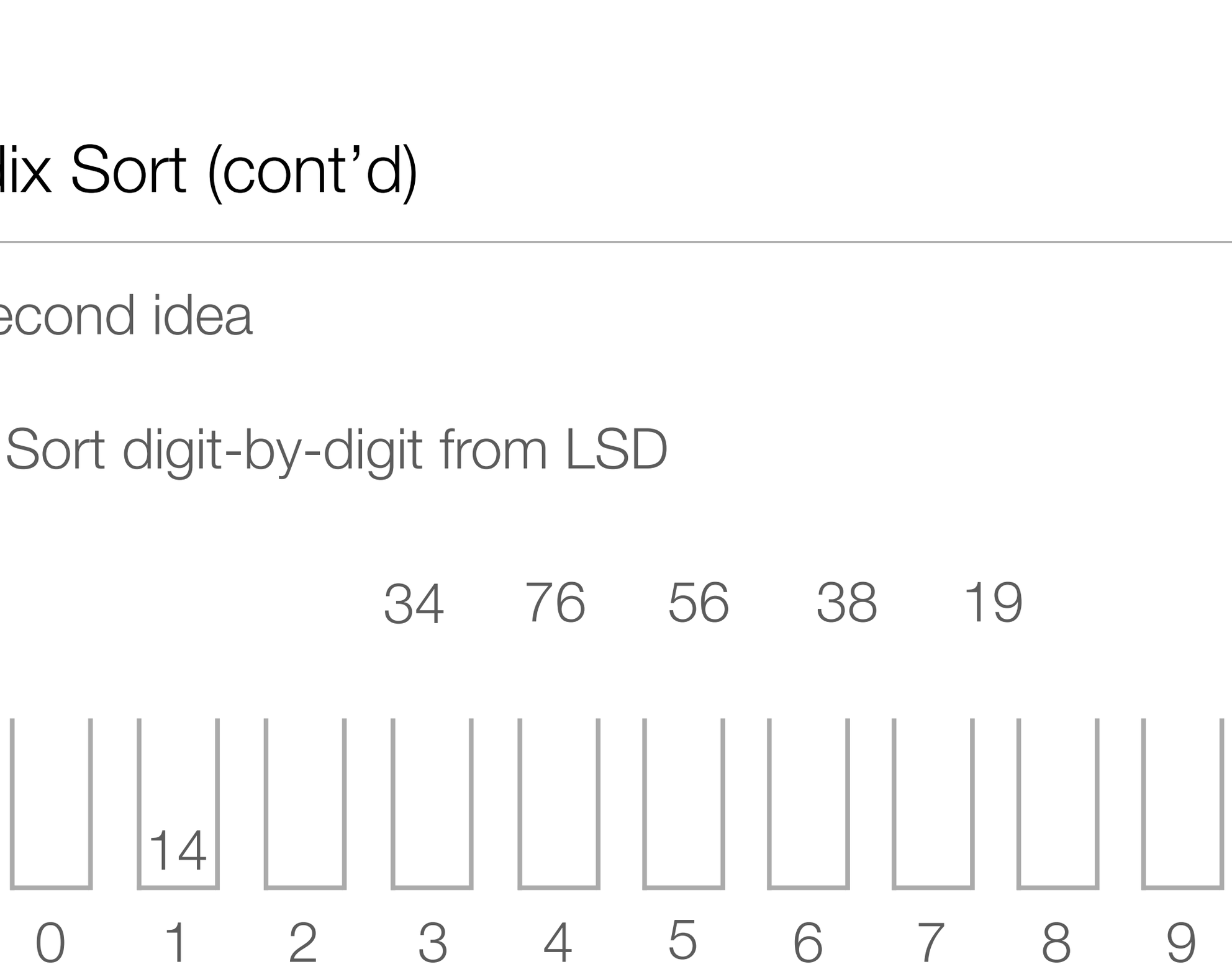
- Second idea
  - Sort digit-by-digit from LSD

14 34 76 56 38 19



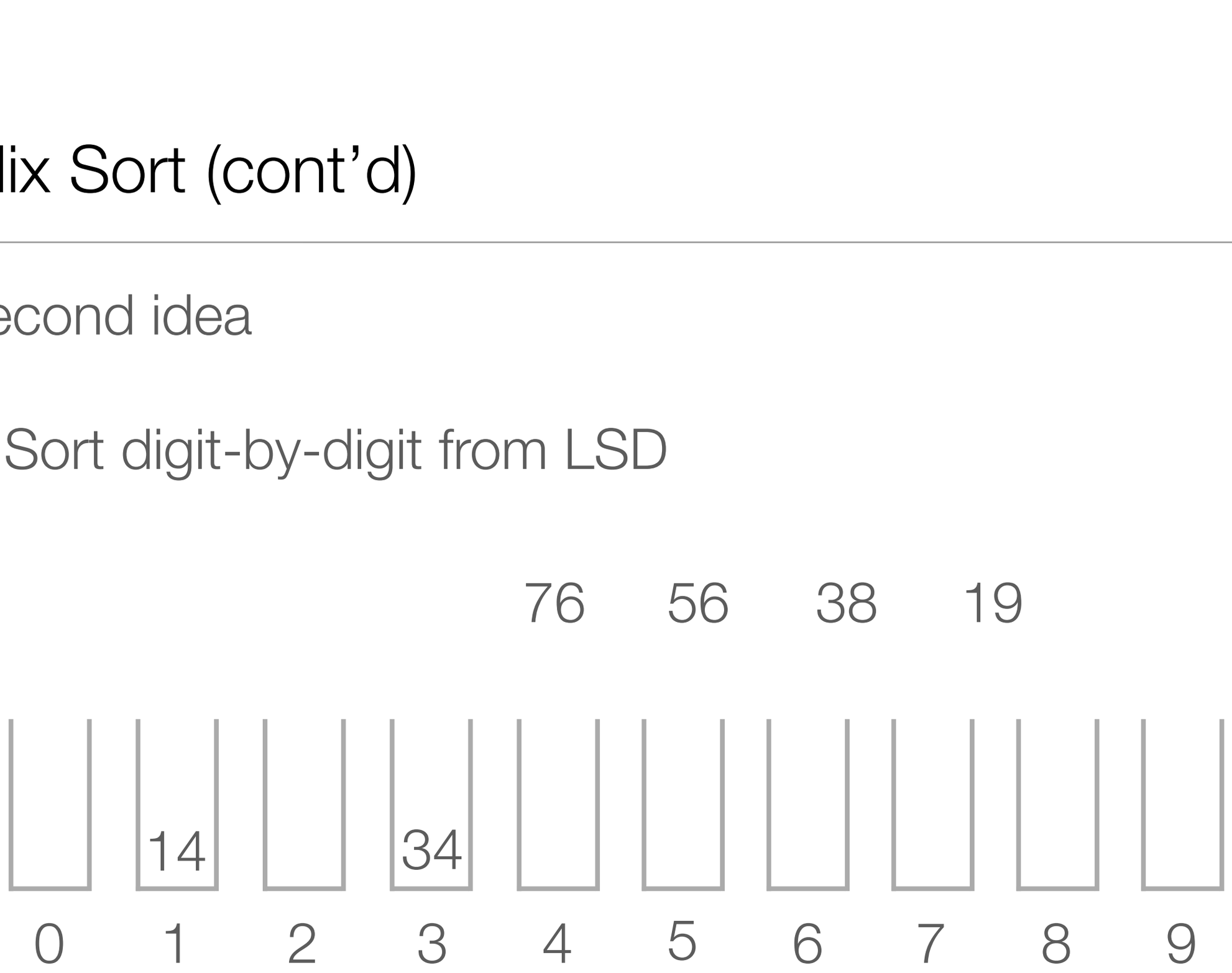
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



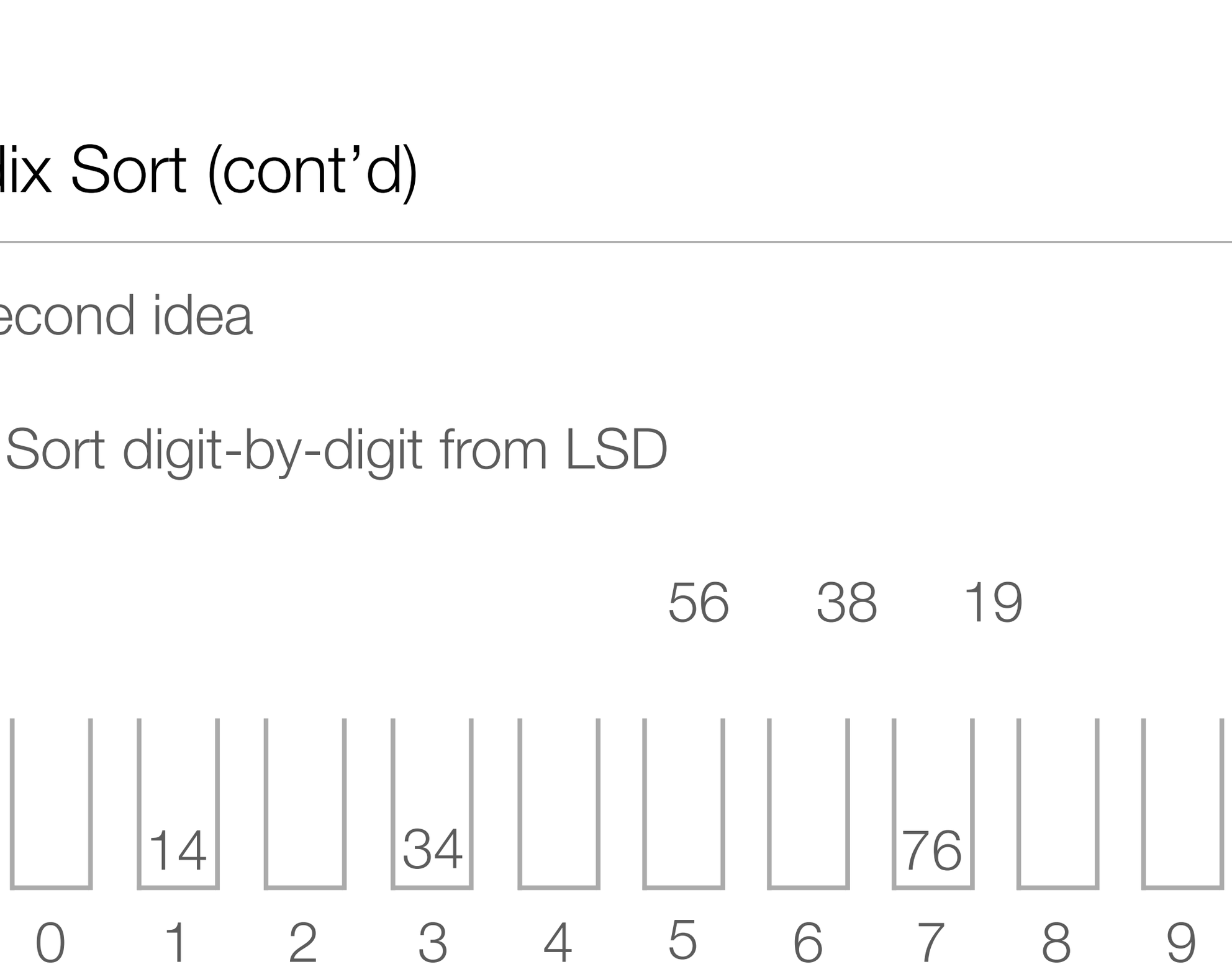
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



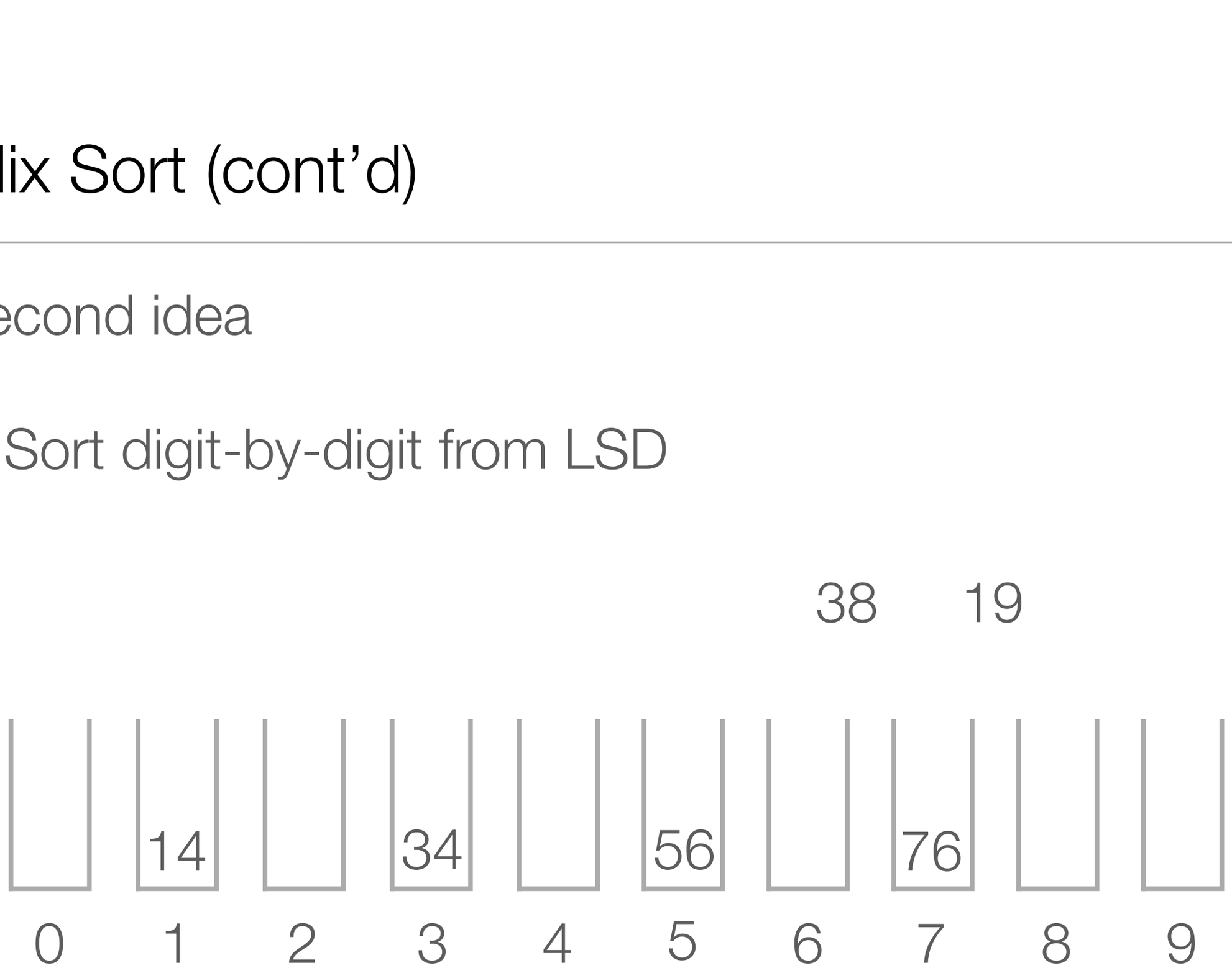
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



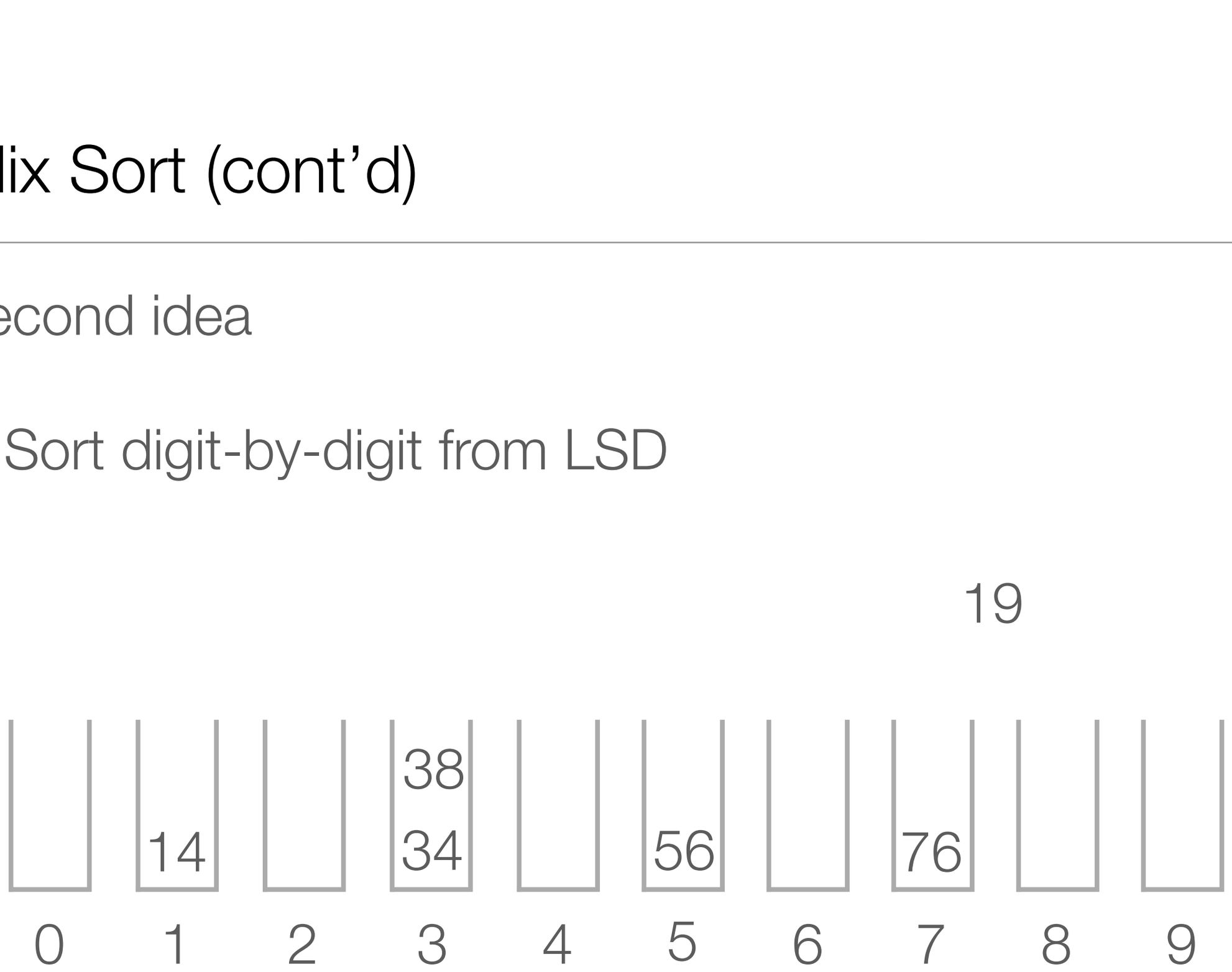
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



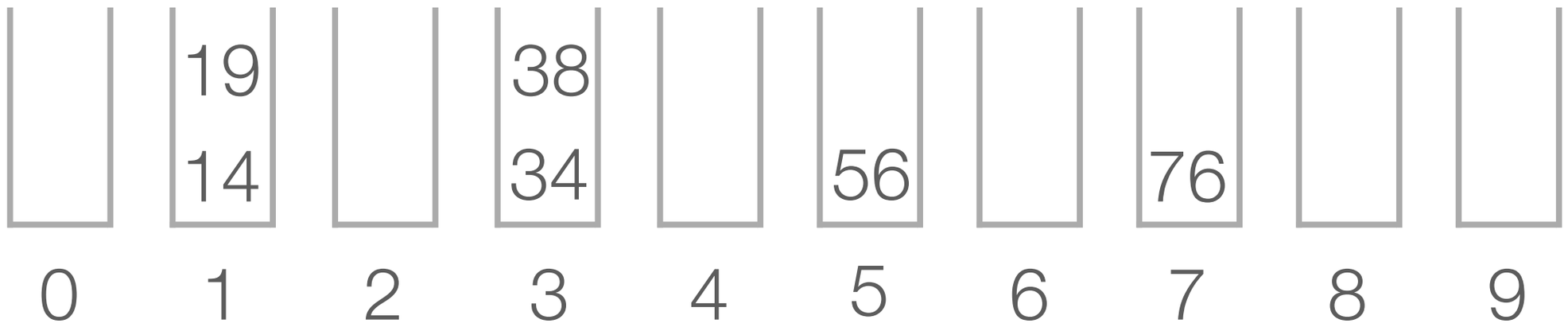
Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD



Radix Sort (cont'd)

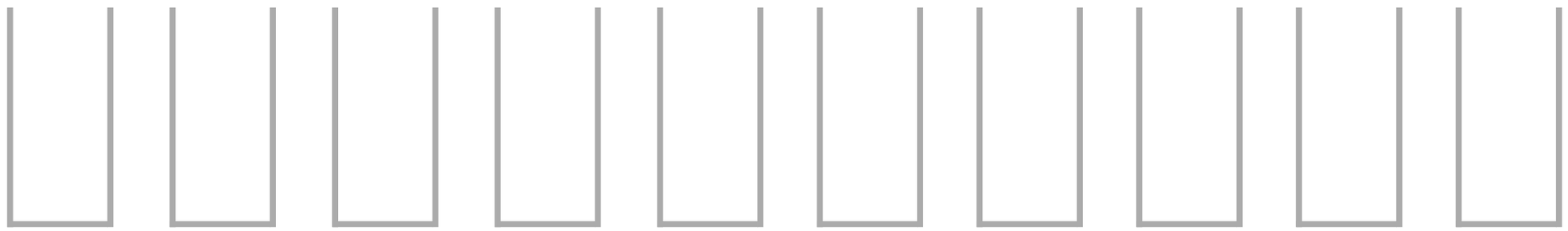
- Second idea
  - Sort digit-by-digit from LSD



Radix Sort (cont'd)

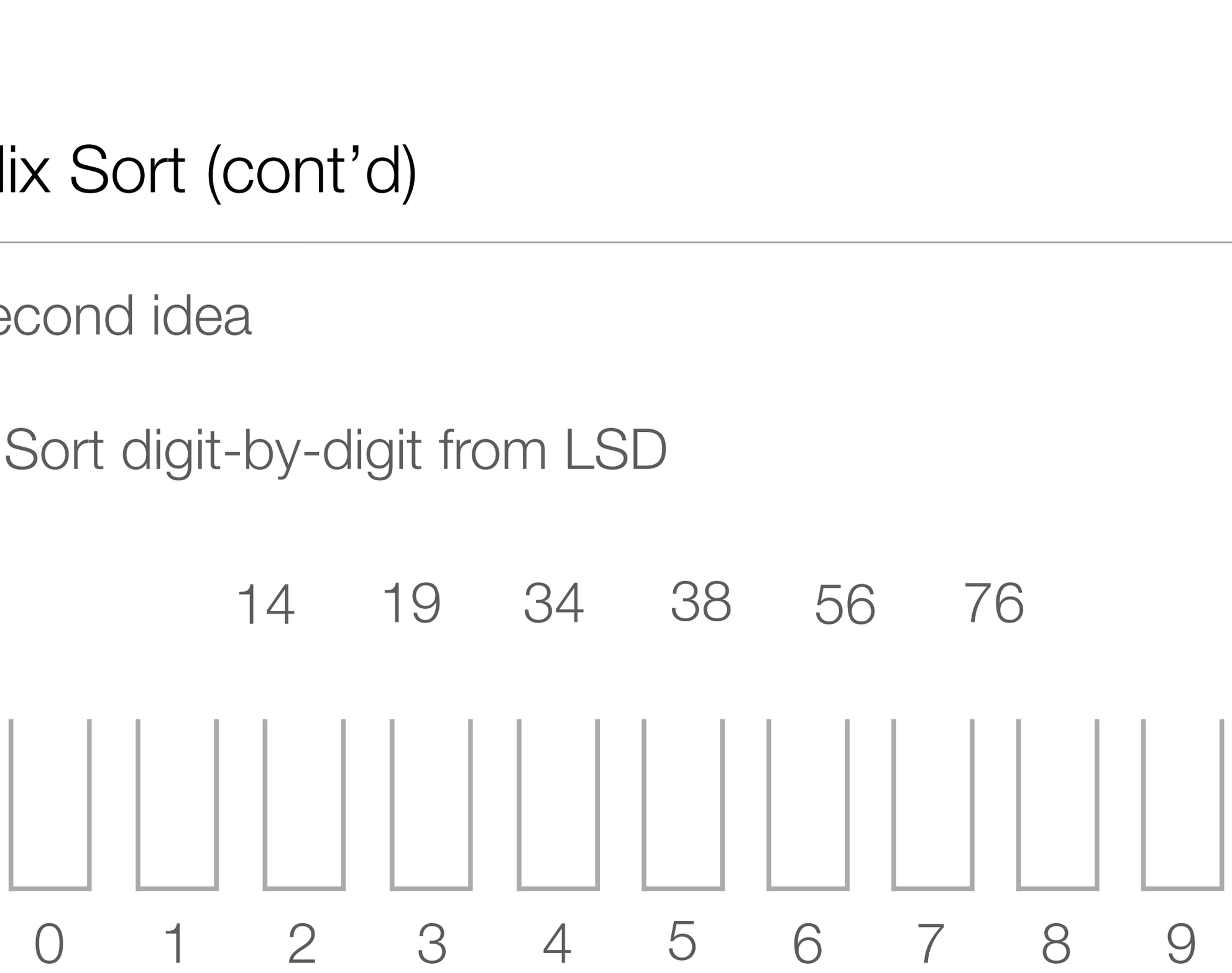
- Second idea
  - Sort digit-by-digit from LSD

14 19 34 38 56 76



Radix Sort (cont'd)

- Second idea
  - Sort digit-by-digit from LSD

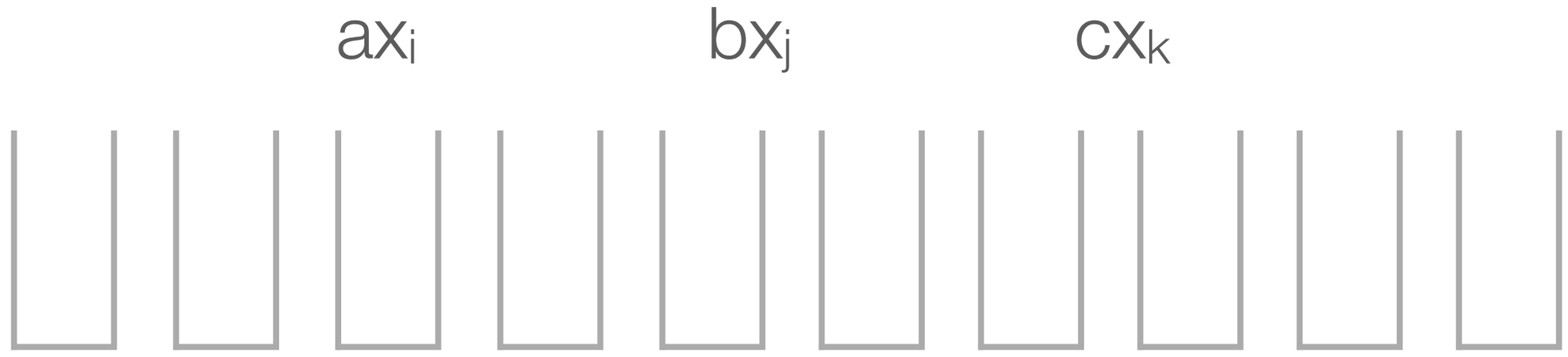


This algorithm is also called ***straight-radix sort***

Radix Sort (cont'd)

- Why the LSD radix sort works?

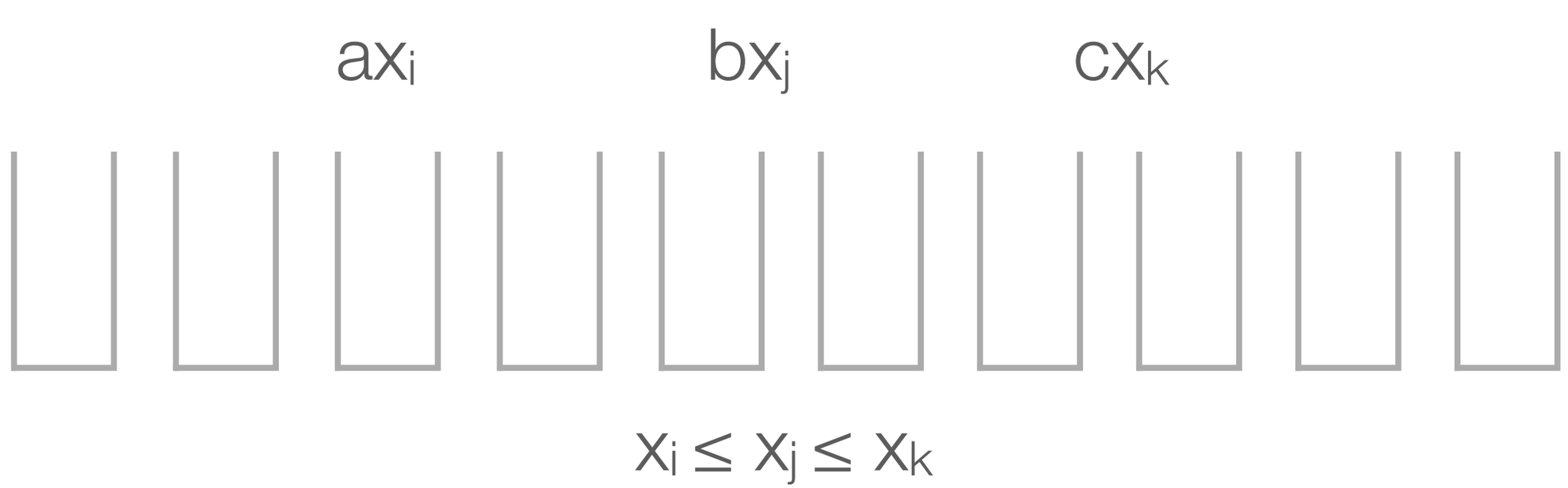
  Induction hypothesis: we know how to sort elements with < n digits



Radix Sort (cont'd)

- Why the LSD radix sort works?

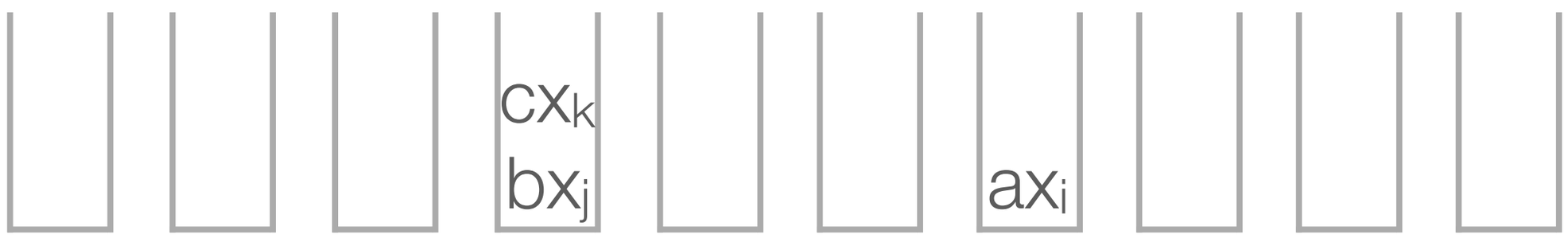
  Induction hypothesis: we know how to sort elements with < n digits



Radix Sort (cont'd)

- Why the LSD radix sort works?

  Induction hypothesis: we know how to sort elements with < n digits

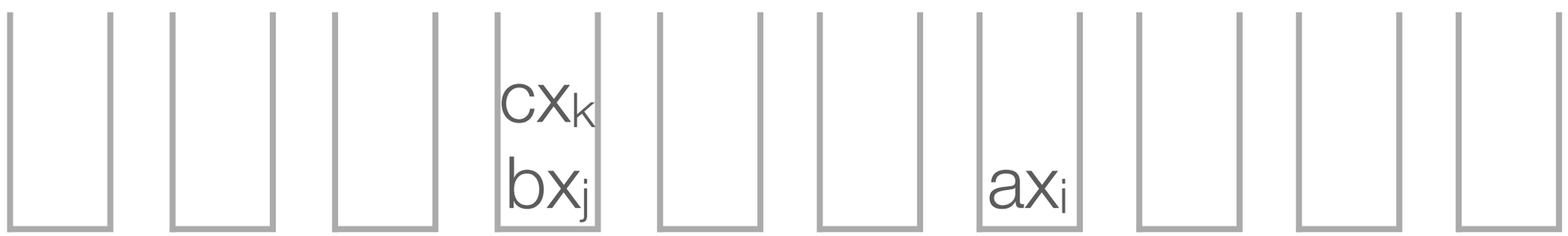


$$x_i \leq x_j \leq x_k$$

Radix Sort (cont'd)

- Why the LSD radix sort works?

  Induction hypothesis: we know how to sort elements with < n digits



$x_i \leq x_j \leq x_k$

Final result: $bx_j\ cx_k\ ax_i$

Radix Sort

```
Algorithm Straight_Radix (X, n, k);
begin
    put all elements of X in a queue GQ;
    for i := 1 to d do
        initialize queue Q[i] to be empty
    for i := k downto 1 do
        while GQ is not empty do
            pop x from GQ;
            di := the i-th digit of x;
            insert x into Q[di];
        for t := 1 to d do
            insert Q[t] into GQ;
    for i := 1 to n do
        pop X[i] from GQ
end
```

{ d is the number of possible digits }

Time complexity: O(nk)

Insertion Sort

- Sort n-1 numbers
- Find the correct place of the n-th number

14 38 76 56 34 19

Insertion Sort

- Sort n-1 numbers
- Find the correct place of the n-th number

14 38 76 56 34 **19**

Insertion Sort

- Sort n-1 numbers
- Find the correct place of the n-th number

14 34 38 56 76 **19**

Insertion Sort

- Sort n-1 numbers
- Find the correct place of the n-th number

14 **19** 34 38 56 76

Selection Sort

- Put the maximum number at the end
- Sort n-1 numbers

14 38 76 56 34 19

Selection Sort

- Put the maximum number at the end
- Sort n-1 numbers

14 38 **76** 56 34 19

Selection Sort

- Put the maximum number at the end
- Sort n-1 numbers

14 38 19 56 34 **76**

Selection Sort

- Put the maximum number at the end
- Sort n-1 numbers

14 19 34 38 56 **76**

Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

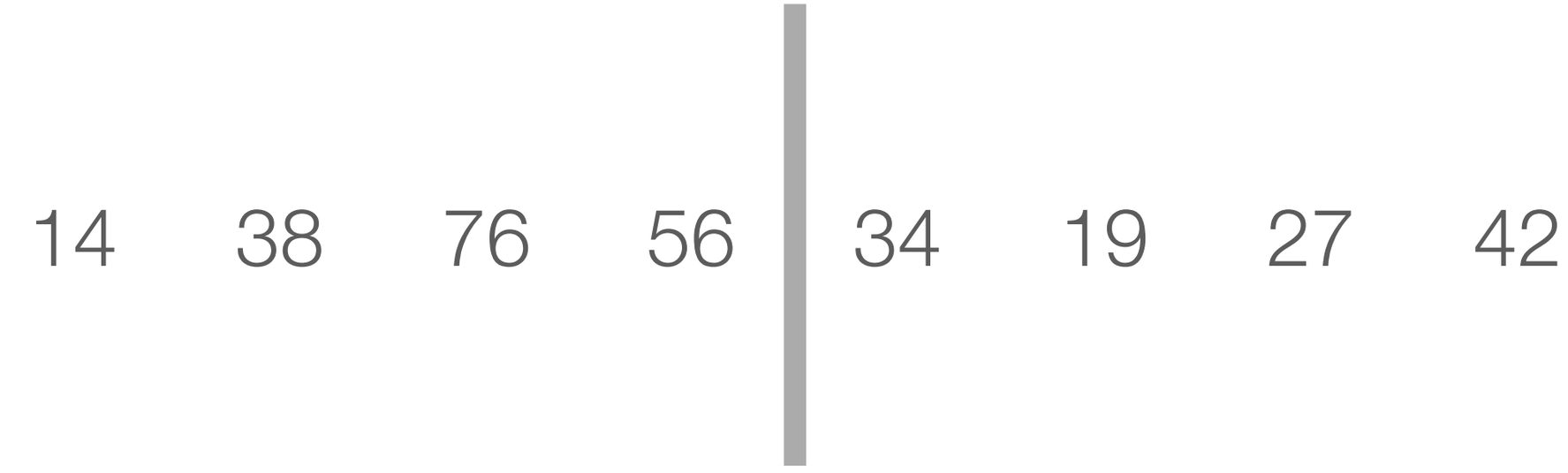
14 38 76 56 34 19 27 42

Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

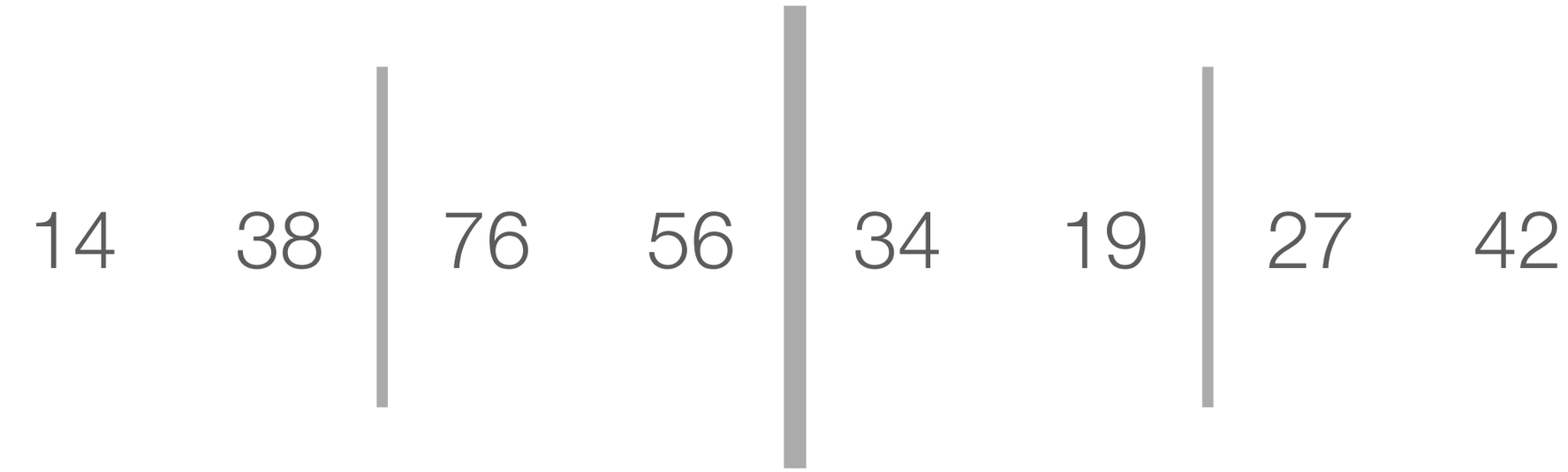


Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

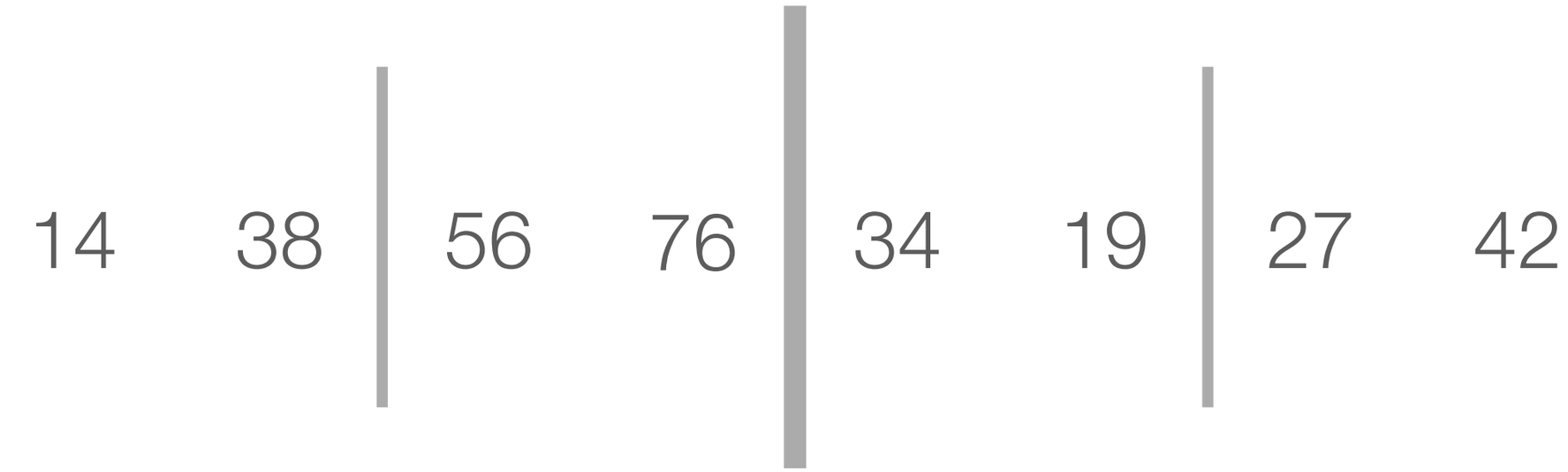


Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

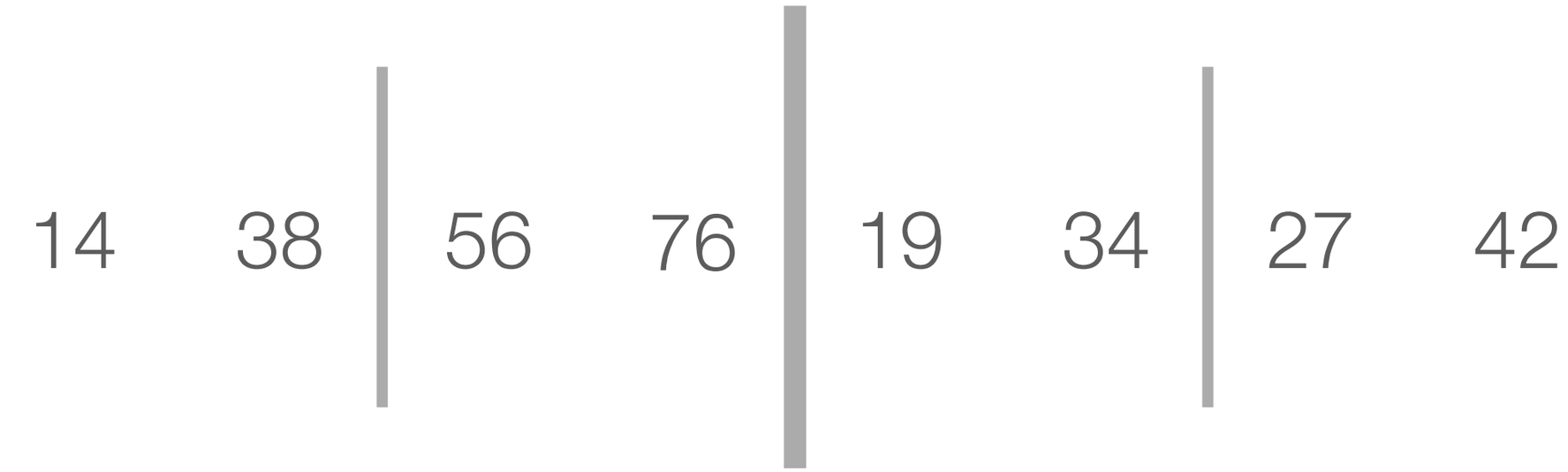


Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

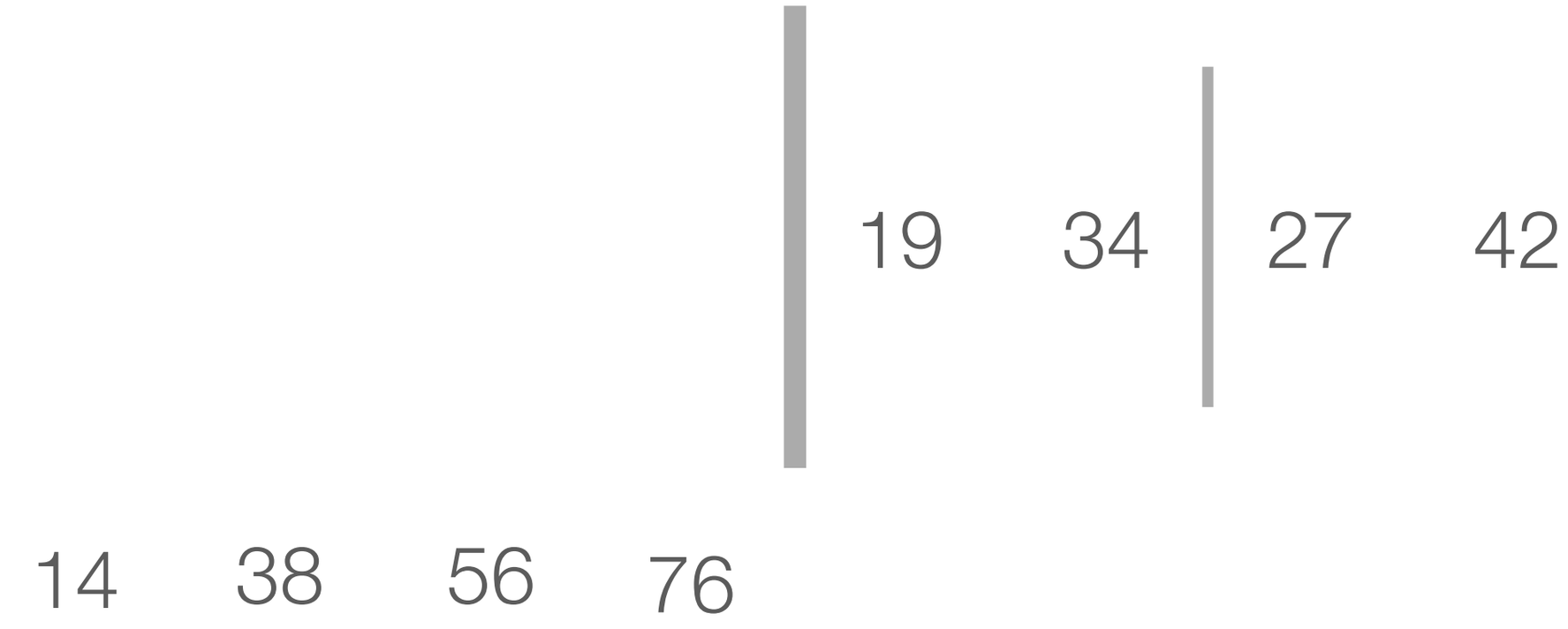


Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

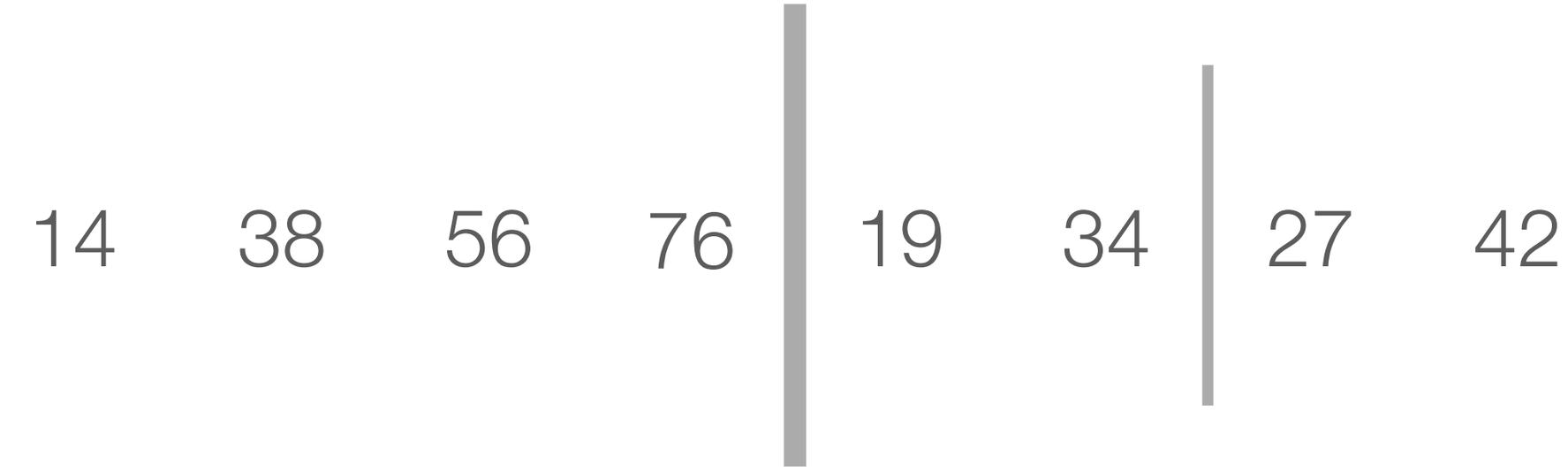


Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

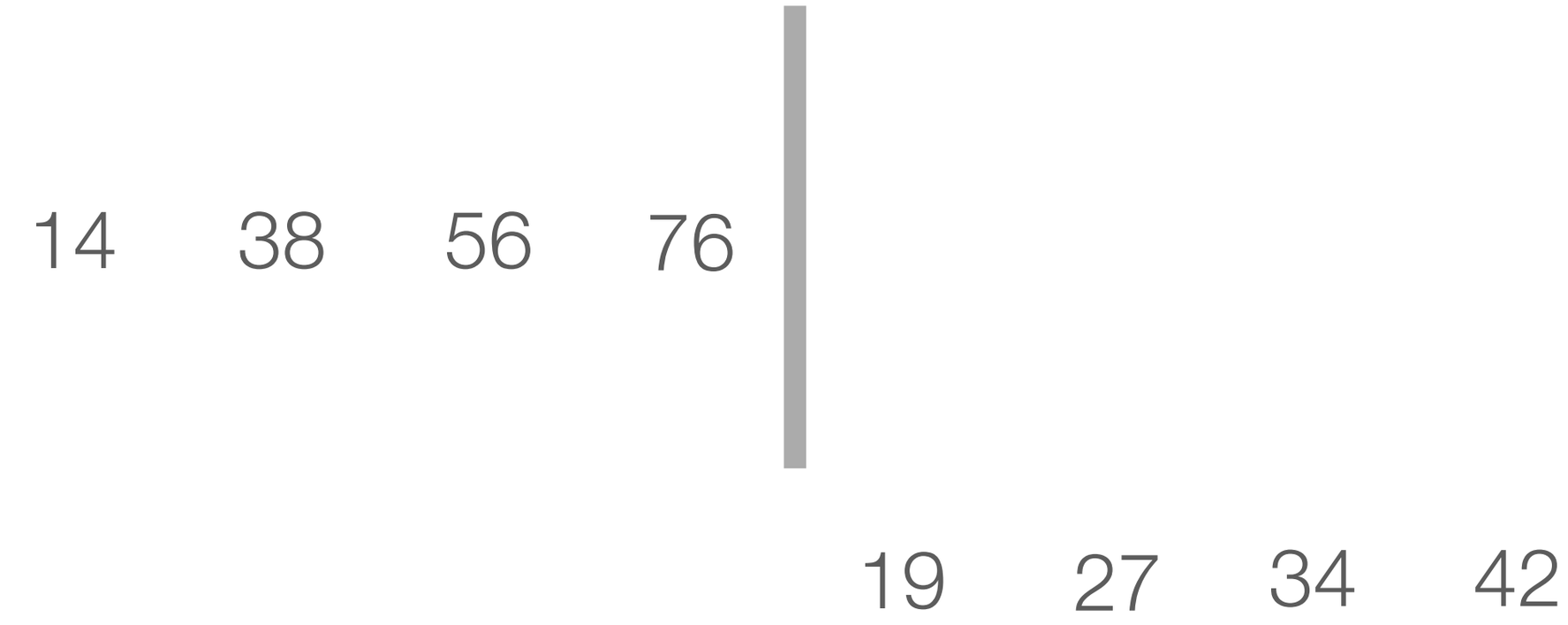


Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

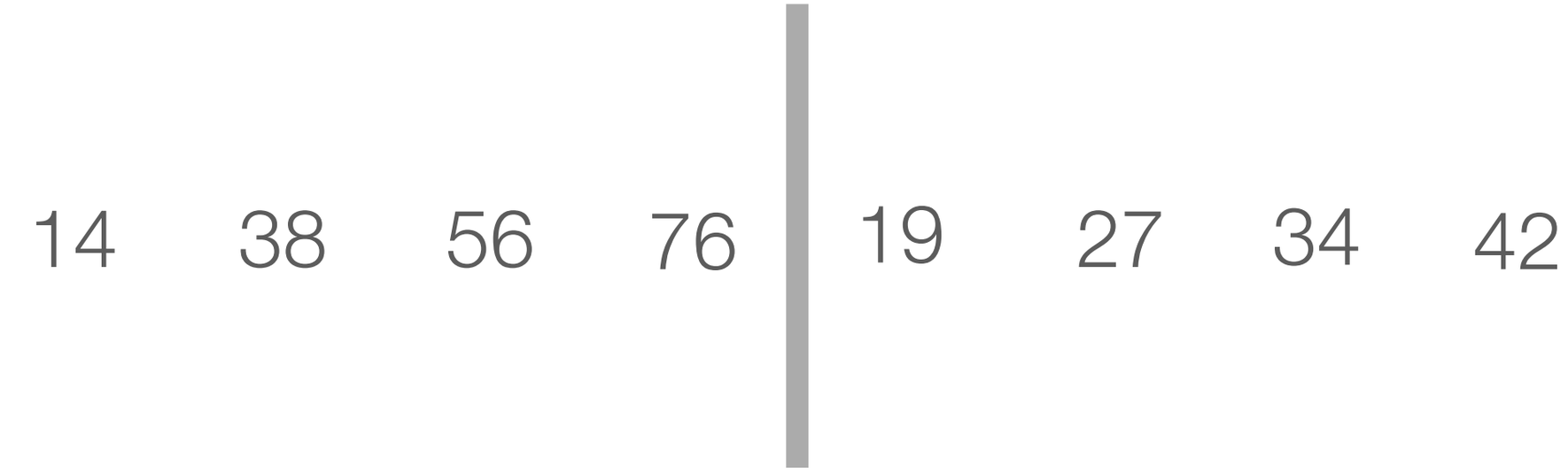


Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$



Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

14 19 27 34 38 42 56 76

Merge Sort

- Split into two arrays
- Sort the two arrays
- Merge the two sorted arrays

$T(2n) = 2T(n) + O(n), T(2) = 1$

Time complexity: $O(n \log n)$

14 19 27 34 38 42 56 76

Merge Sort (cont'd)

```
Algorithm Mergesort (X, n);
begin
    M_Sort(1, n)
end
```

```
procedure M_Sort (Left, Right);
begin
    if Right - Left = 1 then
        if X[Left] > X[Right] then swap(X[Left], X[Right])
    else if Left ≠ Right then
        Middle := ⌈½ (Left + Right)⌉ ;

        M_Sort(Left, Middle - 1);
        M_Sort(Middle, Right);
        …
end
```

Merge Sort (cont'd)

```
procedure M_Sort (Left, Right);
begin
        …
        i := Left; j := Middle; k := 0;
        while (i ≤ Middle - 1) and (j ≤ Right) do
            k := k + 1
            if X[i] ≤ X[j] then TEMP[k] := X[i]; i := i + 1
            else TEMP[k] := X[j]; j := j + 1;
        if j > Right then
            for t := 0 to Middle - 1 - i do
                X[Right - t] := X[Middle - 1 - t]
        for t := 0 to k - 1 do
            X[Left + t] := TEMP[t + 1]
end
```

Merge Sort (cont'd)

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
(2)	(6)	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	(5)	(8)	10	9	12	1	15	7	3	13	4	11	16	14
(2)	(5)	(6)	(8)	10	9	12	1	15	7	3	13	4	11	16	14
2	5	6	8	(9)	(10)	12	1	15	7	3	13	4	11	16	14
2	5	6	8	9	10	(1)	(12)	15	7	3	13	4	11	16	14
2	5	6	8	(1)	(9)	(10)	(12)	15	7	3	13	4	11	16	14
(1)	(2)	(5)	(6)	(8)	(9)	(10)	(12)	15	7	3	13	4	11	16	14
1	2	5	6	8	9	10	12	(7)	(15)	3	13	4	11	16	14
1	2	5	6	8	9	10	12	7	15	(3)	(13)	4	11	16	14
1	2	5	6	8	9	10	12	(3)	(7)	(13)	(15)	4	11	16	14
1	2	5	6	8	9	10	12	3	7	13	15	(4)	(11)	16	14
1	2	5	6	8	9	10	12	3	7	13	15	4	11	(14)	(16)
1	2	5	6	8	9	10	12	3	7	13	15	(4)	(11)	(14)	(16)
1	2	5	6	8	9	10	12	(3)	(4)	(7)	(11)	(13)	(14)	(15)	(16)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)

Figure 6.8 An example of mergesort. The first row is in the initial order. Each row illustrates either an exchange operation or a merge. The numbers that are involved in the current operation are circled.

Source: [Manber 1989]

Quick Sort

- The problem with merge sort was the need for extra storage, since the merge is arbitrary and we cannot predict where each element will end up in the order

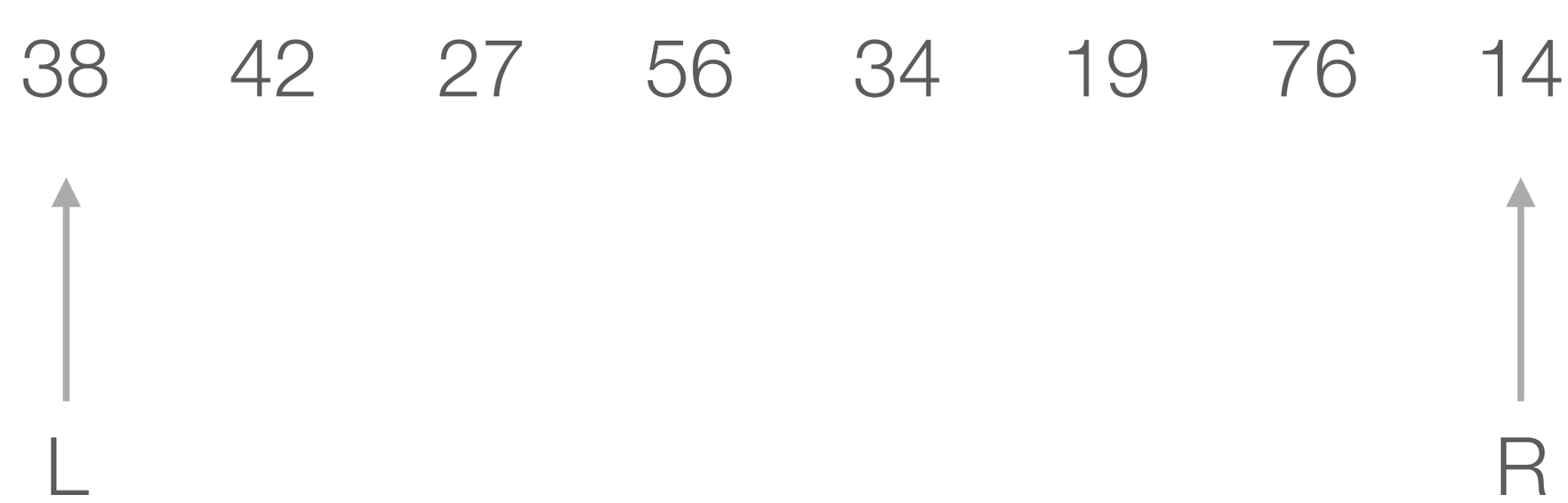
- Suppose we know a number x (called ***pivot***) such that
  - one-half of the elements ≥ x
  - one-half of the elements < x

- We can partition the numbers into two parts and then sort the two parts separately

- The combine step is trivial since the two parts already occupy the correct positions

Quick Sort (cont'd)

Partition



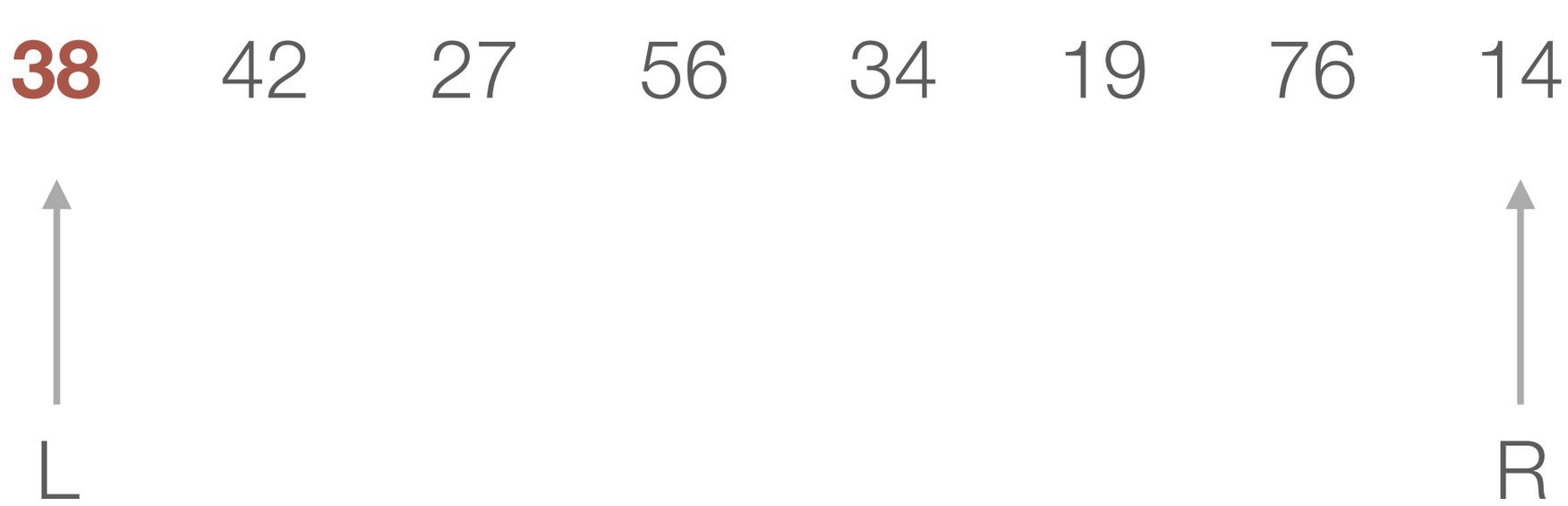
L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

Partition

Pivot: 38



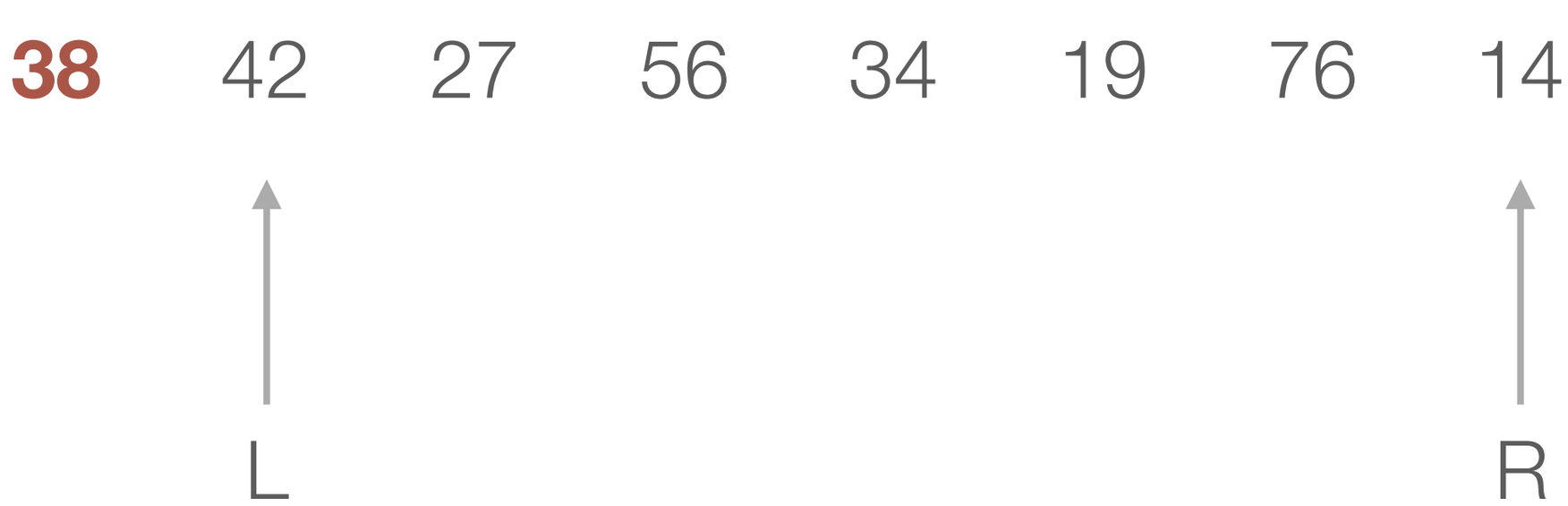
L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

Partition

Pivot: 38



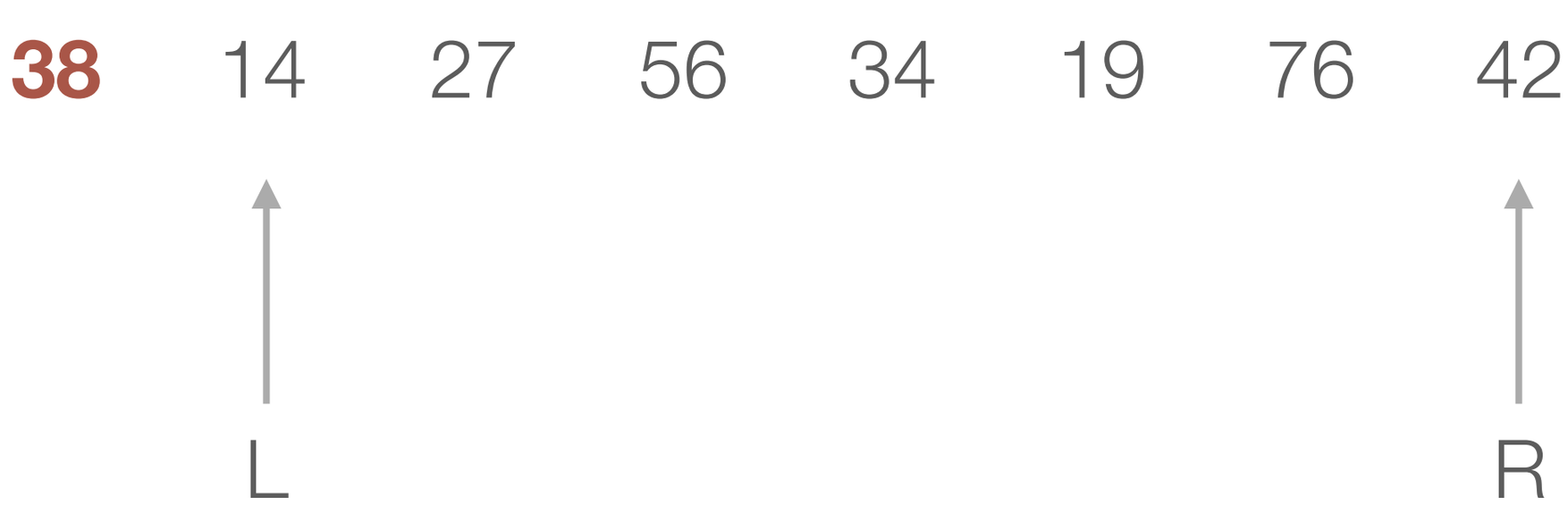
L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

Partition

Pivot: 38



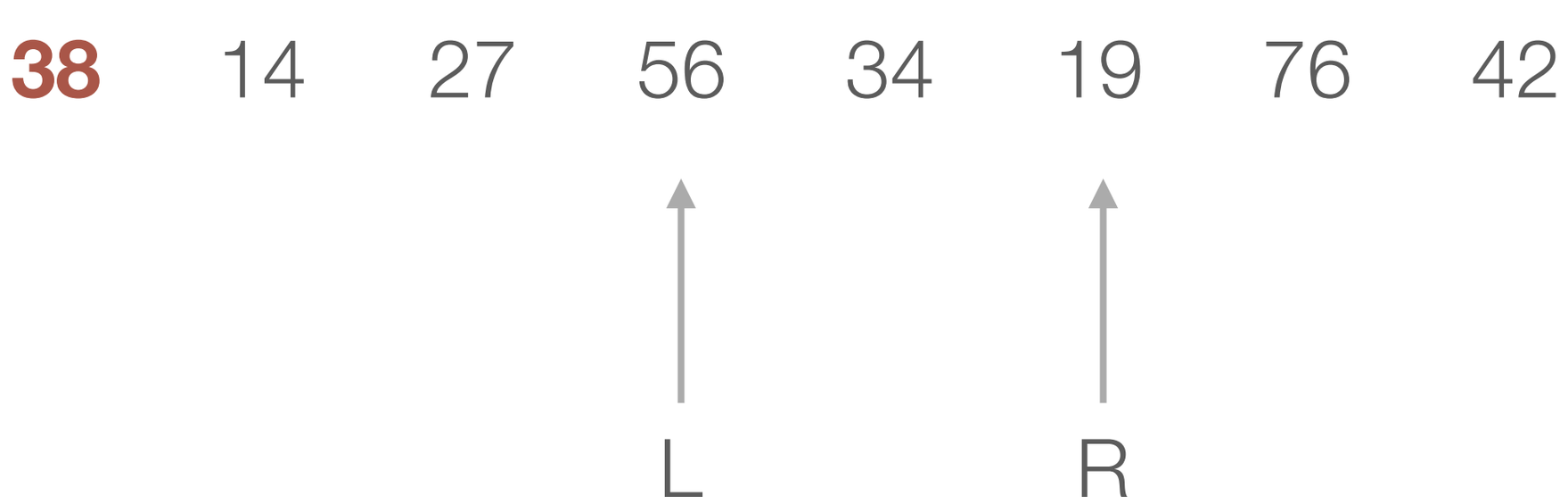
L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

Partition

Pivot: 38



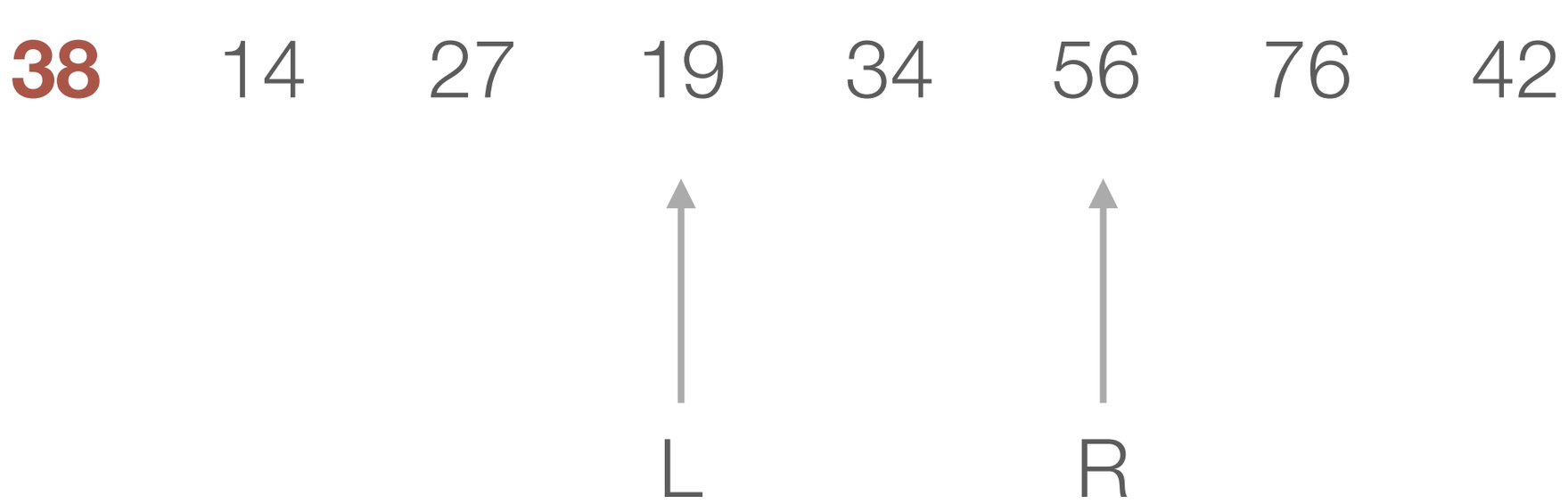
L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

Partition

Pivot: 38



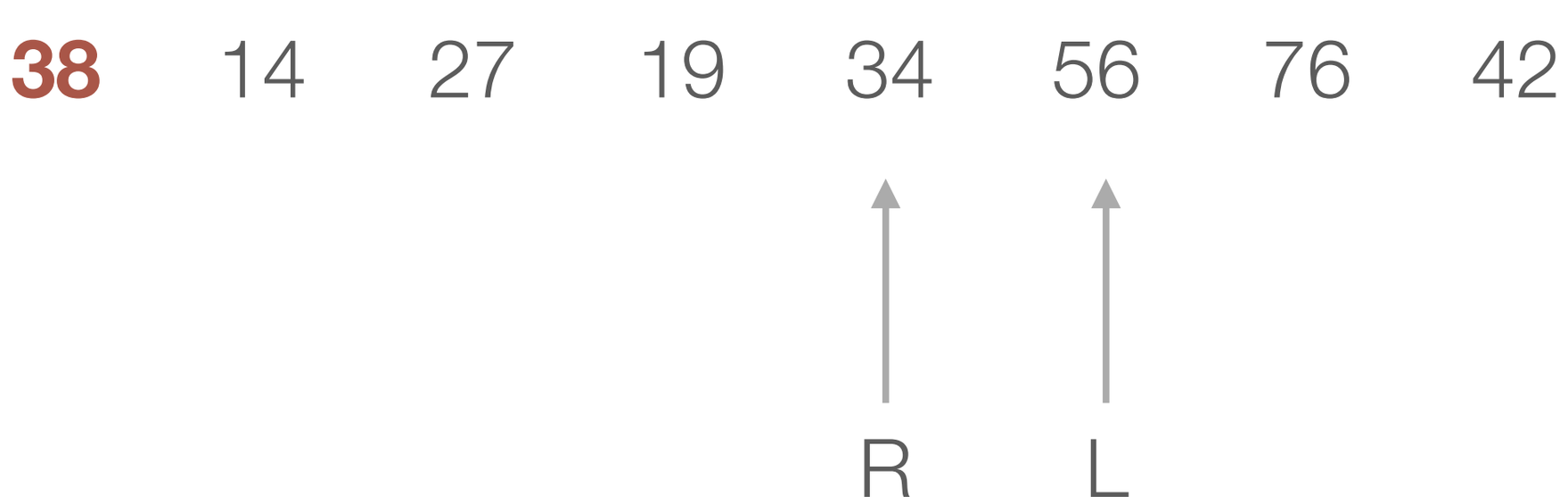
L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

Partition

Pivot: 38



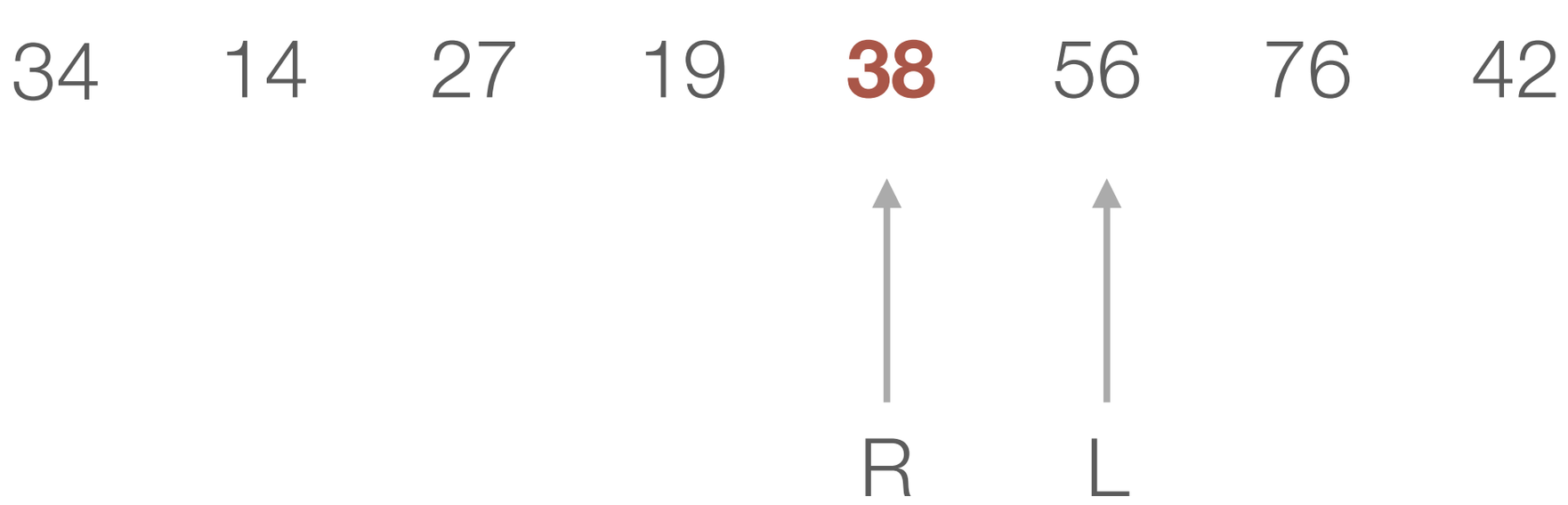
L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

Partition

Pivot: 38



L: Find the next number > pivot

R: Find the next number ≤ pivot

Quick Sort (cont'd)

```
Algorithm Quicksort (X, n);
begin
    Q_Sort(1, n)
end
```

```
procedure Q_Sort (Left, Right);
begin
    if Left < Right then
        Partition(X, Left, Right);
        Q_Sort(Left, Middle - 1);
        Q_Sort(Middle + 1, Right)
end
```

Time complexity: $O(n^2)$, but $O(n \log n)$ in average

Quick Sort (cont'd)

```
Algorithm Partition (X, Left, Right);
begin
    pivot := X[Left];
    L := Left; R := Right;
    while L < R do
        while X[L] ≤ pivot and L ≤ Right do L := L + 1;
        while X[R] > pivot and R ≥ Left do R := R - 1;
        if L < R then swap(X[L], X[R]);
    Middle := R;
    swap(X[Left], X[Middle])
end
```

Quick Sort (cont'd)

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
1	2	4	5	3	(6)	12	9	15	7	10	13	8	11	16	14
(1)	2	4	5	3	(6)	12	9	15	7	10	13	8	11	16	14
(1)	(2)	4	5	3	(6)	12	9	15	7	10	13	8	11	16	14
(1)	(2)	3	(4)	5	(6)	12	9	15	7	10	13	8	11	16	14
(1)	(2)	3	(4)	5	(6)	8	9	11	7	10	(12)	13	15	16	14
(1)	(2)	3	(4)	5	(6)	7	(8)	11	9	10	(12)	13	15	16	14
(1)	(2)	3	(4)	5	(6)	7	(8)	10	9	(11)	(12)	13	15	16	14
(1)	(2)	3	(4)	5	(6)	7	(8)	9	(10)	(11)	(12)	13	15	16	14
(1)	(2)	3	(4)	5	(6)	7	(8)	9	(10)	(11)	(12)	(13)	15	16	14
(1)	(2)	3	(4)	5	(6)	7	(8)	9	(10)	(11)	(12)	(13)	14	(15)	16

Figure 6.12 An example of quicksort. The first line is the initial input. A new pivot is selected in each line. The pivots are circled. When a single number appears between two pivots it is obviously in the right position.

Source: [Manber 1989]

Average-Case Complexity of Quick Sort

When X[i] is selected (at random) as the pivot

$$T(n) \quad = \quad n-1+T(i-1)+T(n-i), \text{ where } n \geq 2$$

The average running time will then be

$$\begin{aligned} T(n) &= n-1+\tfrac{1}{n}\Sigma_{i=1}^{n}(T(i-1)+T(n-i)) \\ &= n-1+\tfrac{1}{n}\Sigma_{i=1}^{n}T(i-1)+\tfrac{1}{n}\Sigma_{i=1}^{n}T(n-i) \\ &= n-1+\tfrac{1}{n}\Sigma_{j=0}^{n-1}T(j)+\tfrac{1}{n}\Sigma_{j=0}^{n-1}T(j) \\ &= n-1+\tfrac{2}{n}\Sigma_{i=0}^{n-1}T(i) \end{aligned}$$

Solving this recurrence relation with full history, T(n) = O(n log n)

Heap Sort

- Input: an array A[1..n]
- Heap sort:
  - Rearrange the array A[1..n] to form a heap
  - Exchange A[1] with A[n]
  - Rearrange the array A[1..n-1] to form a heap
  - Exchange A[1] with A[n - 1]
  - …

Heap Sort (cont'd)

```
Algorithm Heapsort (X, n);
begin
    Build_Heap(X);
    for i := n downto 2 do
        swap(X[1], X[i]);
        Rearrange_Heap(i - 1)
end                                   Time complexity: O(n log n)
```

Rearrange_Heap: basically the same procedure as Remove_Max_from_Heap

The problem now is how to build a heap

Building a Heap

Problem

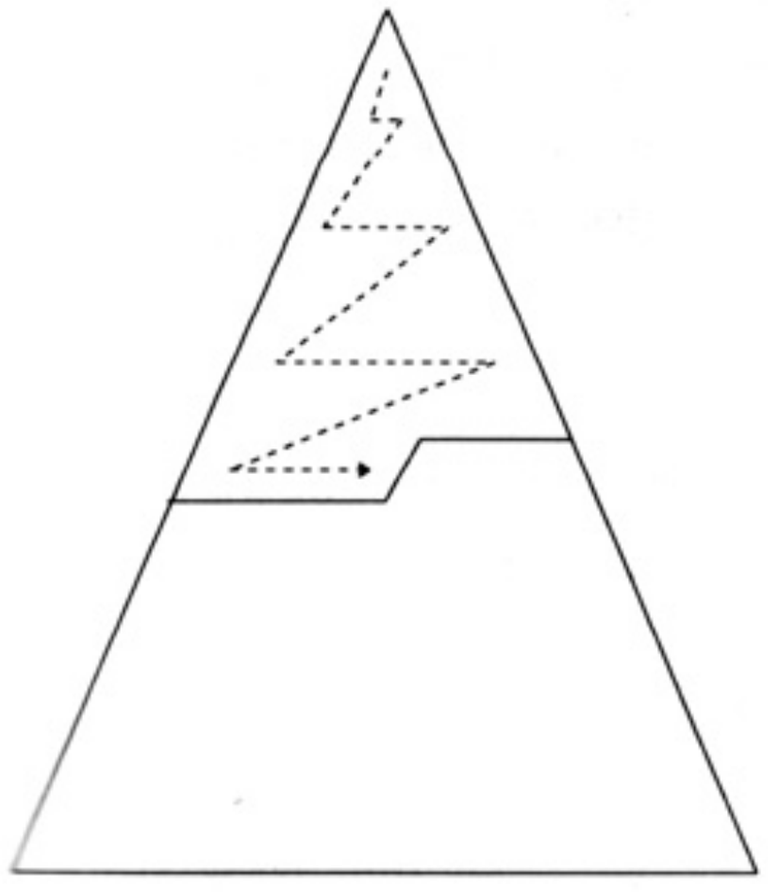
Given an array A[1..n] of elements in an arbitrary order, rearrange the elements so that the array satisfies the heap property

Two natural ways:

- Top down
- bottom up

Building a Heap (cont'd)

- Induction hypothesis (top down): The array A[1..i] is a heap



Source: [Manber 1989]

Building a Heap (cont'd)

- The number of comparisons in the worst case of the insertion of A[i] is $\lfloor \log_2(i+1) \rfloor$

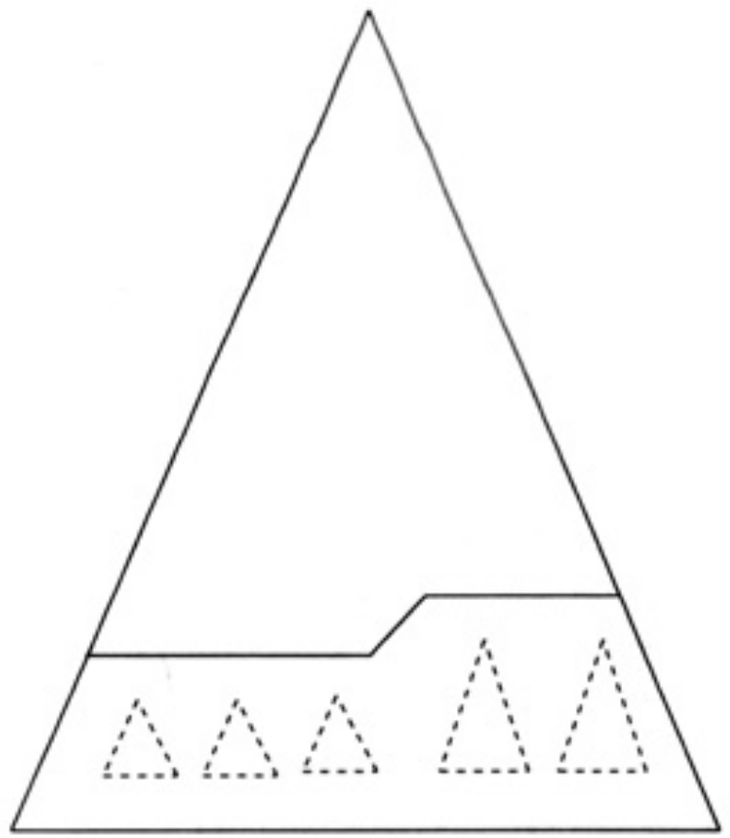
- The complexity of the top down approach:

$$\sum_{i=1}^{n} \lfloor \log_2 i \rfloor \geq \sum_{i=\frac{n}{2}}^{n} \lfloor \log_2 i \rfloor \geq \frac{n}{2} \lfloor \log_2 \frac{n}{2} \rfloor = \Omega(n \log n)$$

Source: [Manber 1989]

Building a Heap (cont'd)

- Induction hypothesis (bottom up): All the trees represented by the array A[i+1..n] satisfy the heap condition



A[⌊n/2⌋ +1..n] are leaves, of which each is a heap

Source: [Manber 1989]

Building a Heap (cont'd)

- The number of comparisons in the worst case of the insertion of A[i] is $2 \lfloor \log_2(n/i) \rfloor$

- The complexity of the bottom up approach is at most twice the sum of the heights of all nodes in the tree

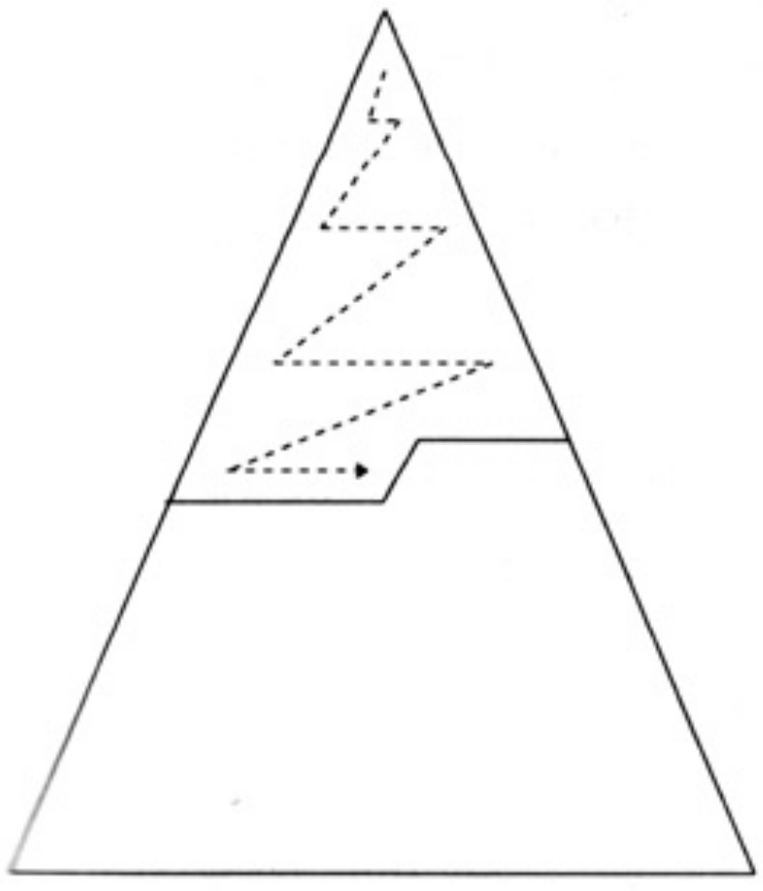
- Let H(i) denote the sum of heights of all nodes in the complete binary tree of height i

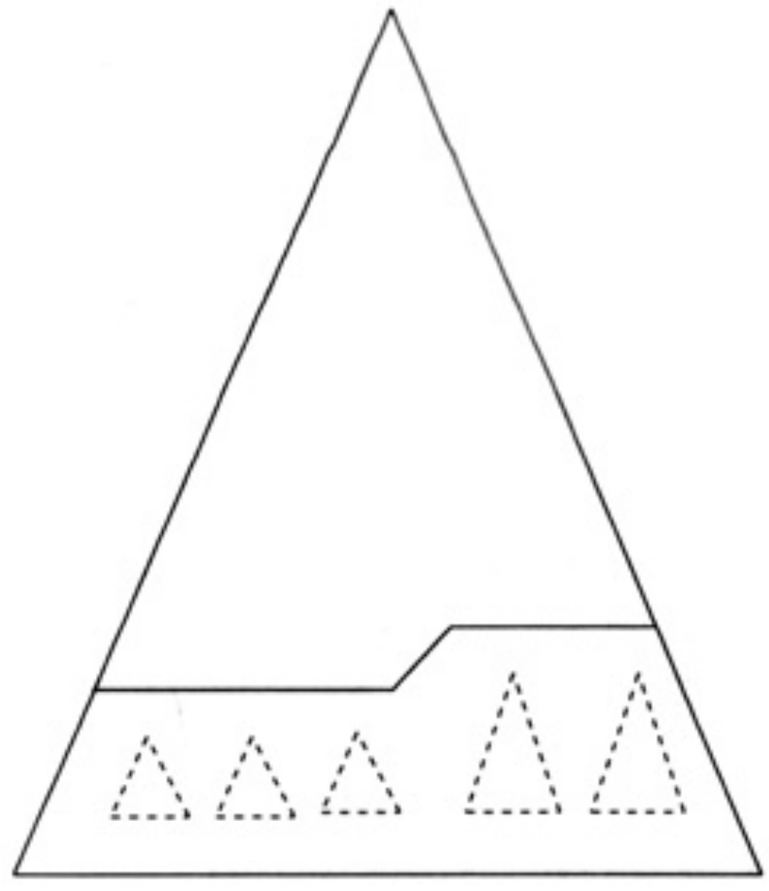
  - $H(i) = 2H(i - 1) + i, H(0) = 0$

  - We get $H(i) = 2^{i + 1} - (i + 2)$ by solving the recurrence relation

- The complexity of the bottom up approach is O(n)

Building a Heap (cont'd)





Why the bottom up approach is better?

Source: [Manber 1989]

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	(14)	15	7	3	13	4	11	16	(1)
2	6	8	5	10	9	(16)	14	15	7	3	13	4	11	(12)	1
2	6	8	5	10	(13)	16	14	15	7	3	(9)	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	(15)	10	13	16	14	(5)	7	3	9	4	11	12	1
2	6	(16)	15	10	13	(12)	14	5	7	3	9	4	11	(8)	1
2	(15)	16	(14)	10	13	12	(6)	5	7	3	9	4	11	8	1
(16)	15	(13)	14	10	(9)	12	6	5	7	3	(2)	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped)

A Lower Bound for Sorting

- A ***lower bound*** for a particular problem is a proof that no algorithm can solve the problem better

- We typically define a ***computation model*** and consider only those algorithms that fit in the model

- ***Decision trees*** model computations performed by ***comparison-based*** algorithms

- A decision tree is a binary tree where

  - each internal node is associated with a ***query*** whose outcome is one of two possibilities, each associated with one of the emanating branches, and

  - each leaf is associated with a possible output

A Lower Bound for Sorting (cont'd)

Theorem

Every decision-tree algorithm for sorting has height Ω(n log n)

- Input for sorting: any sequence $x_1, x_2, \ldots, x_n$
- Output for sorting: the sorted sequence, which is a permutation of the input
- A sorting algorithm has to accept all possible inputs, and thus may output any permutation
- The total number of permutations (represented by the leaves of the decision tree) on n elements is n!
- The complexity (the height of the decision tree) is thus:

$$\log_2(n!) = \log_2(\sqrt{2\pi n}[\tfrac{n}{e}]^n(1 + O(1/n))) = \Omega(n \log n)$$

A Lower Bound for Sorting (cont'd)

- This kind of a lower bound is called an ***information-theoretic*** lower bound
  - It does not depend at all on the computation
  - It depends only on the amount of information contained in the output
- Can we build a decision tree where each internal node has three children?
  - Yes, but the height will be $\log_3 n!$, which is still $\Omega(n \log n)$

Order Statistics: Maximum and Minimum

Problem

Find the maximum and minimum elements in a given sequence

The obvious solution requires (n - 1) + (n - 2) = 2n - 3 comparisons between elements

Can we do better?

Order Statistics: Maximum and Minimum

Problem

Find the maximum and minimum elements in a given sequence

The obvious solution requires (n - 1) + (n - 2) = 2n - 3 comparisons between elements

Can we do better?

Solve the problem for n using the solution of the problem for n - 2 elements

Order Statistics: Maximum and Minimum

Problem

Find the maximum and minimum elements in a given sequence

The obvious solution requires (n - 1) + (n - 2) = 2n - 3 comparisons between elements

Can we do better?

Solve the problem for n using the solution of the problem for n - 2 elements

Base case: n = 1 and n = 2

Order Statistics: Maximum and Minimum

Problem

Find the maximum and minimum elements in a given sequence

The obvious solution requires (n - 1) + (n - 2) = 2n - 3 comparisons between elements

Can we do better?

Solve the problem for n using the solution of the problem for n - 2 elements

Base case: n = 1 and n = 2

Inductive step: x_{n-1}, x_n, MAX_{n-2}, and MIN_{n-2}

Order Statistics: Maximum and Minimum

Problem

Find the maximum and minimum elements in a given sequence

The obvious solution requires (n - 1) + (n - 2) = 2n - 3 comparisons between elements

Can we do better?

Solve the problem for n using the solution of the problem for n - 2 elements

Base case: n = 1 and n = 2

Inductive step: x_{n-1}, x_n, MAX_{n-2}, and MIN_{n-2}

Approximately 3n/2 comparisons

Order Statistic: Kth-Smallest Element

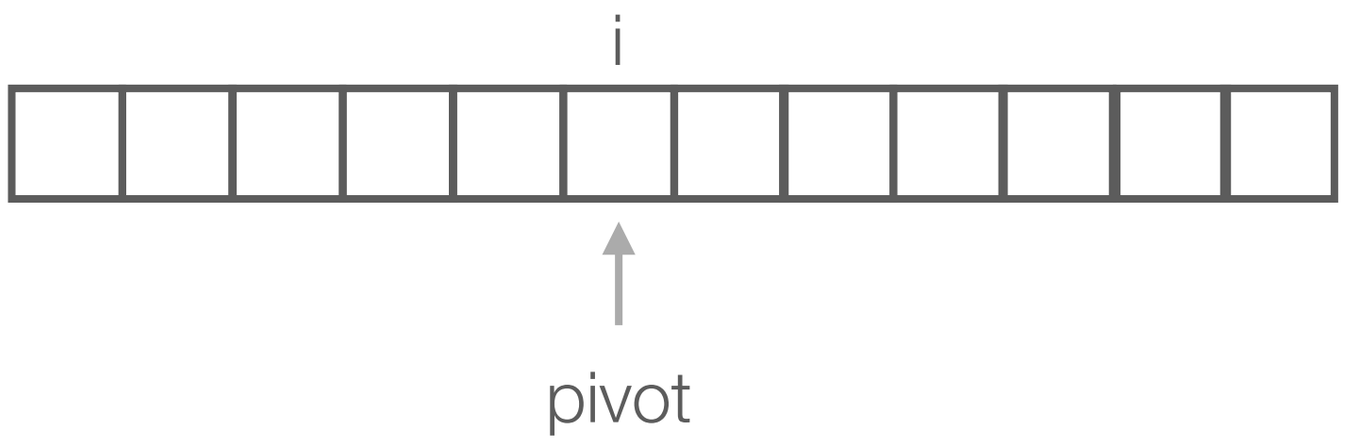
Problem

Given a sequence $S = x_1, x_2, \ldots, x_n$ of elements, and an integer k such that $1 \leq k \leq n$, find the kth-smallest element in S

Approach 1: Find the minimum element k times

Approach 2: Sort the sequence

Approach 3: Recall the pivot in quick sort



Order Statistic: Kth-Smallest Element (cont'd)

```
Algorithm Selection (X, n, k);
begin
    if (k < 1) or (k > n) then print "error"
    else S := Select(1, n, k)
end
```

Complexity: O(n) in average

```
procedure Select (Left, Right, k);
begin
    if Left = Right then Select := Left
    else
        Partition(X, Left, Right);
        Let Middle be the output of Partition;
        if Middle - Left + 1 ≥ k then Select(Left, Middle, k)
        else Select(Middle + 1, Right, k - (Middle - Left + 1))
end
```

Finding a Majority

Problem

Given a sequence of numbers, find the majority in the sequence or determine that none exists

A number is a ***majority*** in a sequence of n numbers if it occurs more than n/2 times in the sequence

Approach 1: Sort and count

Approach 2: Use the median-finding algorithm

Approach 3: Recall the algorithm for finding a celebrity

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

a	a	b	d	b	b	a	a	b	a	a	a

Finding a Majority (cont'd)

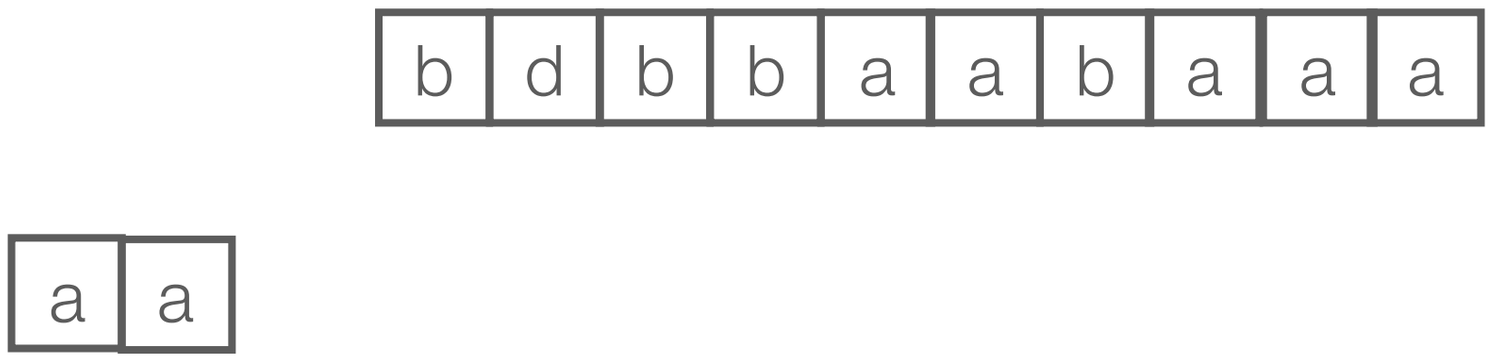
If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

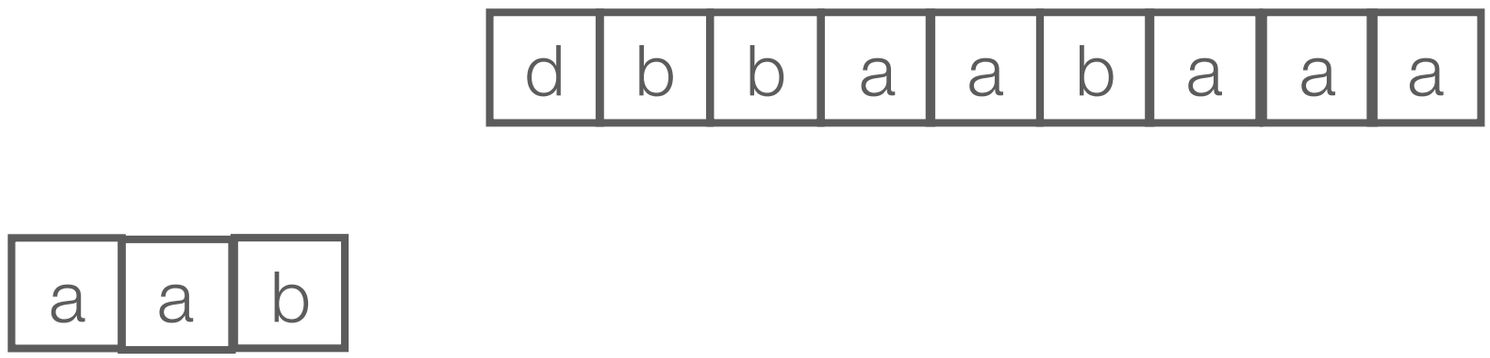
If $x_i = x_j$, what can we conclude?



Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

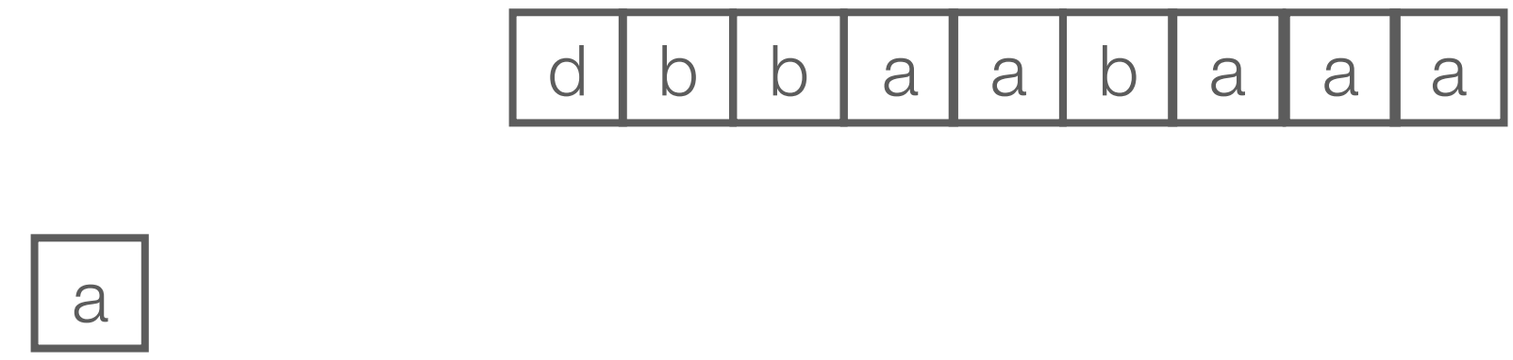
If $x_i = x_j$, what can we conclude?



Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?



Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

b	b	a	a	b	a	a	a

a	d

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

b	b	a	a	b	a	a	a

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

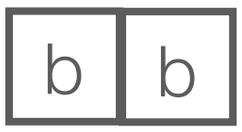
b	a	a	b	a	a	a

b

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

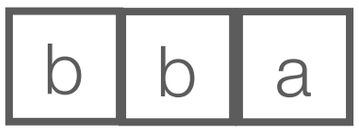
If $x_i = x_j$, what can we conclude?



Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?



Finding a Majority (cont'd)

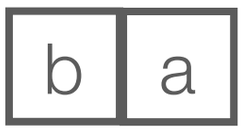
If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?



Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

a

a

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Is the remaining element always the majority?

Finding a Majority (cont'd)

If $x_i \neq x_j$, what can we conclude?

If $x_i = x_j$, what can we conclude?

Is the remaining element always the majority?

b	c	b	c	b	c	b	c	a

Finding a Majority (cont'd)

```
Algorithm Majority (X, n);
begin
    C := X[1]; M := 1;
    for i := 2 to n do
        if M = 0 then C := X[i]; M := 1
        else if C = X[i] then M := M + 1
                else M := M - 1;
    if M = 0 then Majority := -1
    else
        Count := 0;
        for i := 1 to n do
            if X[i] = C then Count := Count + 1;
        if Count > n/2 then Majority := C
        else Majority := -1
end
```