Honors Data Structures

Lecture 19: Sorting II

4/4/2022

Daniel Bauer

Comparison-Based Sorting Algorithms

	T_{Worst}	T_{Best}	T_{Avg}	Space	Stable?
Insertion Sort	$\Theta(N^2)$	$\Theta(N)$	$\Theta(N^2)$	$O(1)$	✓
Heap Sort	Θ				✗
Merge Sort	Θ				✓
Quick Sort	$\Theta(N^2)$	$\Theta(NlogN)$	$\Theta(NlogN)$	$O(1)$	✗

$\Omega(N \log N)$ worst case lower bound on comparison based general sorting!

gray entries: not shown in class

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	1	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	1	0	0	0	0	0	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	1	1	0	0	0	0	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	1	1	1	0	0	0	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	1	2	1	0	0	0	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	1	2	1	1	0	0	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i].

A	1	8	2	3	2	4	6	1

count	0	1	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

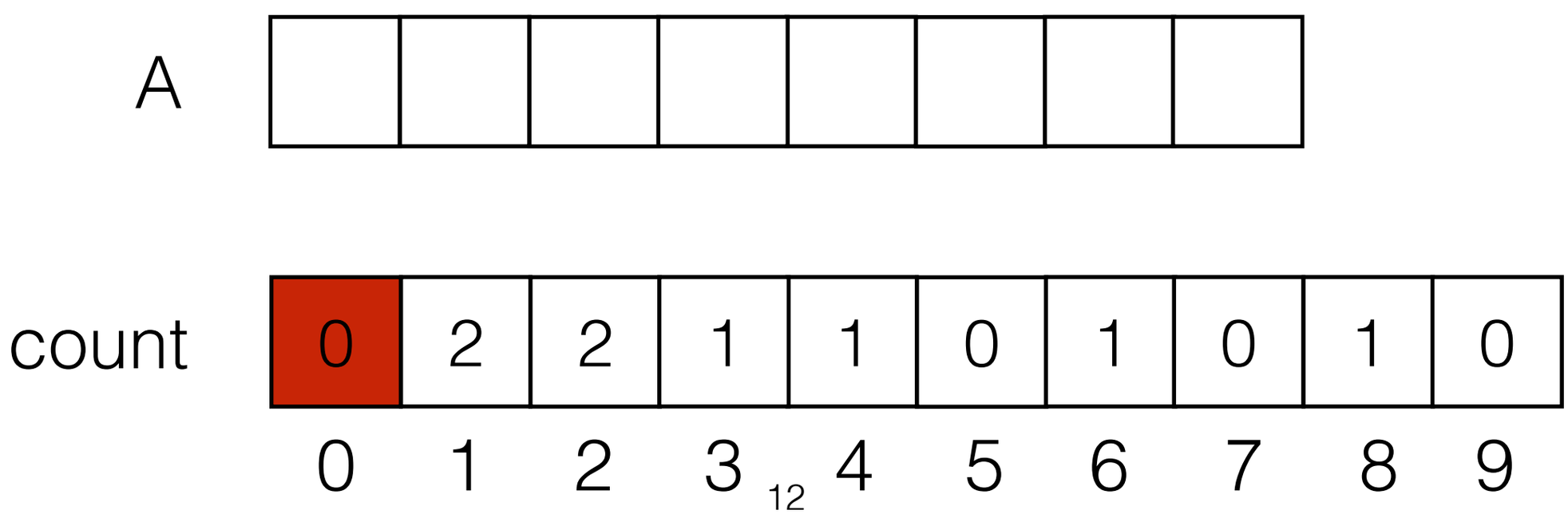
- Assume we know there are *M* possible values.
- Keep an array `count` of length *M*.
- Scan through the input array *A* and for each *i* increment `count`[A_i]. O(N)

A	1	8	2	3	2	4	6	1

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.



Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1								
count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1	2	2						

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1	2	2	3					

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1	2	2	3	4				

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1	2	2	3	4				

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1	2	2	3	4	6			

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1	2	2	3	4	6			

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

A	1	1	2	2	3	4	6	8		

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Bucket Sort

- Then iterate through `count`. For each *i* write `count`[*i*] copies of *i* to *A*.

O(M)

Total time for Bucket Sort:O(N +M)

A	1	1	2	2	3	4	6	8		

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Making Bucket Sort Stable

Example: Sort the following array by the last digit.

A	11	08	52	03	02	04	06	32

- Instead of the count array, keep items on ArrayList.

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0	
1	11
2	
3	
4	
5	
6	
7	
8	
9	

- Goal: Sort the array by the last digit.
- Idea: Instead of an integer count, keep a list of items for each possible last digit.

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0	
1	11
2	
3	
4	
5	
6	
7	
8	08
9	

Making Bucket Sort Stable

A	11	08	52	03	02	04	06	31

0	
1	11
2	52
3	
4	
5	
6	
7	
8	08
9	

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0	
1	11
2	52
3	03
4	
5	
6	
7	
8	08
9	

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0	
1	11
2	52 02
3	03
4	
5	
6	
7	
8	08
9	

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0	
1	11
2	52 02
3	03
4	04
5	
6	
7	
8	08
9	

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0	
1	11
2	52 02
3	03
4	04
5	
6	06
7	
8	08
9	

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0		
1	11	31
2	52	02
3	03	
4	04	
5		
6	06	
7		
8	08	
9		

Making Bucket Sort Stable

A							
11	08	52	03	02	04	06	31

0	
1	11 31
2	52 02
3	03
4	04
5	
6	06
7	
8	08
9	

Read off the results:

11	31	52	02	03	04	06	08

Time: O(N+M)
Space: O(N+M)

Counting Sort

A	11	08	52	03	02	04	06	31

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	0	2	4	5	6	6	7	7	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	0	2	4	5	6	6	7	7	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	0	2	4	5	6	6	7	7	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

11							

Write to correct offset in output array, then increment offset.

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	2	4	5	6	6	7	7	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

11							8

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	2	4	5	6	6	7	7	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

11		52					8

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	2	4	5	6	6	7	8	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

11		52		03			8

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	3	4	5	6	6	7	8	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

11		52	02	03			8

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	3	5	5	6	6	7	8	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

11		52	02	03	04		8

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	4	5	5	6	6	7	8	8
	0	1	2	3	4	5	6	7	8	9

Counting Sort

A	11	08	52	03	02	04	06	31

11		52	02	03	04	06	8

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	4	5	6	6	6	7	8	8
	0	1	2	3	4	5	6	7	8	9

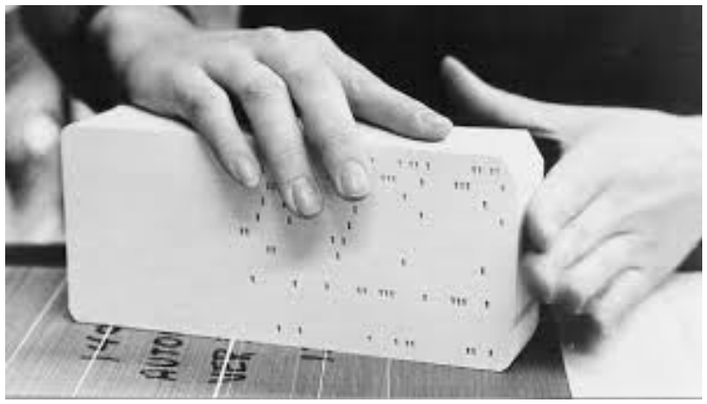
Counting Sort

A	11	08	52	03	02	04	06	31

11	31	52	02	03	04	06	8

count	0	2	2	1	1	0	1	0	1	0
	0	1	2	3	4	5	6	7	8	9

offset	0	1	4	5	6	6	7	7	8	8
	0	1	2	3	4	5	6	7	8	9



Radix Sort

- Generalization of Bucket sort for Large M.
- Assume M contains all base b numbers up to b^d-1 (e.g. all base-10 integers up to 10^3)
- Do d passes over the data, using Bucket Sort for each digit. Total runtime: O(d * (b+N))
- Bucket sort is stable!

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

Radix Sort

064 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	
1	
2	
3	
4	064
5	
6	
7	
8	
9	

- Bucket sort according to least significant digit.

Radix Sort

064 008 216 512 027 729 000 001 343 125

0	
1	
2	
3	
4	064
5	
6	
7	
8	008
9	

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	
1	
2	
3	
4	064
5	
6	216
7	
8	008
9	

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	
1	
2	512
3	
4	064
5	
6	216
7	
8	008
9	

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	
1	
2	512
3	
4	064
5	
6	216
7	027
8	008
9	

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	
1	
2	512
3	
4	064
5	
6	216
7	027
8	008
9	729

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	000
1	
2	512
3	
4	064
5	
6	216
7	027
8	008
9	729

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	000
1	001
2	512
3	
4	064
5	
6	216
7	027
8	008
9	729

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	000
1	001
2	512
3	003
4	064
5	
6	216
7	027
8	008
9	729

- Bucket sort according to least significant digit.

Radix Sort

06**4** 00**8** 21**6** 51**2** 02**7** 72**9** 00**0** 00**1** 34**3** 12**5**

0	000
1	001
2	512
3	003
4	064
5	125
6	216
7	027
8	008
9	729

- Bucket sort according to least significant digit.

Radix Sort

000 001 512 343 064 125 216 027 008 729

0	000
1	001
2	512
3	003
4	064
5	125
6	216
7	027
8	008
9	729

- read off new sequence

Radix Sort

000 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000
1	
2	
3	
4	
5	
6	
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001
1	
2	
3	
4	
5	
6	
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001
1	512
2	
3	
4	
5	
6	
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001
1	512
2	
3	
4	343
5	
6	
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001
1	512
2	
3	
4	343
5	
6	064
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001
1	512
2	125
3	
4	343
5	
6	064
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001
1	512 216
2	125
3	
4	343
5	
6	064
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001
1	512 216
2	125 027
3	
4	343
5	
6	064
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001 008
1	512 216
2	125 027
3	
4	343
5	
6	064
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

0**0**0 0**0**1 5**1**2 3**4**3 0**6**4 1**2**5 2**1**6 0**2**7 0**0**8 7**2**9

0	000 001 008
1	512 216
2	125 027 729
3	
4	343
5	
6	064
7	
8	
9	

- Bucket sort according to second-least significant digit.

Radix Sort

000 001 008 512 216 125 027 729 343 064

0	000 001 008
1	512 216
2	125 027 729
3	
4	343
5	
6	064
7	
8	
9	

- read off new sequence

Radix Sort

000 **0**01 **0**08 **5**12 **2**16 **1**25 **0**27 **7**29 **3**43 **0**64

0	000 001 008 027 064
1	125
2	216
3	343
4	
5	512
6	
7	729
8	
9	

- Bucket sort according to third-least significant digit.