Lecture

Non-Comparison based Sorting Algorithm

Why Non-Comparison Sorting Algorithm ?

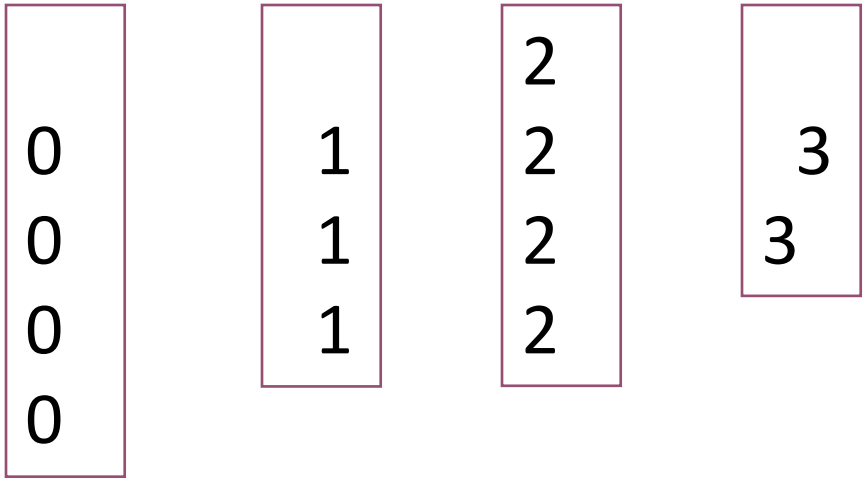
- A sorting algorithm that involves comparing pairs of values can never have a worst-case time better than $O(n \log n)$.
- Under some special conditions, it is possible to design other sorting algorithms that have better worst-case time complexity. These algorithms are:
  - Bucket sort,
  - Counting sort, and
  - Radix sort.

Bucket Sort

- Idea: suppose the values are in the range 0..m-1; start with m empty *buckets* numbered 0 to m-1
- Scan the list and place element s[i] in bucket s[i], and then output the buckets in order
- Will need an array of buckets, and the values in the list to be sorted will be the indexes to the buckets
  - No comparisons will be necessary

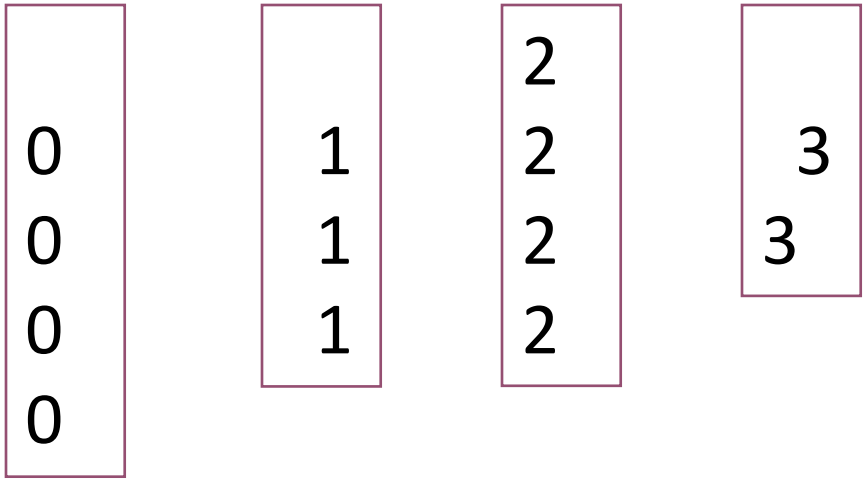
Bucket Sort: Example

0	2	1	2	0	3	2	1	1	0	2	3	0



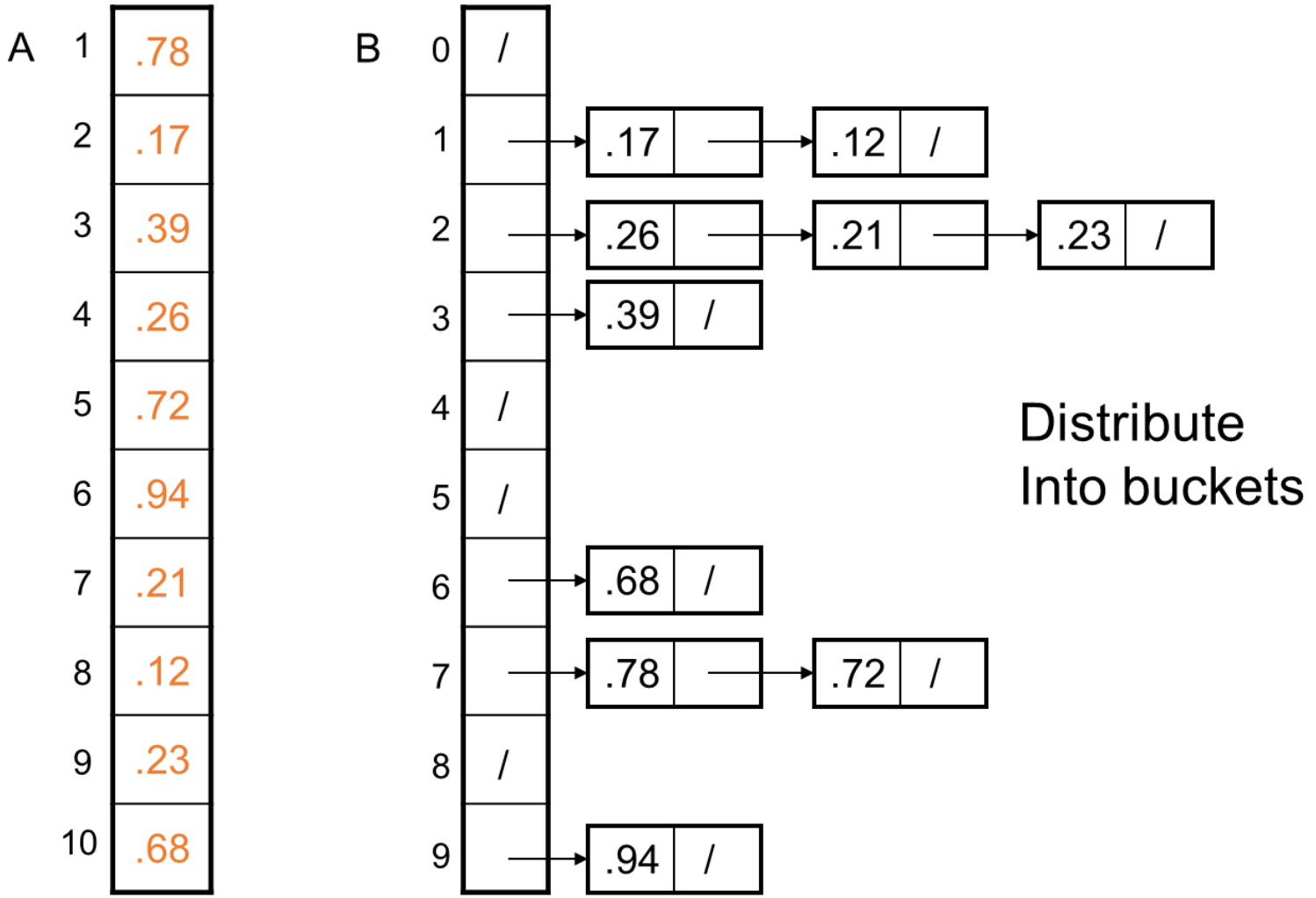
Bucket Sort: Example

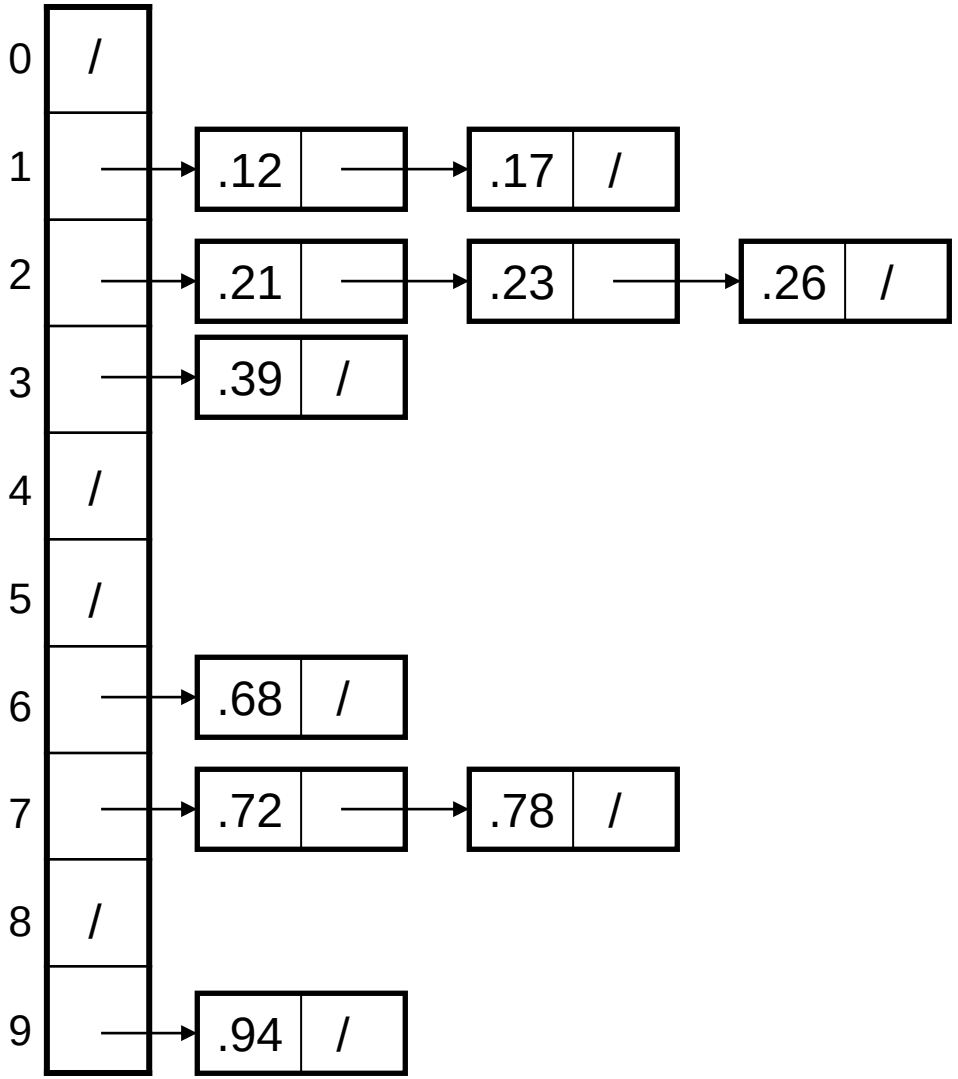
0	2	1	2	0	3	2	1	1	0	2	3	0



0	0	0	0	1	1	1	2	2	2	2	3	3

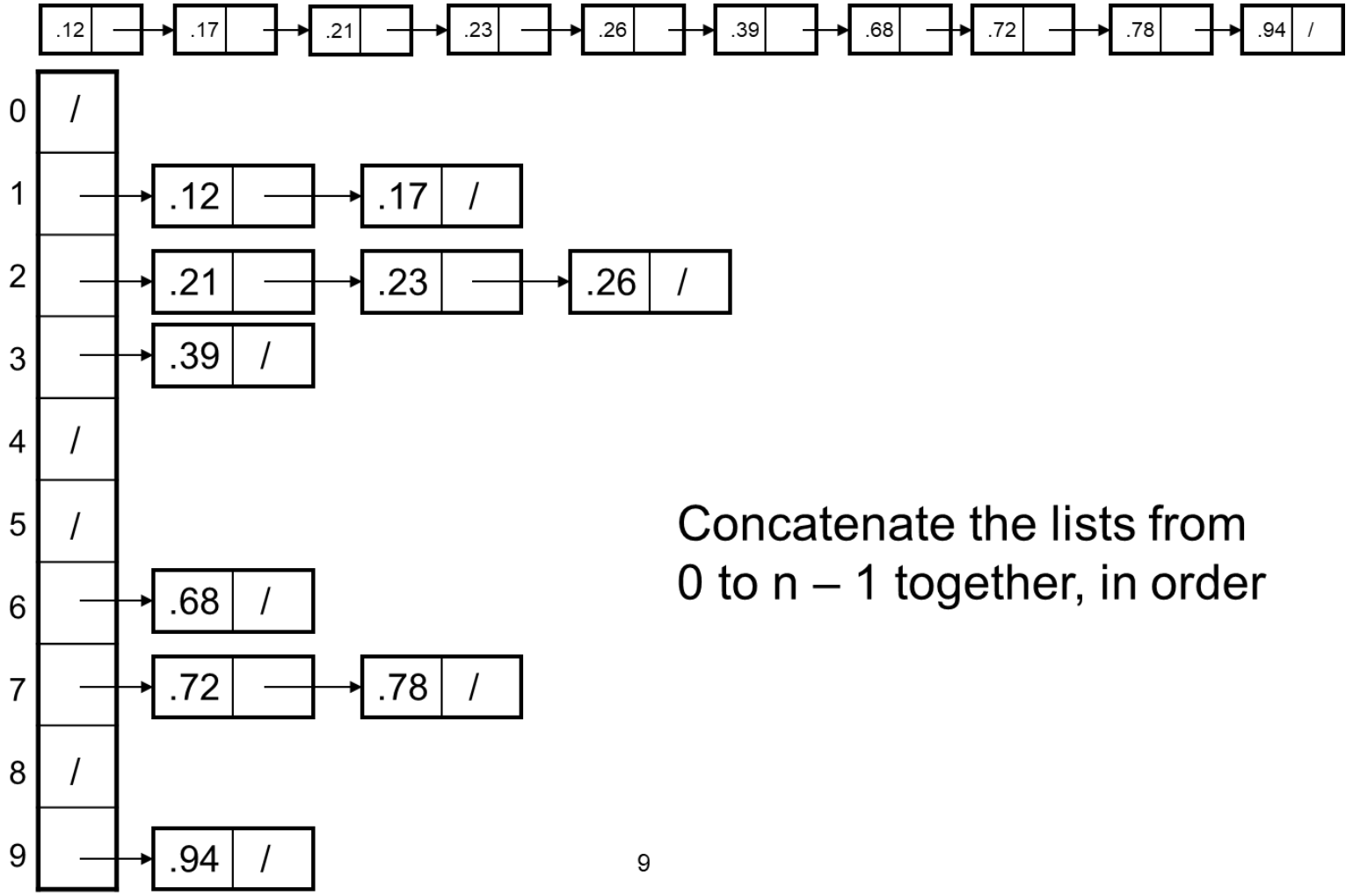
Bucket Sort: Example





Sort within each bucket

Bucket Sort: Example



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

Bucket Sort: Algorithm

Each element A[i] in the array satisfies $0 \leq A[i] < 1$.
B[0….n-1] is an auxiliary array of linked lists (buckets).

Bucket Sort (A)

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

Time Complexity			Space Complexity
Best Case	Average Case	Worst Case	Worst case
$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$

Bucket Sort : Exercise

.67	.35	.44	.68	.41	.63	.17	.34	.66	.87

Radix Sort

Sorts the elements by comparing significant digits from least significant digit to most significant digit.

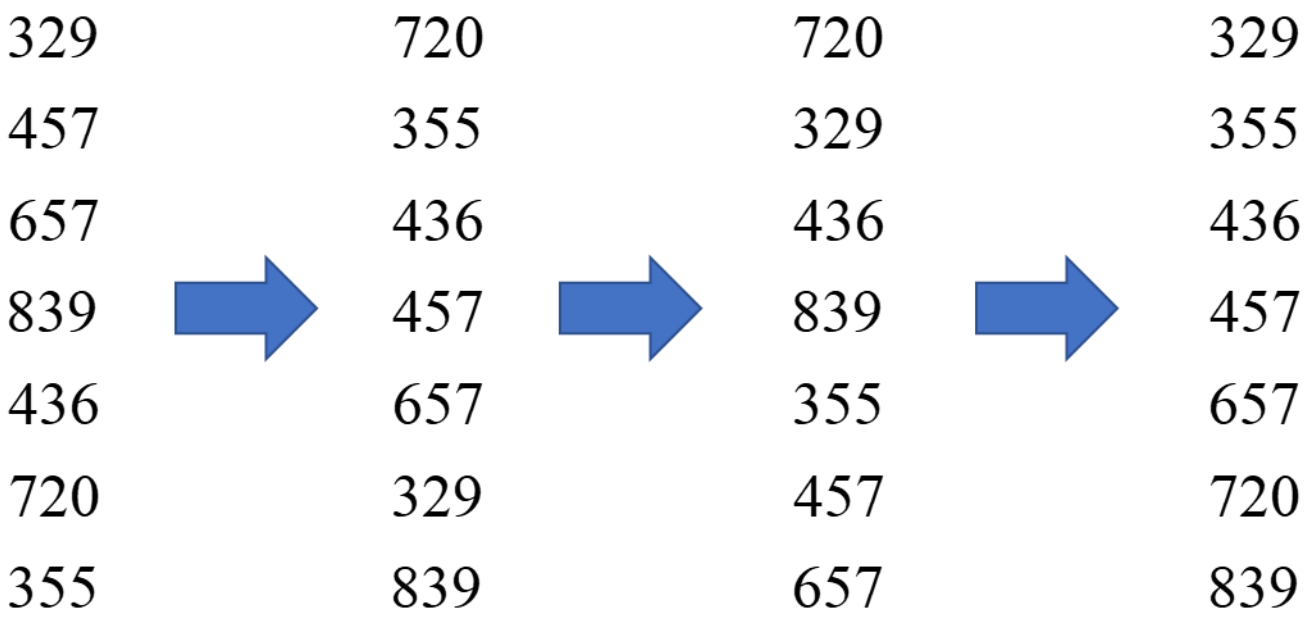
Special Condition:

The values to be sorted are sequences of length ‘k’.

Benefit of applying special condition:

- Helps in sorting each digit/character.

Radix Sort


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

Radix Sort: Time complexity

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      use a stable sort to sort array A on digit i
```

Time Complexity			Space Complexity
Best Case	Average Case	Worst Case	Worst case
$\Omega(d(n+k))$	$\Theta(d(n+k))$	$O(d(n+k))$	$O(n+k)$

Radix Sort: Exercise

index	0	1	2	3	4	5	6	7	8	9	10	11
value	71**4**	12**8**	20**6**	3**4**	72**2**	**8**	14**2**	53**3**	64**6**	2**9**	24**0**	37**3**

Thank You