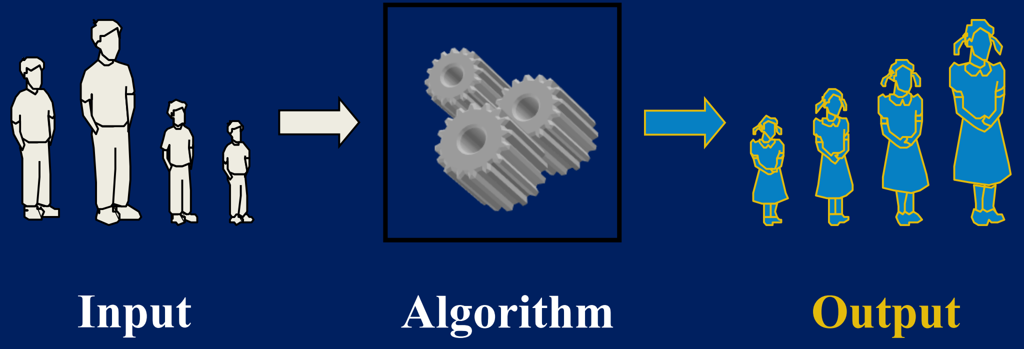
Chapter 6: Non-comparison Sorting



Prof. Anand Singh Jalal

How Fast Can We Sort?

	Best	Average	Worst
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)
Merge Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))

- What is common to all these algorithms?
  - Make **comparisons** between input elements

$a_i < a_j$, $a_i \le a_j$, $a_i = a_j$, $a_i \ge a_j$, or $a_i > a_j$

To sort n elements, comparison sorts **must** make $\Omega(n\lg n)$ comparisons in the best case.

Can we do better?

- Linear sorting algorithms
  - Counting Sort
  - Radix Sort
  - Bucket sort
- Make certain assumptions about the data
- Linear sorts are NOT **“comparison sorts”**

COUNTING SORT ALGORITHM

Counting Sort

- It is linear time sorting algorithm.
- The basic idea of counting sort is to determine, for each input element x, the number of elements less than x.
- This information can be used to place element x directly into its position in the output array.
- The complexity is $O(n + k)$, Where n is the size of input and k is the value of maximum element present in the array.

Counting Sort

$$\text{for } i \leftarrow 1 \text{ to } k$$
$$\quad \text{do } C[i] \leftarrow 0;$$
$$\text{for } j \leftarrow 1 \text{ to } length[A]$$
$$\quad \text{do } C[A[j]] \leftarrow C[A[j]] + 1;$$
$$\text{for } i \leftarrow 2 \text{ to } k$$
$$\quad \text{do } C[i] \leftarrow C[i] + C[i-1];$$
$$\text{for } j \leftarrow length[A] \text{ downto } 1$$
$$\quad \text{do begin}$$
$$\qquad B[C[A[j]]] \leftarrow A[j];$$
$$\qquad C[A[j]] \leftarrow C[A[j]] - 1;$$
$$\quad \text{end - for}$$

$$\boxed{A: 3, 6, 4, 1, 3, 4, 1, 4}$$

$$\boxed{C: 2, 0, 2, 3, 0, 1}$$

C is an array where C[j] refers to how many times j appears in A.

$$\boxed{C: 2, 2, 4, 7, 7, 8}$$

Now C is an array where C[j] refers to how many elements are ≤ j

$$\boxed{A: 3, 6, 4, 1, 3, 4, 1, \hat{4}}$$

$$\boxed{B: \ , \ , \ , \ , \ , \ , 4,}$$

$$\boxed{C: 2, 2, 4, 6, 7, 8}$$

Counting sort …

$C:\ 2, 2, 4, 6, 7, 8$

$A:\ 3, 6, 4, 1, 3, 4, \hat{1}, 4$

$B:\ \ ,1,\ ,\ ,\ ,\ ,4,$

$C: 1, 2, 4, 6, 7, 8$

$A:\ 3, 6, 4, 1, 3, \hat{4}, 1, 4$

$B:\ \ ,1,\ ,\ ,\ ,4, 4,$

$C: 1, 2, 4, 5, 7, 8$

$A:\ 3, 6, \hat{4}, 1, 3, 4, 1, 4$

$B: 1, 1,\ ,3, 4, 4, 4,$

$C:\ 0, 2, 3, 4, 7, 8$

$A:\ 3, \hat{6}, 4, 1, 3, 4, 1, 4$

$B: 1, 1,\ ,3, 4, 4, 4, 6$

$C:\ 0, 2, 3, 4, 7, 7$

$A:\ 3, 6, 4, 1, \hat{3}, 4, 1, 4$

$B:\ \ ,1,\ ,3,\ ,4, 4,$

$C: 1, 2, 3, 5, 7, 8$

$A:\ 3, 6, 4, \hat{1}, 3, 4, 1, 4$

$B: 1, 1,\ ,3,\ ,4, 4,$

$C:\ 0, 2, 3, 5, 7, 8$

$A:\ \hat{3}, 6, 4, 1, 3, 4, 1, 4$

$B: 1, 1, 3, 3, 4, 4, 4, 6$

$C:\ 0, 2, 2, 4, 7, 7$

Counting sort ...

$\text{for } i \leftarrow 1 \text{ to } k$ $\quad O(k)$

$\quad \text{do } C[i] \leftarrow 0;$

$\text{for } j \leftarrow 1 \text{ to } length[A]$ $\quad O(n)$

$\quad \text{do } C[A[j]] \leftarrow C[A[j]] + 1;$

$\text{for } i \leftarrow 2 \text{ to } k$ $\quad O(k)$

$\quad \text{do } C[i] \leftarrow C[i] + C[i-1];$

$\text{for } j \leftarrow length[A] \text{ downto } 1$ $\quad O(n)$

$\quad \text{do begin}$

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

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

$\quad \text{end - for}$

$\boxed{O(n+k)}$

Counting sort is stable

That is, the same value appear in the output array in the same order as they do in the input array.

since we iterated through A backwards and decrement C[i] every time we see i. we preserve the order of duplicates in A.

RADIX SORT ALGORITHM

Radix Sort

- Represents keys as d-digit numbers in some base-k

$$key = x_1x_2...x_d \quad \text{where } 0 \le x_i \le k-1$$

- Example: key=15

$$key_{10} = 15,\ d=2,\ k=10 \quad \text{where } 0 \le x_i \le 9$$

$$key_2 = 1111,\ d=4,\ k=2 \quad \text{where } 0 \le x_i \le 1$$

Radix Sort ...

- Assumptions

  $d = O(1)$ and $k = O(n)$

- Sorting looks at one column at a time
  - For a d digit number, sort the least significant digit first
  - Continue sorting on the next least significant digit, until all digits have been sorted
  - Requires only d passes through the list

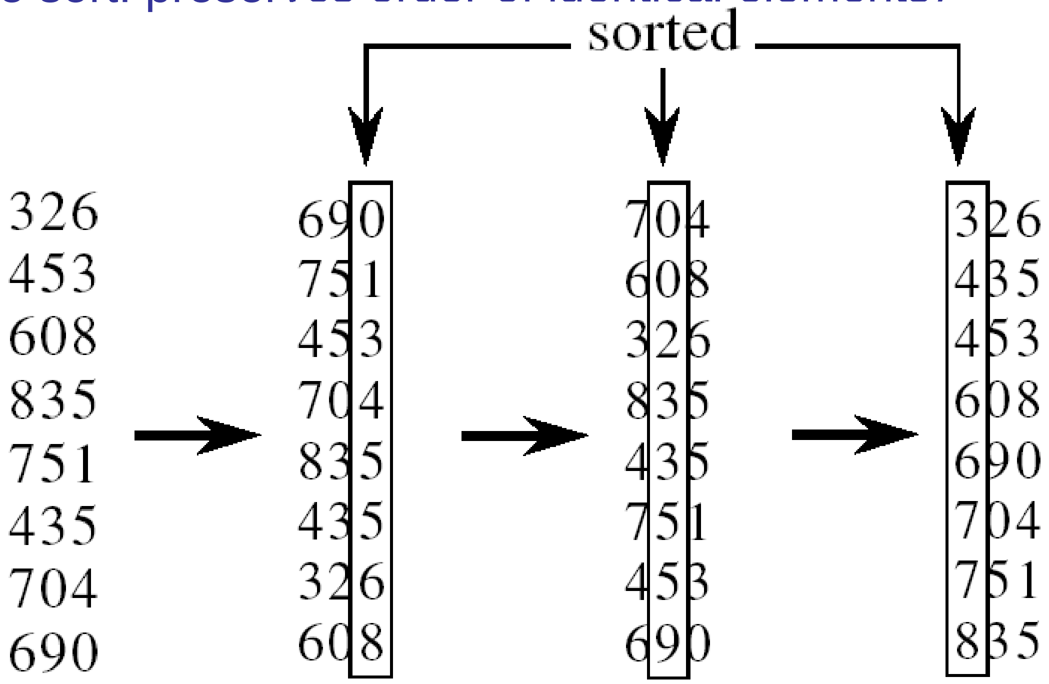
Radix Sort ...

Alg.: RADIX-SORT(A, d)

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

do use a stable sort to sort array A on digit i

(stable sort: preserves order of identical elements)



Analysis of Radix Sort

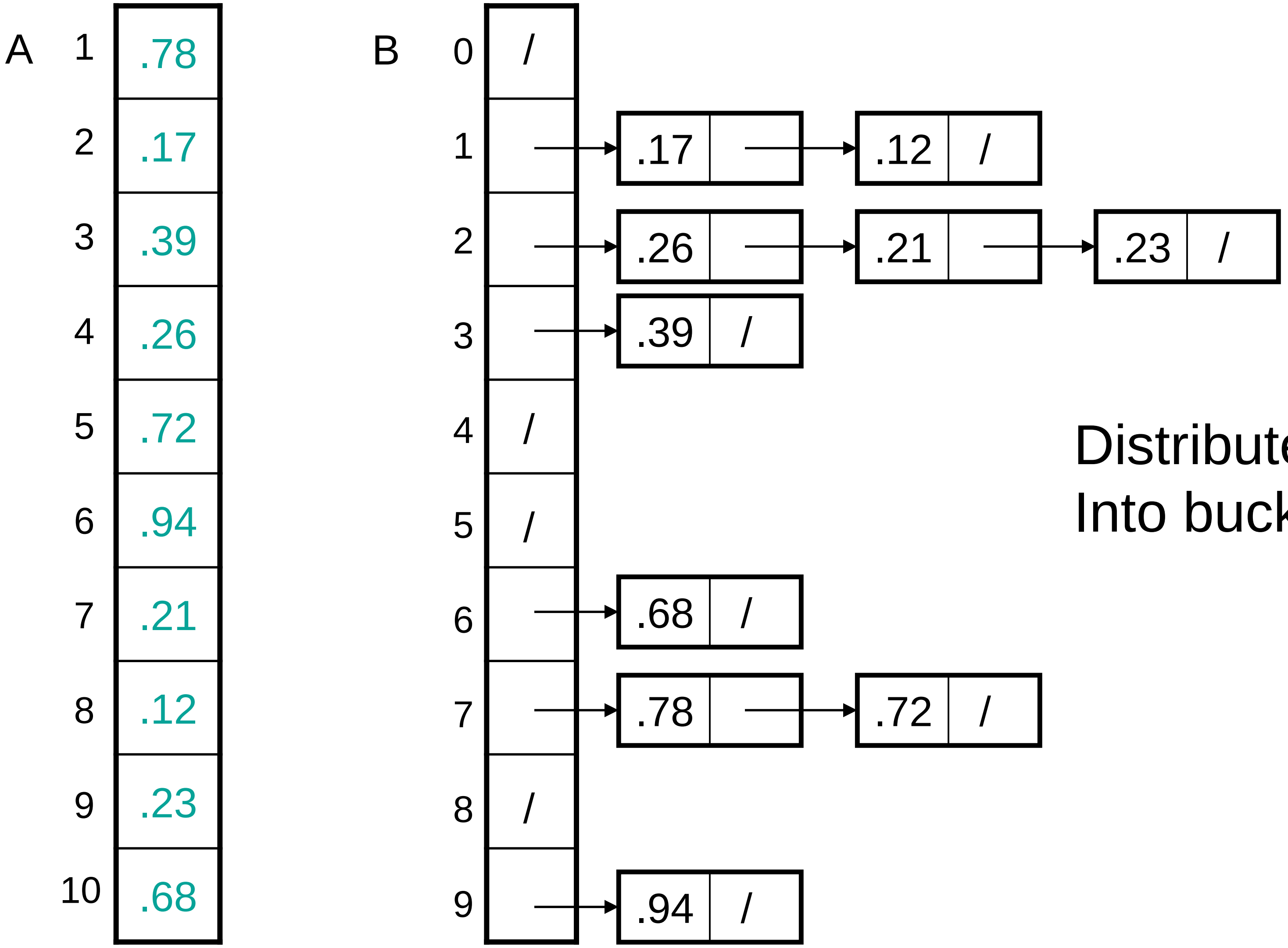
- Given n numbers of d digits each, where each digit may take up to k possible values, RADIX-SORT correctly sorts the numbers in $O(d(n+k))$
  - One pass of sorting per digit takes $O(n+k)$ assuming that we use **counting sort**
  - There are d passes (for each digit)
  - Assuming $d=O(1)$ and $k=O(n)$, running time is $O(n)$

Bucket Sort

Bucket Sort

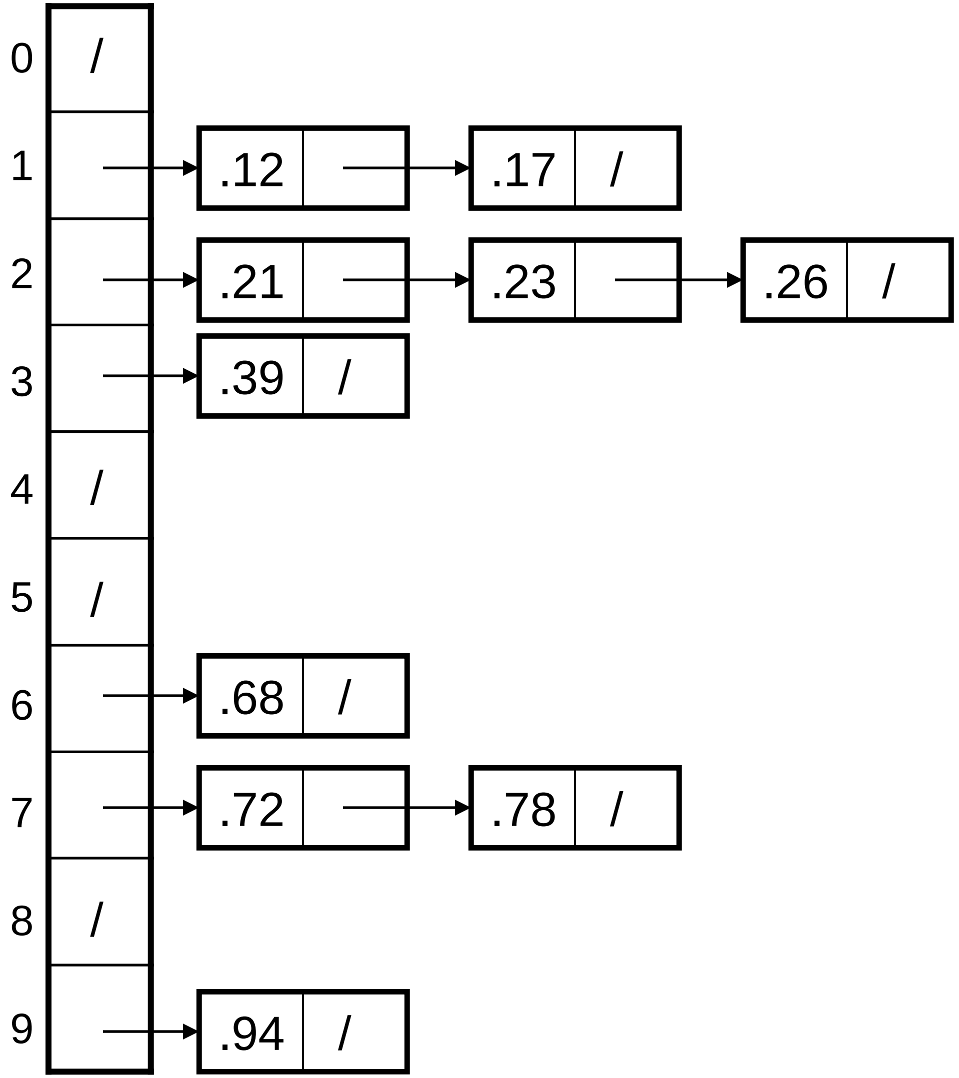
- Assumption:
  - the input is generated by a random process that distributes elements uniformly over [0, 1)
- Idea:
  - Divide [0, 1) into k equal-sized buckets ($k=\Theta(n)$)
  - Distribute the n input values into the buckets
  - Sort each bucket (e.g., using quicksort)
  - Go through the buckets in order, listing elements in each one
- **Input:** A[1 . . n], where $0 \le A[i] < 1$ for all i
- **Output:** elements A[i] sorted

Bucket Sort ..



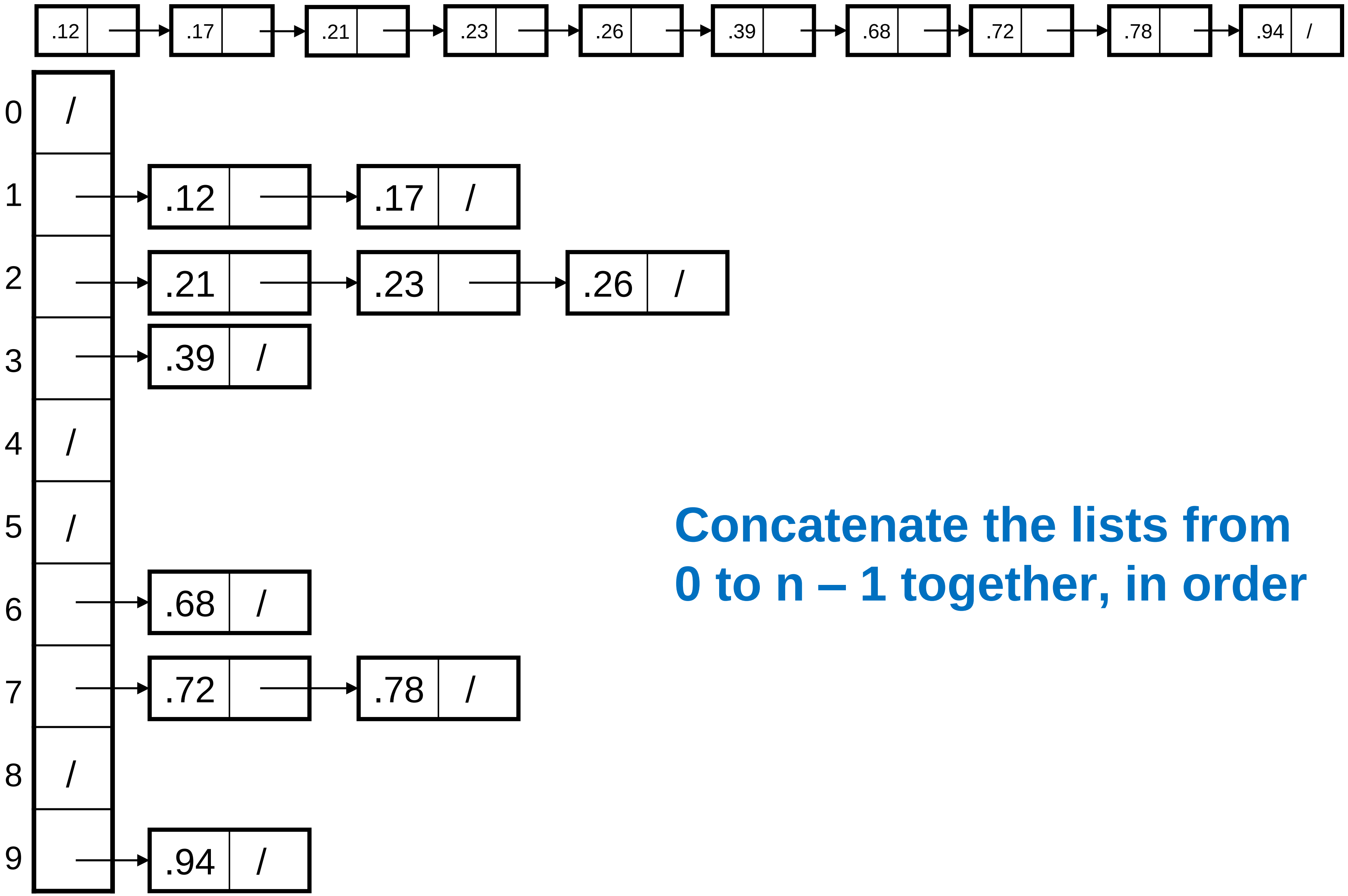
Distribute
Into buckets

Bucket Sort ..



Sort within each bucket

Bucket Sort ..



Concatenate the lists from 0 to n – 1 together, in order

Analysis of Bucket Sort

Alg.: BUCKET-SORT(A, n)

for i ← 1 **to** n

do insert A[i] into list B[⌊nA[i]⌋]

O(n)

for i ← 0 **to** k - 1

do sort list B[i] with quicksort sort

k O(n/k log(n/k))
=O(nlog(n/k)

concatenate lists B[0], B[1], . . . , B[n -1]

together in order

O(k)

return the concatenated lists

O(n) (if k=Θ(n))

Any Questions ?

Dr. Anand Singh Jalal
Professor
Email: asjalal@gla.ac.in