

FIT2004

Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Compane, Aamir Cheema, Arun Konagurthu and Lloyd Allison



Faculty of Information Technology, Monash University

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act. Do not remove this notice

Ready?

Agenda

- Sorting Algorithms
 - Comparison based
 - Selection
 - Insertion
 - Non-comparison based (the IMBA ones)
 - Counting
 - Radix

Let us begin...

Sorting

Non-Comparison

- We can sort without comparing elements in a list!

Sorting

Non-Comparison

- We can sort without comparing elements in a list!
 - Counting sort
 - Radix sort

Questions?

- Very simple concept
- I am sure we all know this...
- Now let us begin with a list

4	2	1	3	1	4	5
---	---	---	---	---	---	---

- Very simple concept
- I am sure we all know this...
- Now let us begin with a list

4	2	1	3	1	4	5
---	---	---	---	---	---	---

- What is the maximum number?

- Very simple concept
- I am sure we all know this...
- Now let us begin with a list

4	2	1	3	1	4	5
---	---	---	---	---	---	---

- What is the maximum number?
 - 5 but how do we know?

- Very simple concept
- I am sure we all know this...
- Now let us begin with a list

4	2	1	3	1	4	5
---	---	---	---	---	---	---

- What is the maximum number?
 - 5 but how do we know? Loop through the list in $O(N)$

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---

- We know max is 5

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---

Anyone noticed
the list is crooked?
#OCDtrigger

- We know max is 5

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---

- We know max is 5

0	1	2	3	4	5
---	---	---	---	---	---

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---

- We know max is 5

0	1	2	3	4	5

- Out input


4	2	1	3	1	4	5
---	---	---	---	---	---	---

- We know max is 5

0	1	2	3	4	5
0	0	0	0	0	0

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	0	0	0	0	0

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	0	0	0	1	0

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	0	1	0	1	0

- Out input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	1	1	0	1	0

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	1	1	1	1	0

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	2	1	1	1	0

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	0

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	1

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---




- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	1

ItemID
Frequency

- Our input

4	2	1	3	1	4	5
---	---	---	---	---	---	---



- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency

- So how do we sort it now then?

- Our input

--	--	--	--	--	--	--

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency


- So how do we sort it now then?

- Our input

--	--	--	--	--	--	--

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency




- So how do we sort it now then?

- Our input



- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency




- So how do we sort it now then?

- Our input

1	1					
---	---	--	--	--	--	--

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency




- So how do we sort it now then?

- Our input

1	1	2				
---	---	---	--	--	--	--

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency



- So how do we sort it now then?


- Our input

1	1	2	3			
---	---	---	---	--	--	--

- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	1

ItemID
Frequency



- So how do we sort it now then?


- Our input

1	1	2	3	4	4	
---	---	---	---	---	---	--

- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	1

ItemID
Frequency




- So how do we sort it now then?

- Our input

1	1	2	3	4	4	5
---	---	---	---	---	---	---

- We know max is 5

0	1	2	3	4	5	ItemID
0	2	1	1	2	1	Frequency



- So how do we sort it now then?

- Our input

1	1	2	3	4	4	5
---	---	---	---	---	---	---

- We know max is 5

0	1	2	3	4	5
0	2	1	1	2	1



ItemID
Frequency

- So how do we sort it now then?



Counting Sort

Complexity

- Time?

Counting Sort

Complexity

- Time?
 - Find the maximum $O(N)$

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max

Counting Sort

Complexity

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array

Counting Sort

Complexity

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array
 - How to make it fast?

Counting Sort

Complexity

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array
 - How to make it fast?

0	1	2	3	4	5
0	2	1	1	2	1

Index
Frequency



- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array
 - How to make it fast?
 - Therefore this is $O(N)$ since we can have $O(1)$ access to the count-array

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array
 - How to make it fast?
 - Therefore this is $O(N)$ since we can have $O(1)$ access to the count-array
 - Loop through count-array to rebuild the original list $O(M+N)$

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array
 - How to make it fast?
 - Therefore this is $O(N)$ since we can have $O(1)$ access to the count-array
 - Loop through count-array to rebuild the original list $O(M)$
 - Total = $O(N + M + N + M + N) = O(N+M)$

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array
 - How to make it fast?
 - Therefore this is $O(N)$ since we can have $O(1)$ access to the count-array
 - Loop through count-array to rebuild the original list $O(M)$
 - Total = $O(N + M + N + M + N) = O(N+M)$
 - So we want $M \ll N$ for this to be good
 - Else even $N \log N < M$

- Time?
 - Find the maximum $O(N)$
 - Build the count-array $O(M)$ where M is the max
 - Go through input list and update the count-array
 - How to make it fast?
 - Therefore this is $O(N)$ since we can have $O(1)$ access to the count-array
 - Loop through count-array to rebuild the original list $O(M)$
 - Total = $O(N + M + N + M + N) = O(N+M)$
 - So we want $M \ll N$ for this to be good
 - If we are doing alphabets only, then the $M = 26$ for the 26 character (after ascii conversion + maths)

Questions?

Counting Sort

Complexity

- Space?

Counting Sort

Complexity

- Space?
 - Input list $O(N)$
 - Count-array $O(M)$

- Space?
 - Input list $O(N)$
 - Count-array $O(M)$

 - Total = $O(N + M)$
 - Auxiliary = $O(M)$

Questions?

- Live programming session
- Let us try to code this since it is simple...

- Live programming session
- Let us try to code this since it is simple...

- I will start writing the first part
 - You try to add in your own codes and compare at each step

Questions?

Counting Sort

Issue...

- Now imagine the following:

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

Counting Sort

Issue...

- Now imagine the following:

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What is my complexity?

Counting Sort

Issue...

- Now imagine the following:

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What is my complexity?
 - Time...
 - Space...

Counting Sort

Issue...

- Now imagine the following:

200	456	291	981	369	421	671
	271					

- What is my complexity?
 - Time...
 - Space...
- What if one of the value is **LARGE**

Counting Sort

Issue...

- Now imagine the following:

200	456	291	981	369	421	671
	271					

- What is my complexity?
 - Time...
 - Space...
 - What if one of the value is **LARGE**
- M is large!!!**

Counting Sort

Issue...

- Now imagine the following:

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What is my complexity?
 - Time...
 - Space...
- Let us leave it at it is first...

Questions?

- Stable?

- Stable?
 - No
 - We only remember the frequency

- Stable?
 - No
 - We only remember the frequency

- But can we make it stable?

- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory

Counting Sort

4a	2	1a	3	1b	4b	5
----	---	----	---	----	----	---

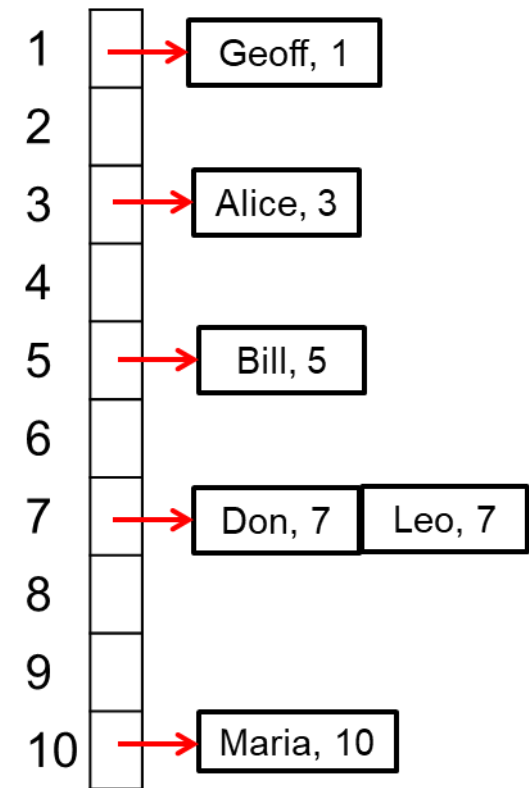
0	1	2	3	4	5
	1a	2	3	4a	5
	1b			4b	

Index
Frequency

- Stable?
 - No
 - We only remember the frequency

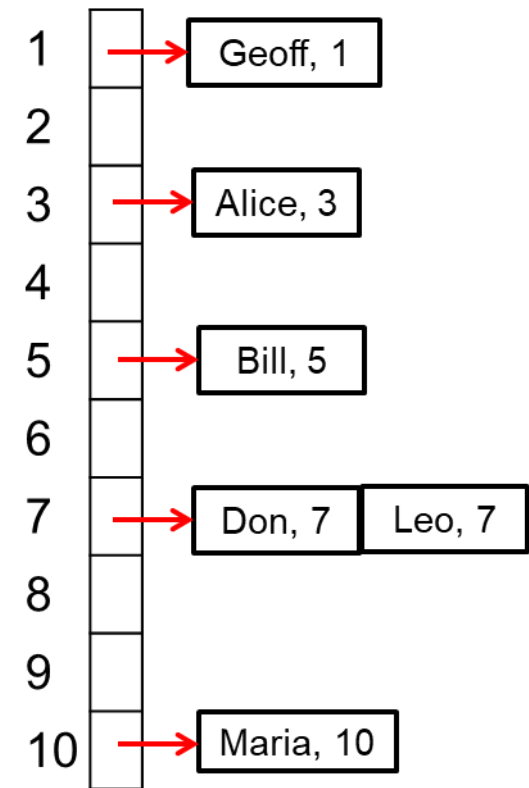
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining

- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining



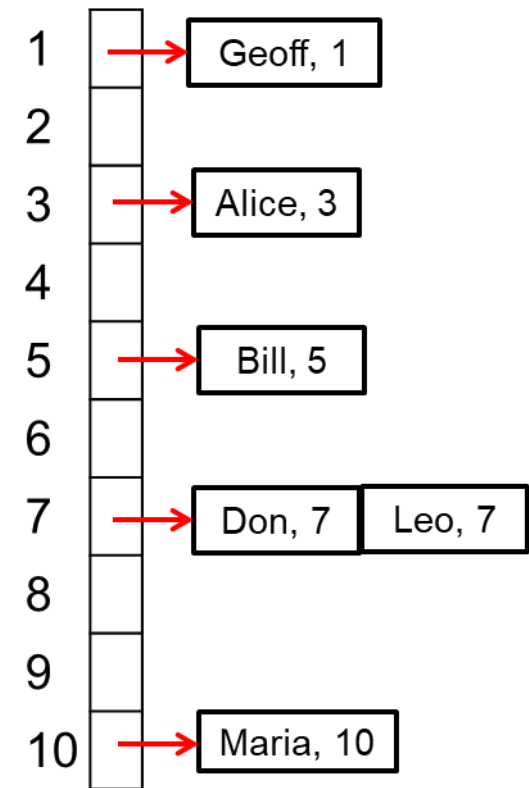
Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - At most we have N items only anyways
 - So it is $O(M + N)$ space still



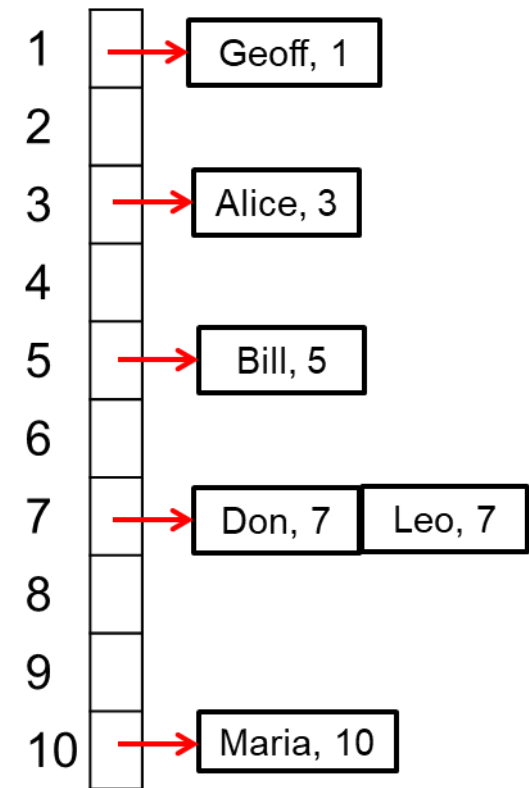
Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - At most we have N items only anyways
 - So it is $O(M + N)$ space still
 - Can you see why?



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

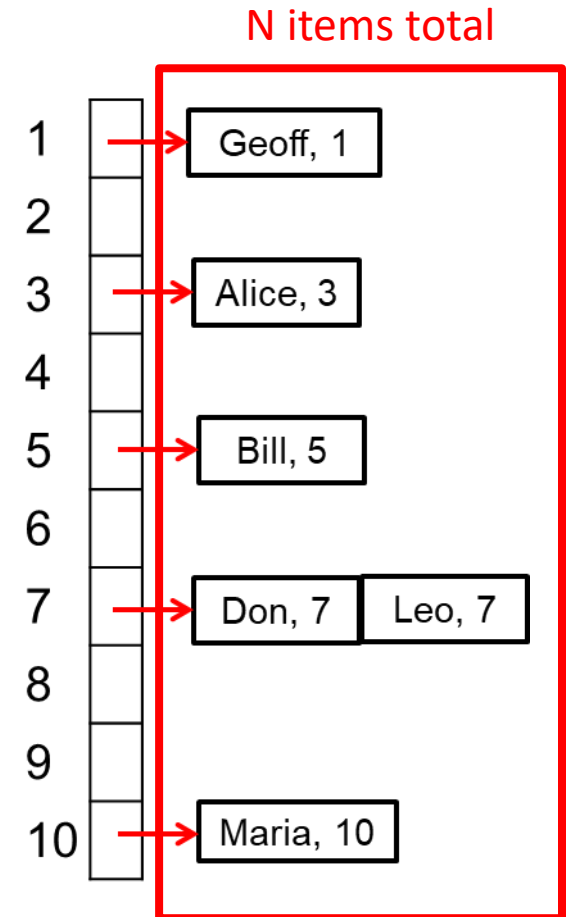
- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - At most we have N items only anyways
 - So it is $O(M + N)$ space still
 - Can you see why?



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

N items

- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - At most we have N items only anyways
 - So it is $O(M + N)$ space still
 - Can you see why?



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

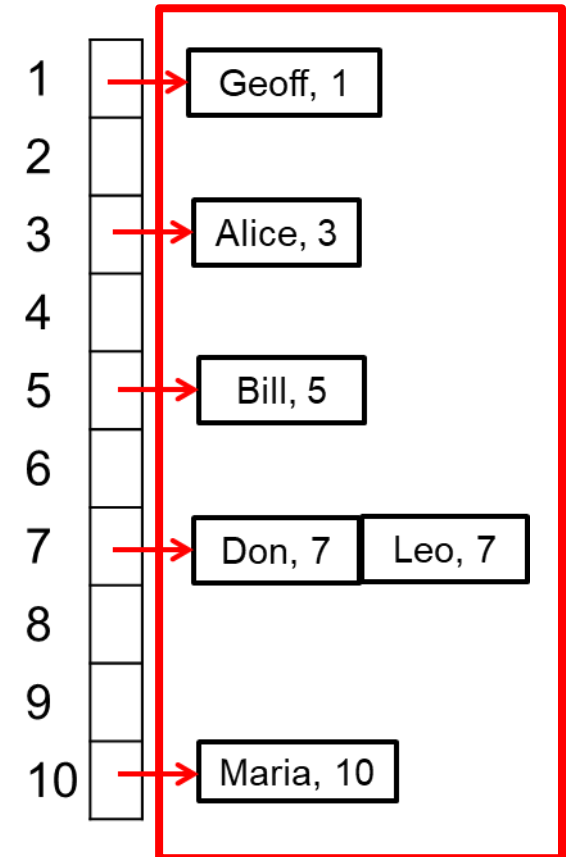
N items

Counting Sort

Not $O(N \cdot M)$

N items total

- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - At most we have N items only anyways
 - So it is $O(M + N)$ space still
 - Can you see why?



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

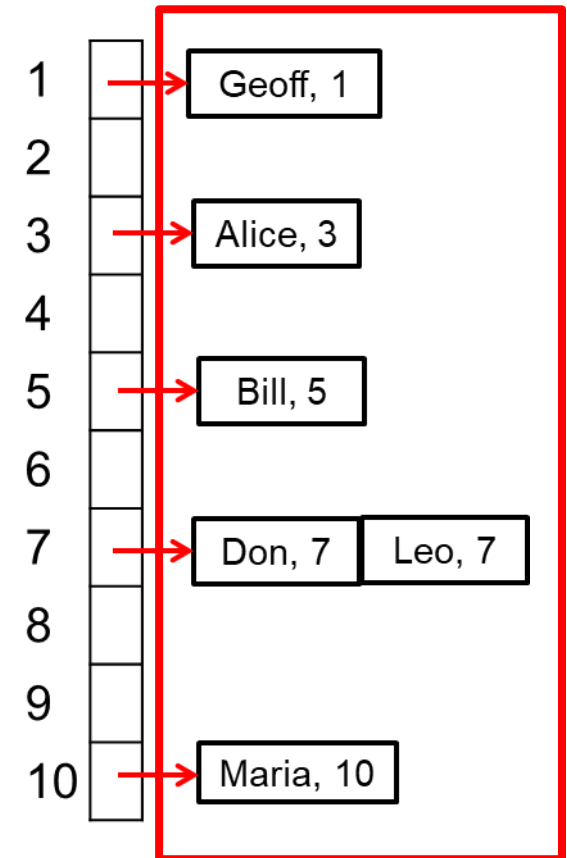
N items

Counting Sort

Not $O(N \cdot M)$

N items total

- Stable?
 - No
 - We only remember the frequency
- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - At most we have N items only anyways
 - So it is $O(M + N)$ space still
 - Can you see why?



Marks	3	5	7	1	7	10
Name	Alice	Bill	Don	Geoff	Leo	Maria

N items

Questions?

- Stable?
 - No
 - We only remember the frequency

- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - There is another way, refer to **Nathan's** amazing slide

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

count

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1

count position

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

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

count position

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

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

count position

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

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

count position

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

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

count position

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

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

count position

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

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

count position

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

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

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

count position

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

1	1
2	2
3	2
4	5
5	5
6	6
7	6
8	8

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count position

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

1	1
2	2
3	2
4	5
5	5
6	6
7	6
8	8

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count **position**

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

1	1
2	2
3	2
4	5
5	5
6	6
7	6
8	8

Output

1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count position

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

1	1
2	2
3	3
4	5
5	5
6	6
7	6
8	8

Output

	(3,a)						
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count position

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

1	2
2	2
3	3
4	5
5	5
6	6
7	6
8	8

Output

(1,p)	(3,a)						
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count **position**

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

1	2
2	2
3	4
4	5
5	5
6	6
7	6
8	8

Output

(1,p)	(3,a)	(3,c)					
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count position

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

1	2
2	2
3	4
4	5
5	5
6	7
7	6
8	8

Output

(1,p)	(3,a)	(3,c)			(7,f)		
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count position

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

1	2
2	2
3	4
4	5
5	6
6	7
7	6
8	8

Output

(1,p)	(3,a)	(3,c)		(5,g)	(7,f)		
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count position

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

1	2
2	2
3	5
4	5
5	6
6	7
7	6
8	8

Output

(1,p)	(3,a)	(3,c)	(3,b)	(5,g)	(7,f)		
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count position

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

1	2
2	2
3	5
4	5
5	6
6	7
7	7
8	8

Output

(1,p)	(3,a)	(3,c)	(3,b)	(5,g)	(7,f)	(7,d)	
1	2	3	4	5	6	7	8

Stable Counting Sort (Method 1)

Input

(3,a)	(1,p)	(3,c)	(7,f)	(5,g)	(3,b)	(7,d)	(8,w)
-------	-------	-------	-------	-------	-------	-------	-------

Construct count:

- For each key in input,
- $\text{count}[\text{key}] += 1$

Construct position:

- Initialise first position as a 1
- $\text{position}[i] = \text{position}[i-1] + \text{count}[i-1]$

Construct output

- Go through input, looking at each (key, val)
- Set $\text{output}[\text{position}[\text{key}]]$ to the (key, val) pair from input
- Increment $\text{position}[\text{key}]$

count **position**

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

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

Output

(1,p)	(3,a)	(3,c)	(3,b)	(5,g)	(7,f)	(7,d)	(8,w)
1	2	3	4	5	6	7	8

Questions?

- Stable?
 - No
 - We only remember the frequency

- But can we make it stable?
 - Yes but at the cost of memory
 - Similar to separate chaining
 - There is another way, refer to **Nathan's** amazing slide
 - Are the complexity the same?

Questions?

Have a break again!

Counting Sort

Remember this issue...

- Now imagine the following:

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What is my complexity?
 - Time...
 - Space...
- Let us leave it at it is first...

Counting Sort

Remember this issue...

- Now imagine the following:

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What is my complexity?
 - Time...
 - Space...
- Let us leave it at it is first... We shall resolve this now...

Counting Sort

Remember this issue...

- Now imagine the following:

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What is my complexity?
 - Time...
 - Space...
- Let us leave it at it is first... We shall resolve this now...

Questions?

Radix Sort

A different outlook...

- With this input...

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

Radix Sort

A different outlook...

- With this input...

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What if we view it differently?

Radix Sort

A different outlook...

- With this input...

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What if we view it differently?

200
151
291
981
369
421

Radix Sort

A different outlook...

- With this input...

200	151	291	981	369	421	671
-----	-----	-----	-----	-----	-----	-----

- What if we view it differently? How would we sort it?

200
151
291
981
369
421

Radix Sort

A different outlook...

- With this input...
 - What if we view it differently? How would we sort it?

200
151
291
981
369
421
671

Radix Sort

A different outlook...

- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant

200
151
291
981
369
421
671

Radix Sort

A different outlook...

- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



200
151
291
981
369
421
671

Radix Sort

A different outlook...

- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



200
151
291
981
369
421
671

Radix Sort

A different outlook...

- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant

200
151
291
981
369
421
671

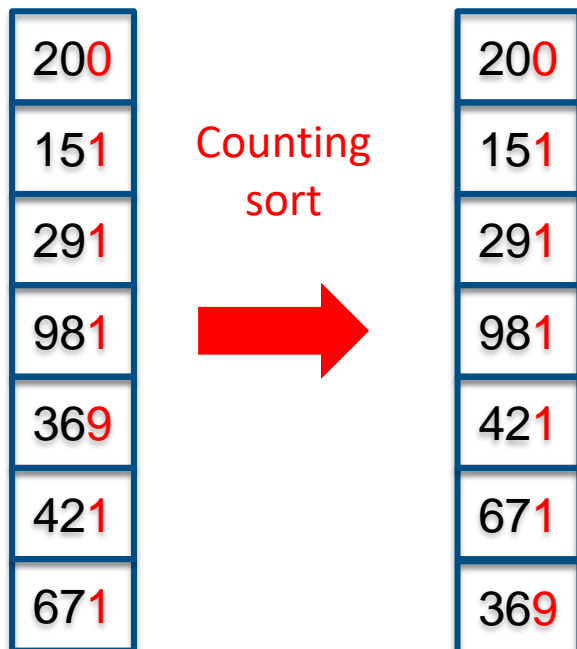
Counting
sort



Radix Sort

A different outlook...

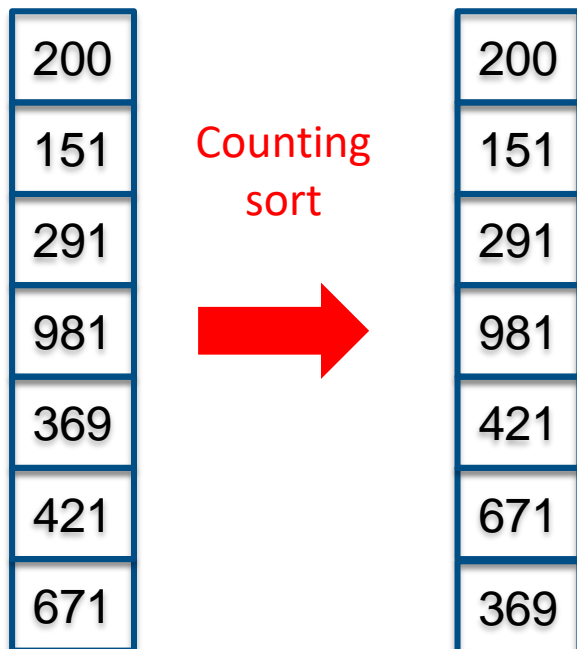
- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



Radix Sort

A different outlook...

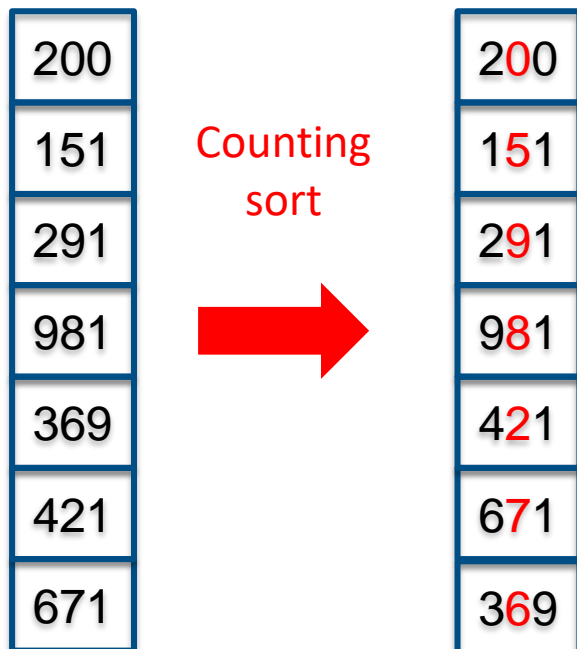
- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



Radix Sort

A different outlook...

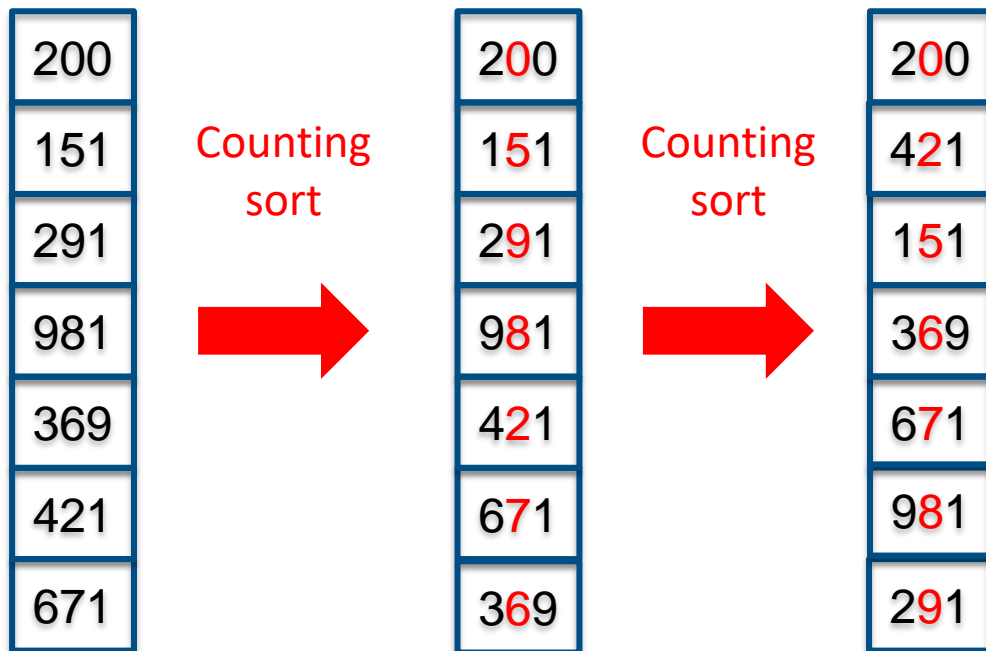
- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



Radix Sort

A different outlook...

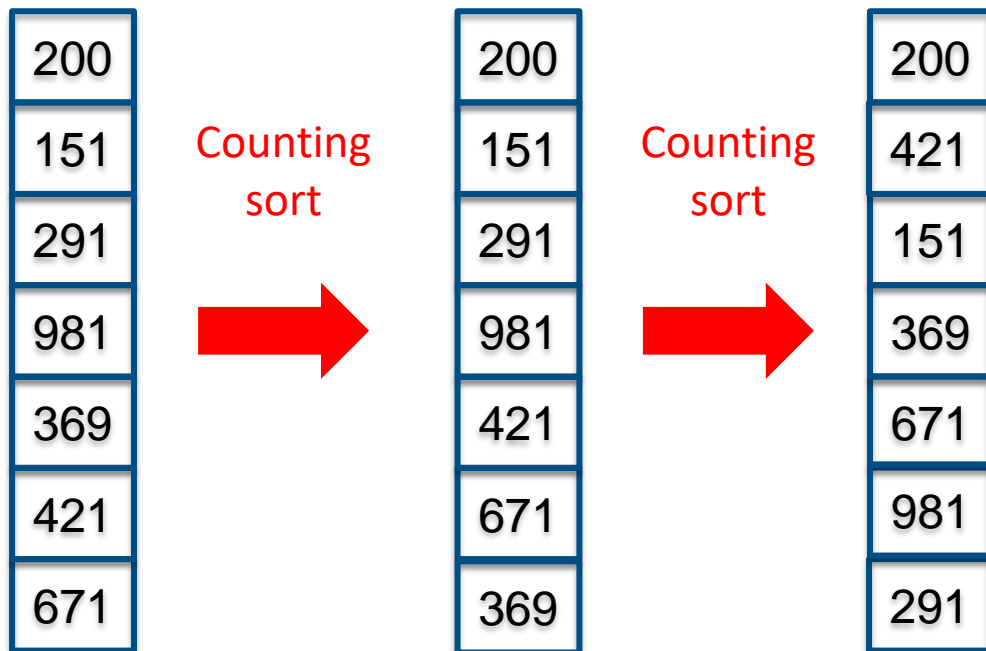
- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



Radix Sort

A different outlook...

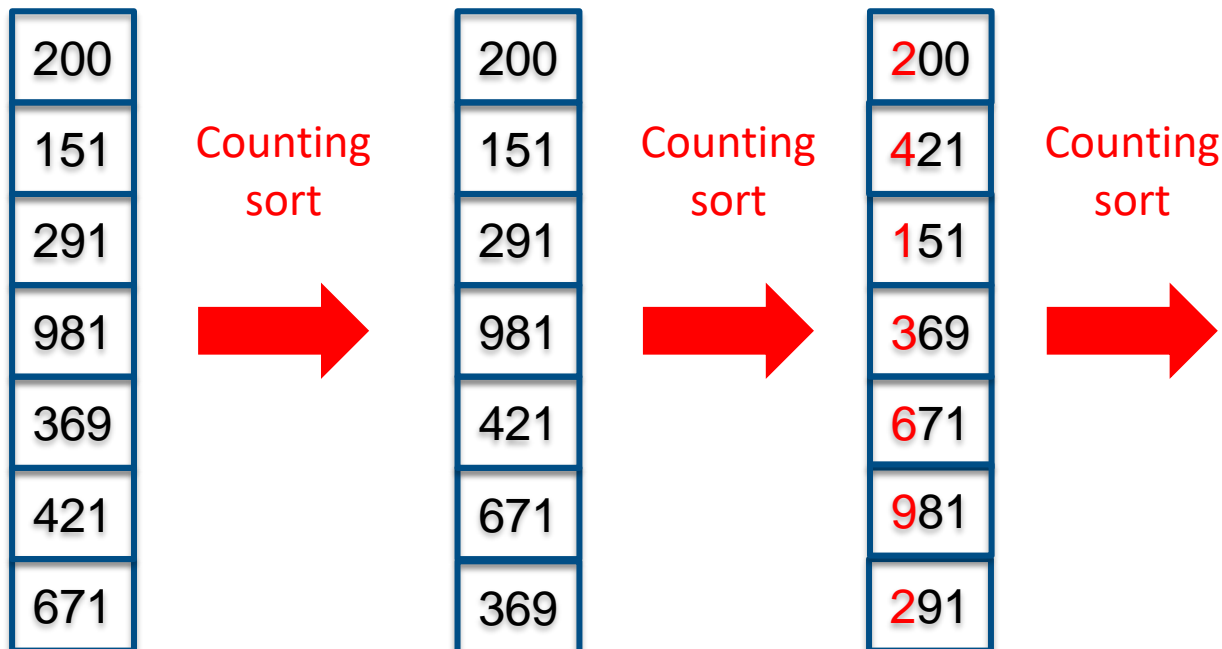
- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



Radix Sort

A different outlook...

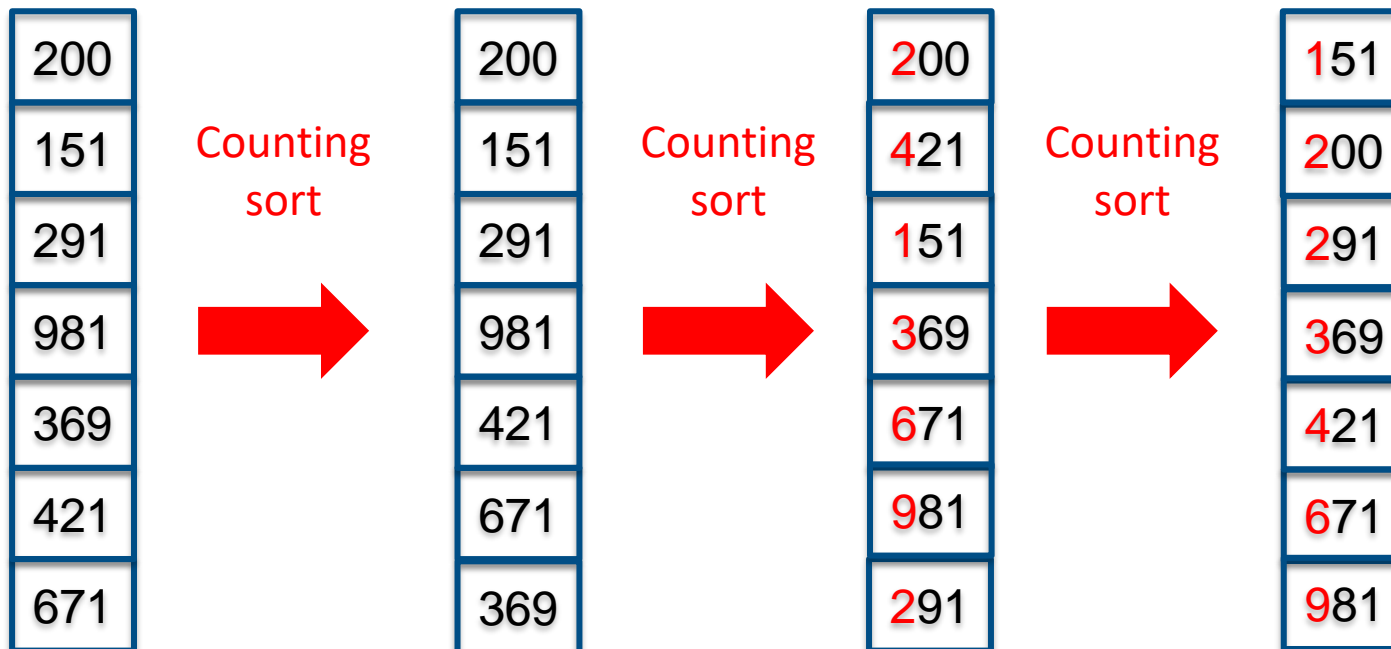
- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant



Radix Sort

A different outlook...

- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant

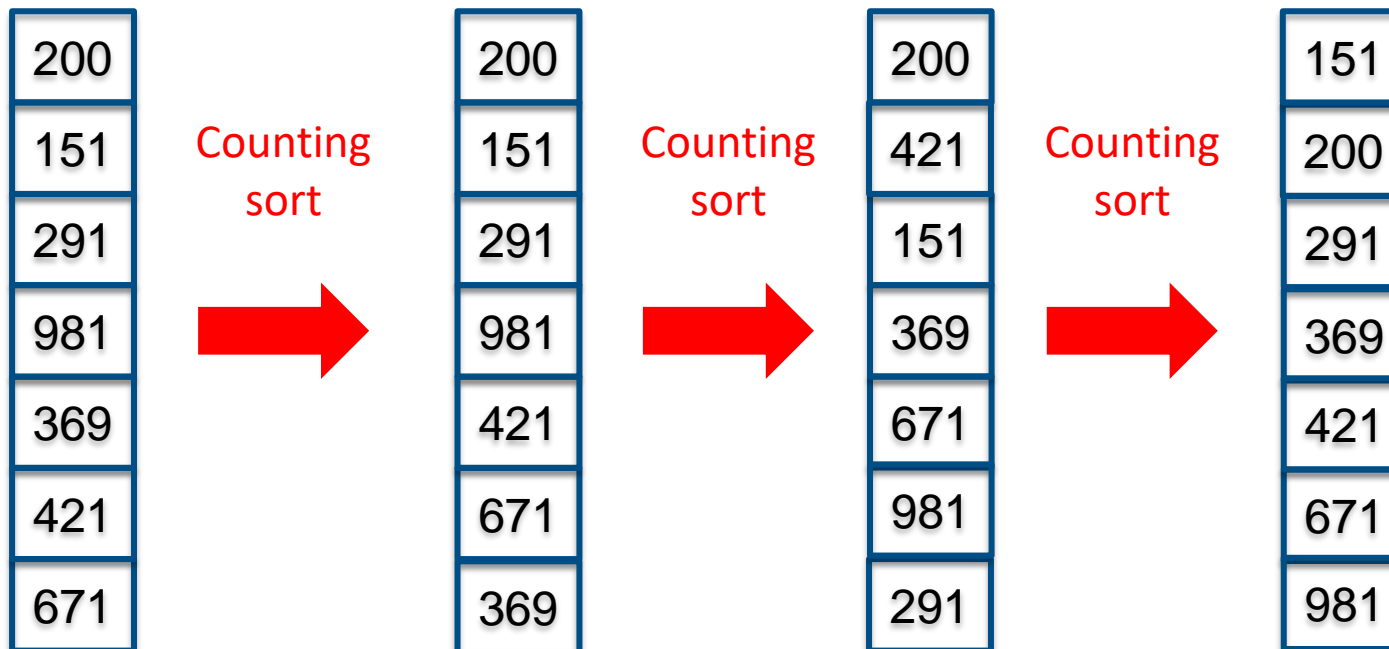


Radix Sort

A different outlook...



- With this input...
 - What if we view it differently? How would we sort it?
 - Right most digit = least significant
 - Left most digit = most significant

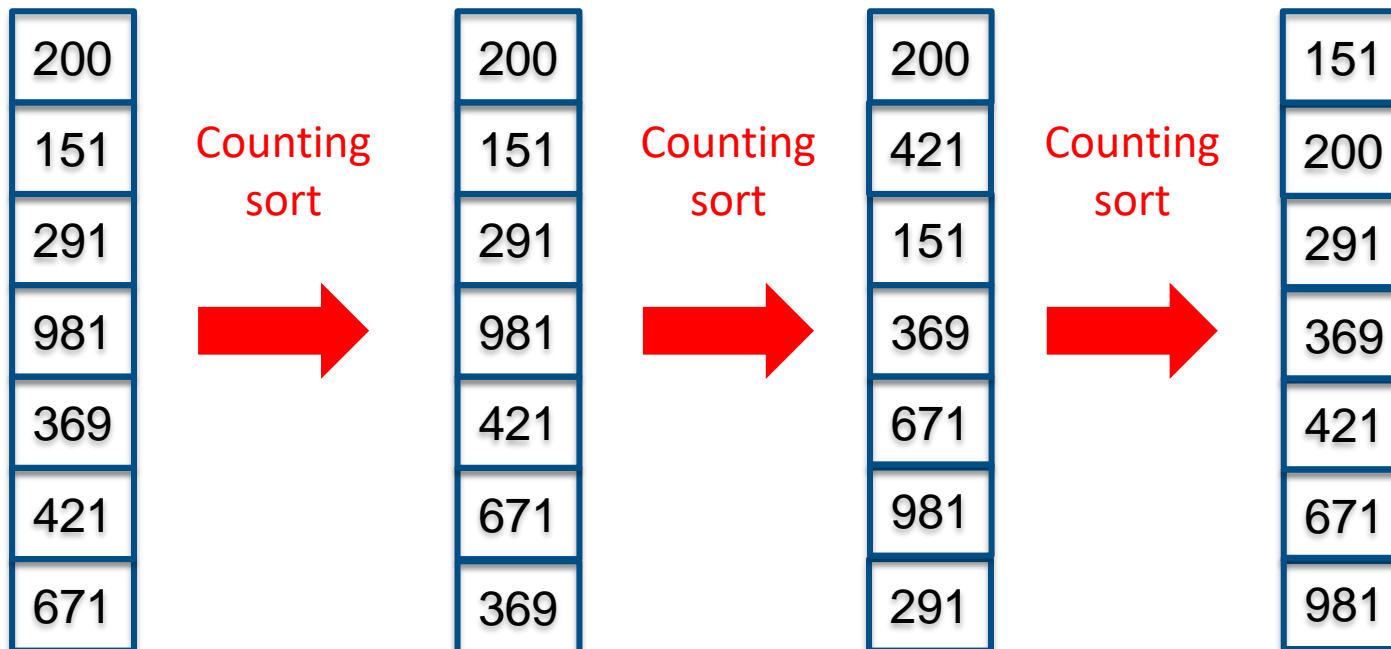


Questions?

Radix Sort

A different outlook...

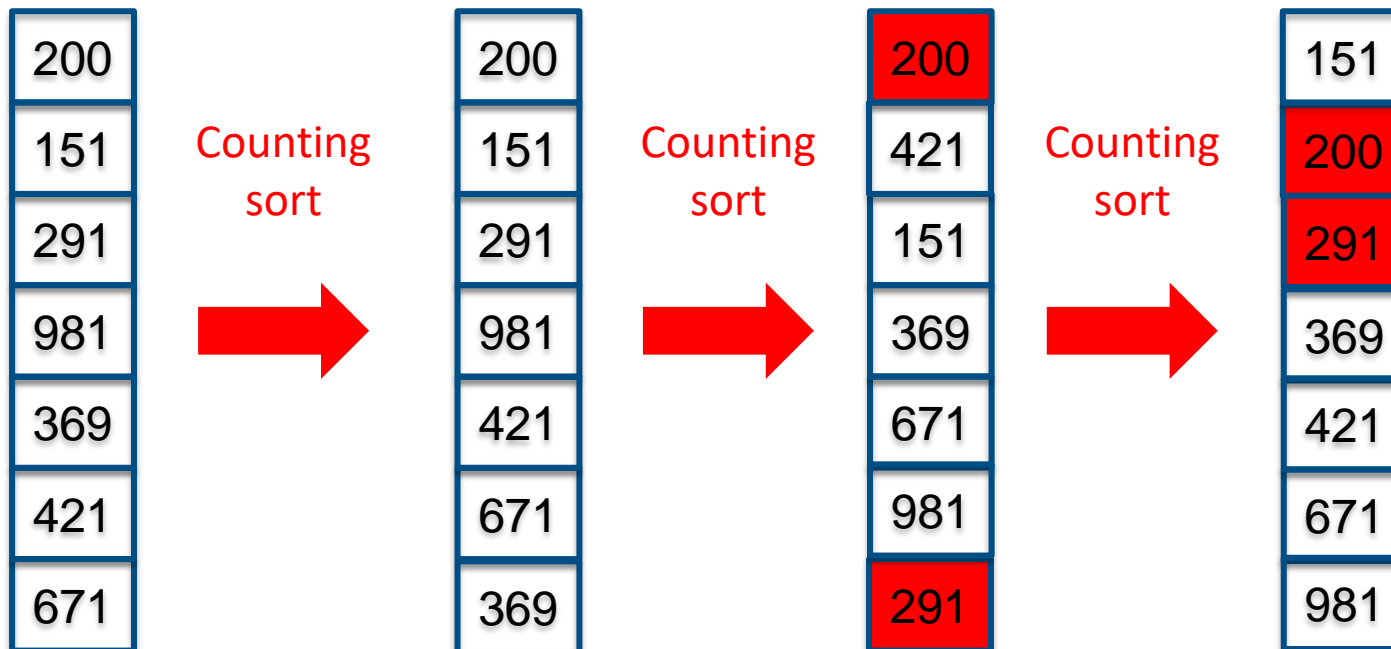
- With this input...
 - What if we view it differently? How would we sort it?
 - But the sorting need to be **stable**



Radix Sort

A different outlook...

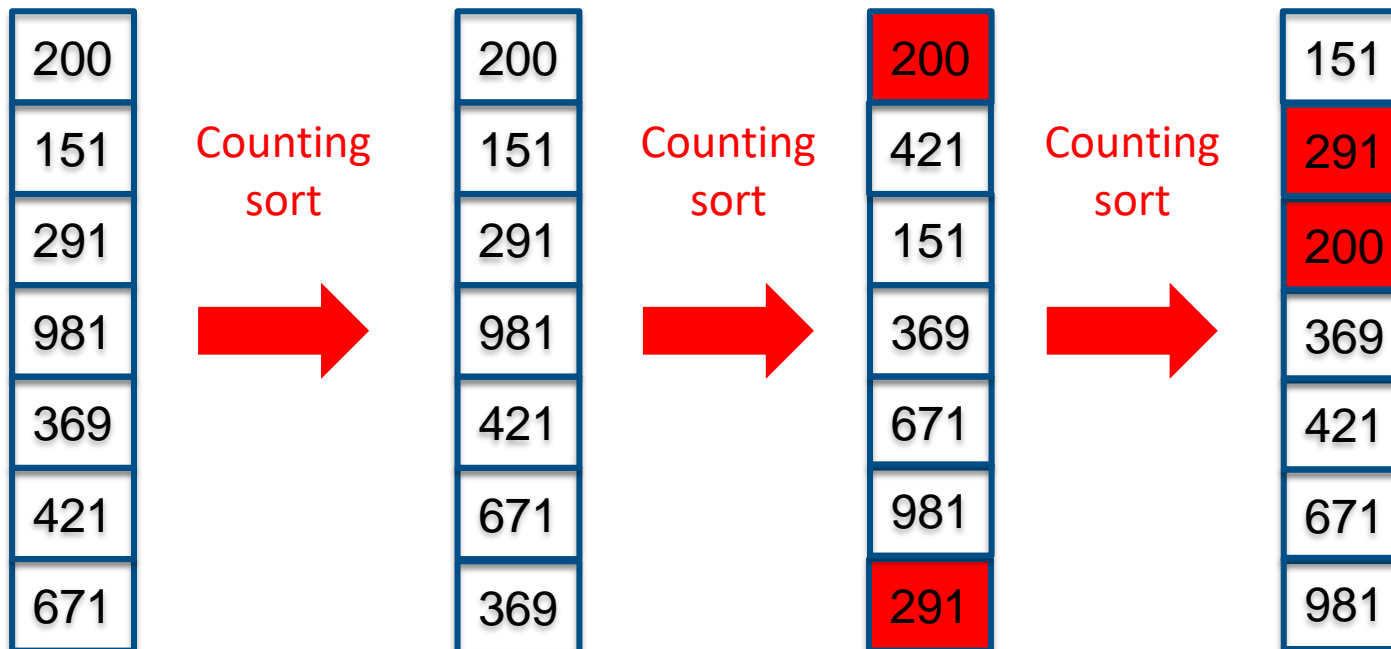
- With this input...
 - What if we view it differently? How would we sort it?
 - But the sorting need to be **stable**



Radix Sort

A different outlook...

- With this input...
 - What if we view it differently? How would we sort it?
 - But the sorting need to be **stable**, if not...



**It's a
disastah!**

Questions?

Radix Sort

Complexity

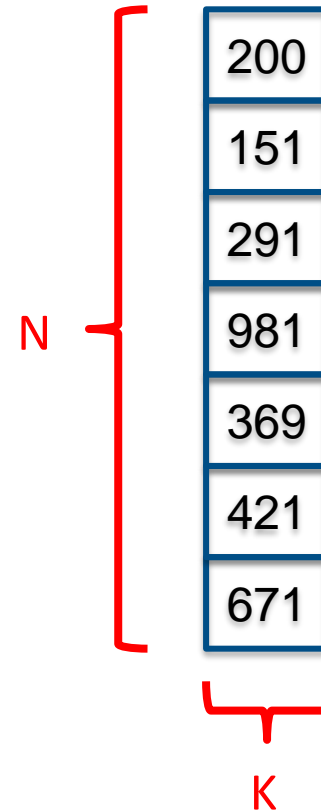
- What is the complexity?
 - Time
 - Space

200
151
291
981
369
421
671

Radix Sort

Complexity

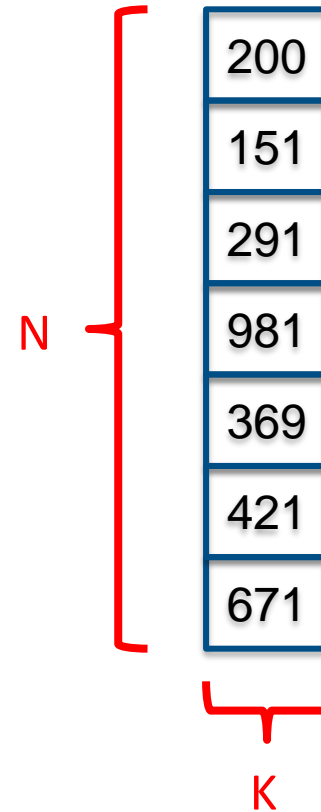
- What is the complexity?
 - Time
 - Space



Radix Sort

Complexity

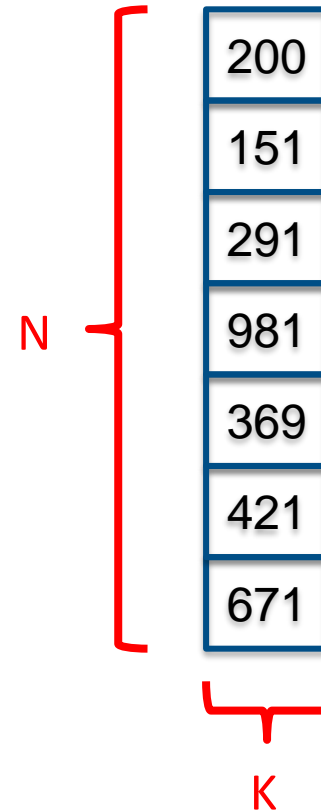
- What is the complexity?
 - Time
 - $O(KN)$?
 - Space



Radix Sort

Complexity

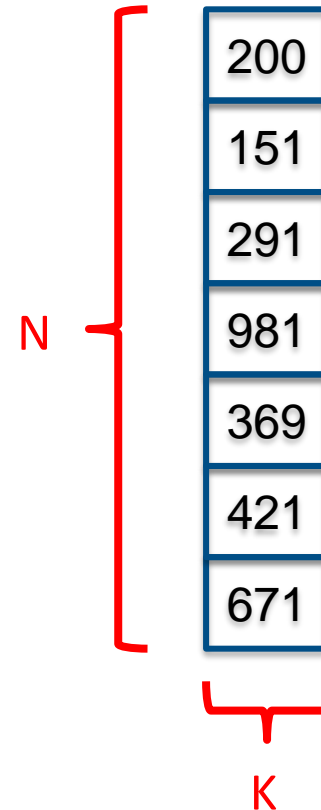
- What is the complexity?
 - Time
 - $O(KN) + O(KM)$
where M is the number of unique characters
 - Space



Radix Sort

Complexity

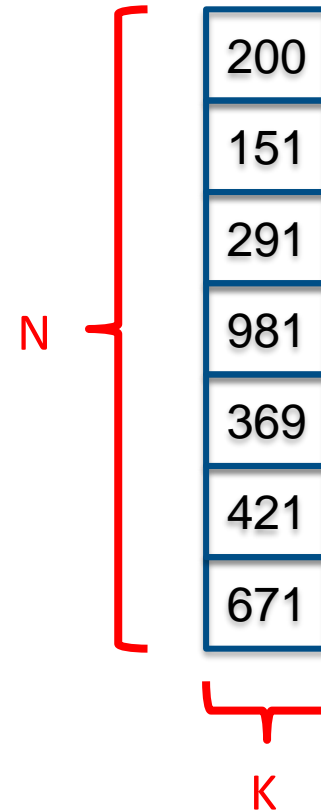
- What is the complexity?
 - Time
 - $O(KN) + O(KM)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max
 - Space



Radix Sort

Complexity

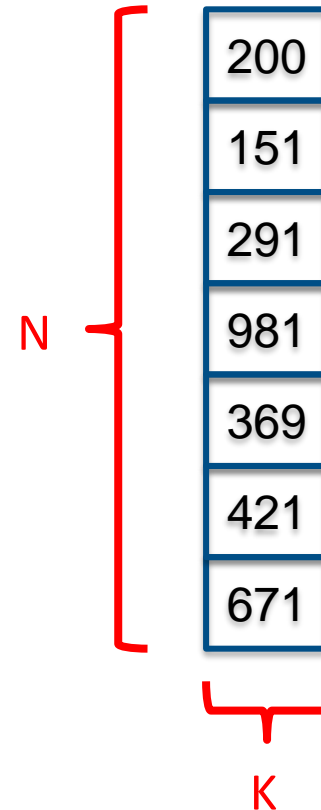
- What is the complexity?
 - Time
 - $O(KN) + O(KM)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Space



Radix Sort

Complexity

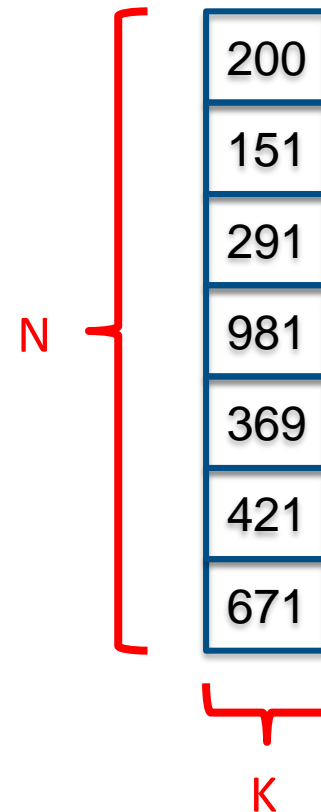
- What is the complexity?
 - Time
 - $O(KN) + O(KM)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns
 - Space



Radix Sort

Complexity

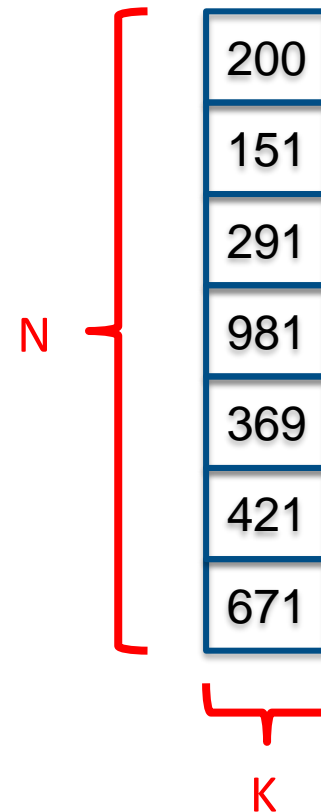
- What is the complexity?
 - Time
 - $O(KN) + O(KM)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns giving us $O(K) * O(N+M)$
 - Space



Radix Sort

Complexity

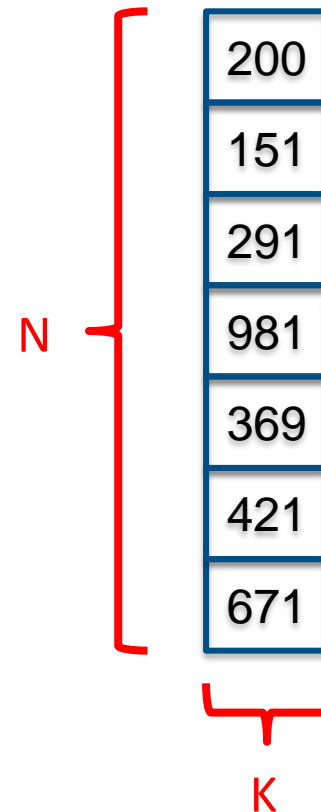
- What is the complexity?
 - Time
 - $O(KN + KM)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns giving us $O(K) * O(N+M)$
 - Space



Radix Sort

Complexity

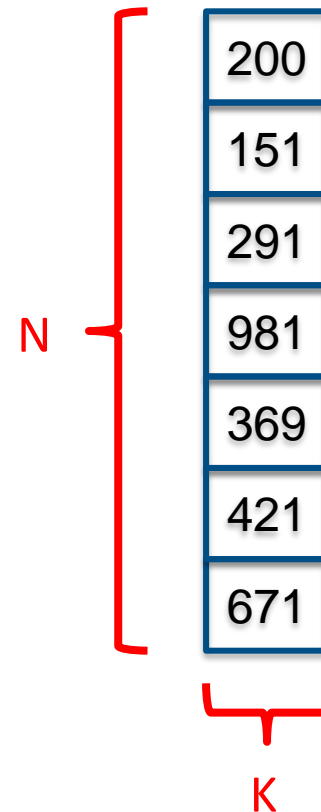
- What is the complexity?
 - Time
 - $O(KN + KM)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns giving us $O(K) * O(N+M)$
 - Space
 - Input is $O(KN)$
 - Each counting sort needs $O(M+N)$



Radix Sort

Complexity

- What is the complexity?
 - Time
 - $O(KN + KM)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns giving us $O(K) * O(N+M)$
 - Space
 - Input is $O(KN)$
 - Each counting sort needs $O(M+N)$
 - Total is $O(KN + M + N)$



Radix Sort

Complexity

- What is the complexity?

- But we know $M = 10$ for 0, 1, ..., 9

- Time

- $O(KN + KM)$

- where M is the number of unique characters

- Why? Recall counting sort, we account for the max giving us $O(N+M)$

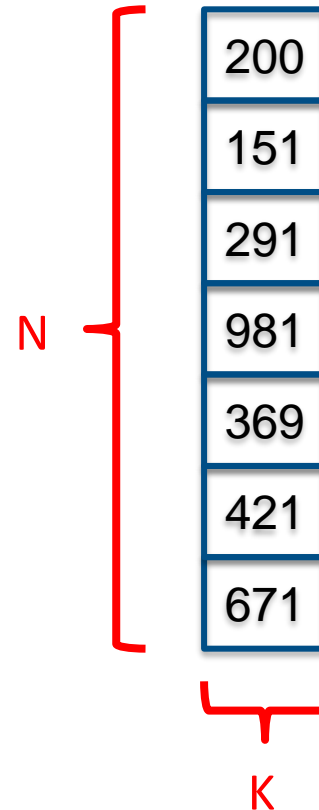
- Then we have K columns giving us $O(K) * O(N+M)$

- Space

- Input is $O(KN)$

- Each counting sort needs $O(M+N)$

- Total is $O(KN + M + N)$



Radix Sort

Complexity

- What is the complexity?

- But we know $M = 10$ for 0, 1, ..., 9

- Time

- $O(KN + KM) \approx O(KN)$

- where M is the number of unique characters

- Why? Recall counting sort, we account for the max giving us $O(N+M)$

- Then we have K columns giving us $O(K) * O(N+M)$

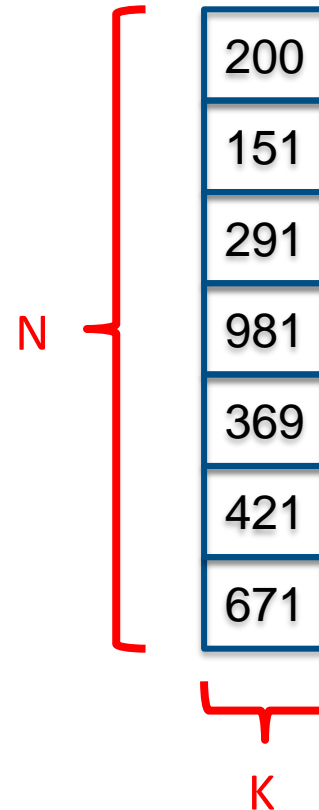
- Space

- Input is $O(KN)$

- Each counting sort needs $O(M+N)$

- Total is $O(KN + M + N) \approx O(KN)$

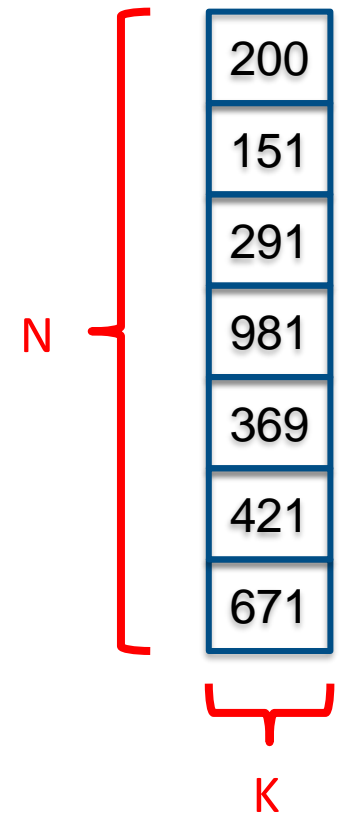
- Auxiliary is $O(M + N) \approx O(N)$



Radix Sort

Complexity

- What is the complexity?
 - Better than merge sort $O(k N \log N)$!
 - But we know $M = 10$ for 0, 1, ..., 9
 - Time
 - $O(KN + KM) \approx O(KN)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns giving us $O(K) * O(N+M)$
 - Space
 - Input is $O(KN)$
 - Each counting sort needs $O(M+N)$
 - Total is $O(KN + M + N) \approx O(KN)$
 - Auxiliary is $O(M + N) \approx O(N)$



Radix Sort

Complexity

■ What is the complexity?

– Better than merge sort $O(k N \log N)$!

– But we know $M = 10$ for 0, 1, ..., 9

– Time

■ $O(KN + KM) \approx O(KN)$

where M is the number of unique characters

■ Why? Recall counting sort, we account for the max giving us $O(N+M)$

■ Then we have K columns giving us $O(K) * O(N+M)$

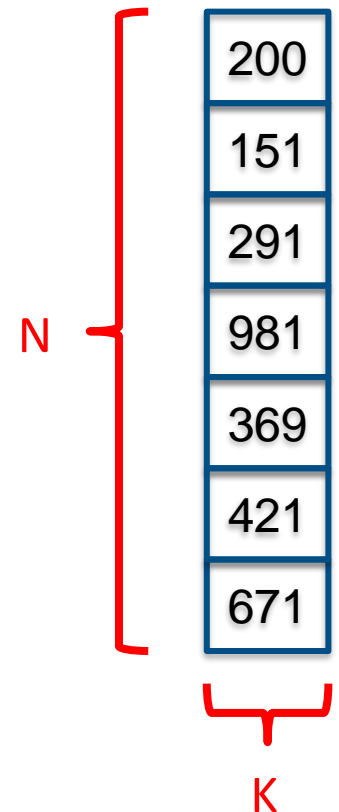
– Space

■ Input is $O(KN)$

■ Each counting sort needs $O(M+N)$

■ Total is $O(KN + M + N) \approx O(KN)$

■ Auxiliary is $O(M + N) \approx O(N)$ <- why no K ? Come ask me if interested...

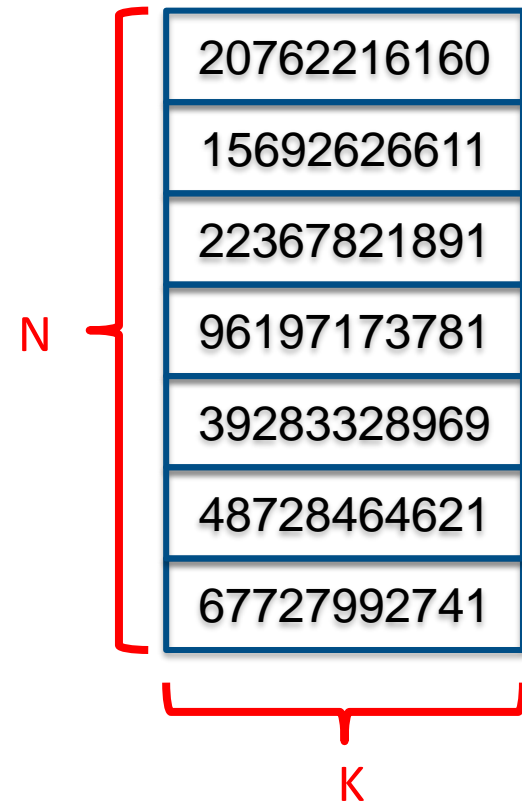


Questions?

Radix Sort

Complexity

- What is the complexity?
 - What if k is bigger?
 - But we know $M = 10$ for 0, 1, ..., 9
 - Time
 - $O(KN + KM) \approx O(KN)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns giving us $O(K) * O(N+M)$
 - Space
 - Input is $O(KN)$
 - Each counting sort needs $O(M+N)$
 - Total is $O(KN + M + N) \approx O(KN)$
 - Auxiliary is $O(M + N) \approx O(N)$



Radix Sort

Complexity

- What is the complexity?

- What if k is bigger?

- But we know $M = 10$ for 0, 1, ..., 9

- Time

- $O(KN + KM) \approx O(KN)$

- where M is the number of unique characters

- Why? Recall counting sort, we account for the max giving us $O(N+M)$

- Then we have K columns giving us $O(K) * O(N+M)$

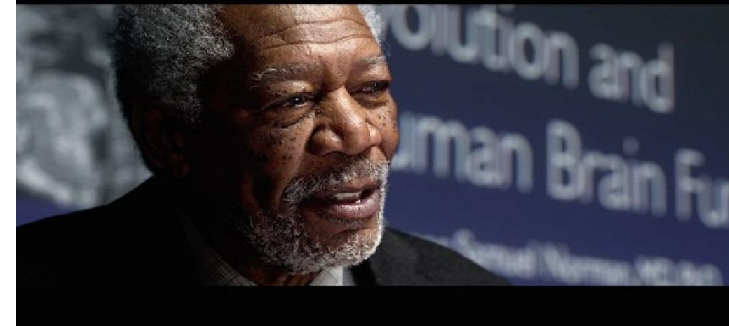
- Space

- Input is $O(KN)$

- Each counting sort needs $O(M+N)$

- Total is $O(KN + M + N) \approx O(KN)$

- Auxiliary is $O(M + N) \approx O(N)$

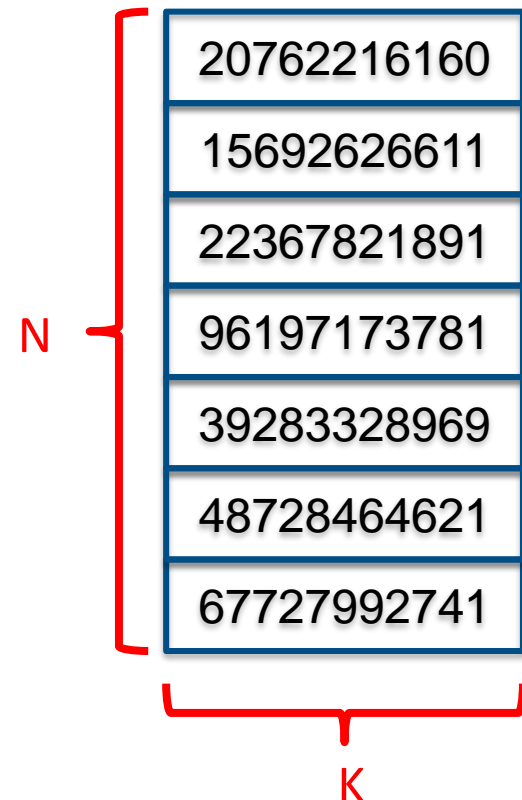


N	96197173781
	39283328969
	48728464621
	67727992741
	K

Radix Sort

Complexity

- What is the complexity?
 - What if k is bigger?
 - We increase $M = 100$ for 0, 1, ..., 99
 - Time
 - $O(KN + KM) \approx O(KN)$
where M is the number of unique characters
 - Why? Recall counting sort, we account for the max giving us $O(N+M)$
 - Then we have K columns giving us $O(K) * O(N+M)$
 - Space
 - Input is $O(KN)$
 - Each counting sort needs $O(M+N)$
 - Total is $O(KN + M + N) \approx O(KN)$
 - Auxiliary is $O(M + N) \approx O(N)$



Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$

Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$
- M is the base

Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$
- M is the base
 - For decimal numbers, it is 10 from 0 to 10

Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$
- M is the base
 - For decimal numbers, it is 10 from 0 to 10
 - For binary numbers,

Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$
- M is the base
 - For decimal numbers, it is 10 from 0 to 10
 - For binary numbers, it is 2 from 0 to 1

Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$
- M is the base
 - For decimal numbers, it is 10 from 0 to 10
 - For binary numbers, it is 2 from 0 to 1
 - We can increase the M , to reduce the K ?

Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$
- M is the base
 - For decimal numbers, it is 10 from 0 to 10
 - For binary numbers, it is 2 from 0 to 1
 - We can increase the M , to reduce the K ?
- If we deal with the English alphabet, this would be 26 from a to z

baihns
hnmapg
lhhang
uhnagh
banana
trolls
hahaha

Radix Sort

Complexity

- Time complexity is $O(KN + KM)$
- Space complexity is $O(KN + M + N)$
- M is the base
 - For decimal numbers, it is 10 from 0 to 10
 - For binary numbers, it is 2 from 0 to 1
 - We can increase the M , to reduce the K ?
 - If we deal with the English alphabet, this would be 26 from a to z
 - Nathan did a good analysis on it

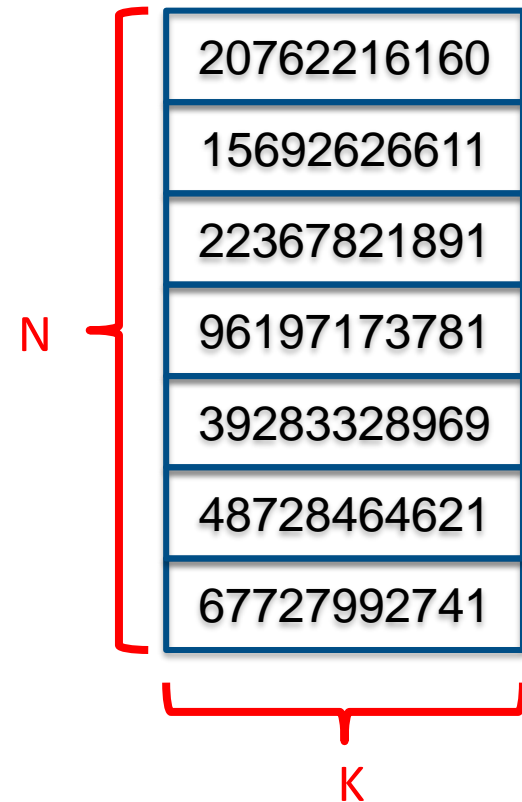
baihns
hnmapg
lhhang
uhnagh
banana
trolls
hahaha

Questions?

Radix Sort

TL;DR

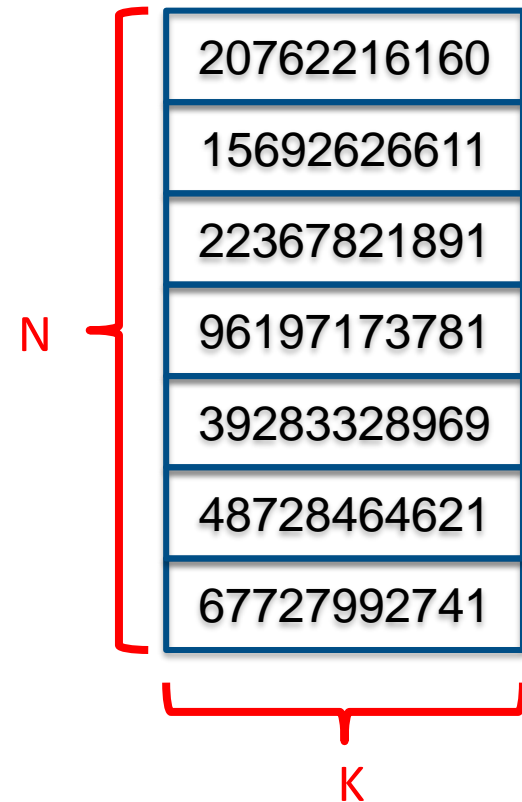
- So you know radix sort
- What have you notice?



Radix Sort

TL;DR

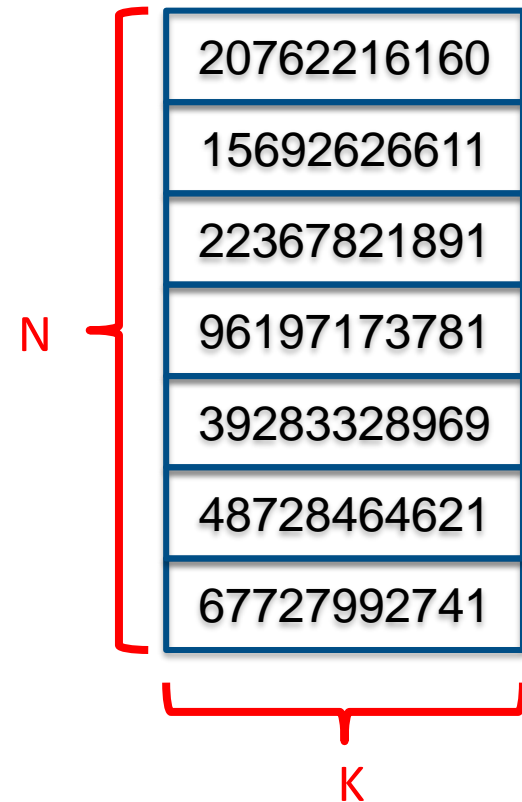
- So you know radix sort
- What have you notice?
 - It is counting sort really, done multiple times



Radix Sort

TL;DR

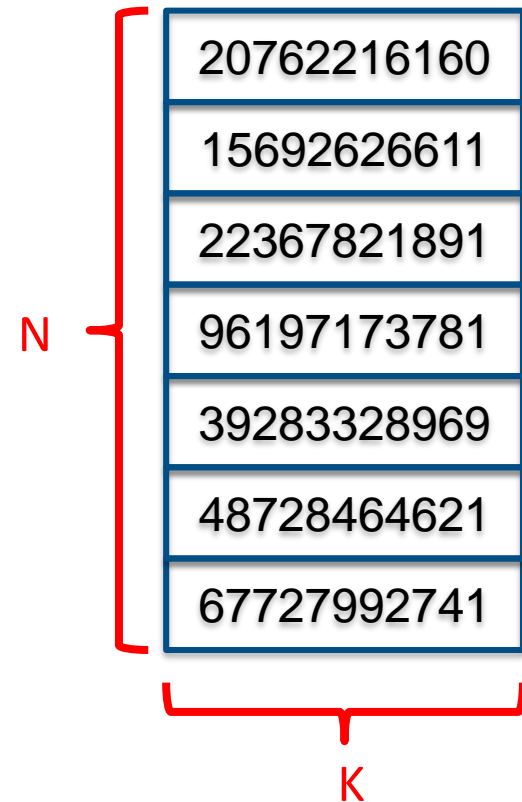
- So you know radix sort
- What have you notice?
 - It is counting sort really, done multiple times
 - Usually least significant (right) to most significant (left)



Radix Sort

TL;DR

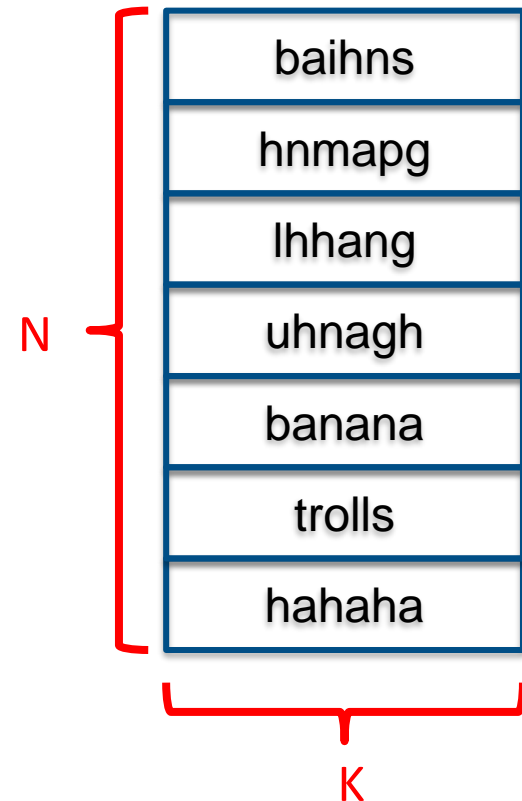
- So you know radix sort
- What have you notice?
 - It is counting sort really, done multiple times
 - We can reduce this by increasing the base
 - Usually least significant (right) to most significant (left)



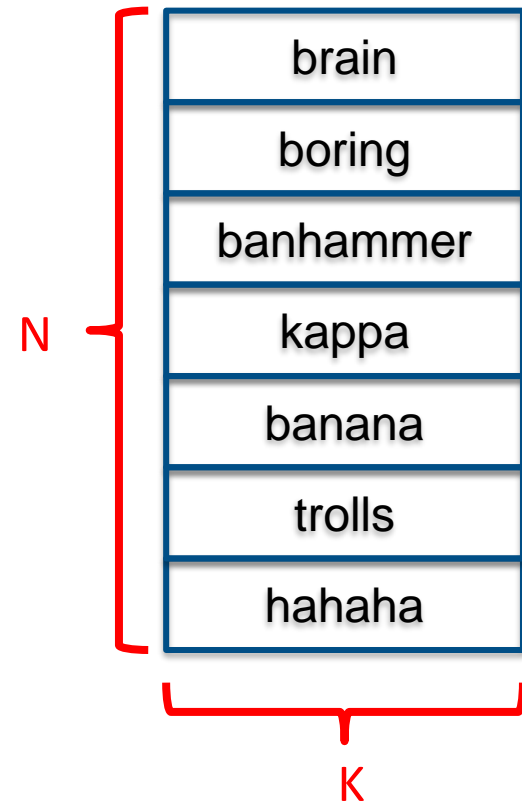
Radix Sort

TL;DR

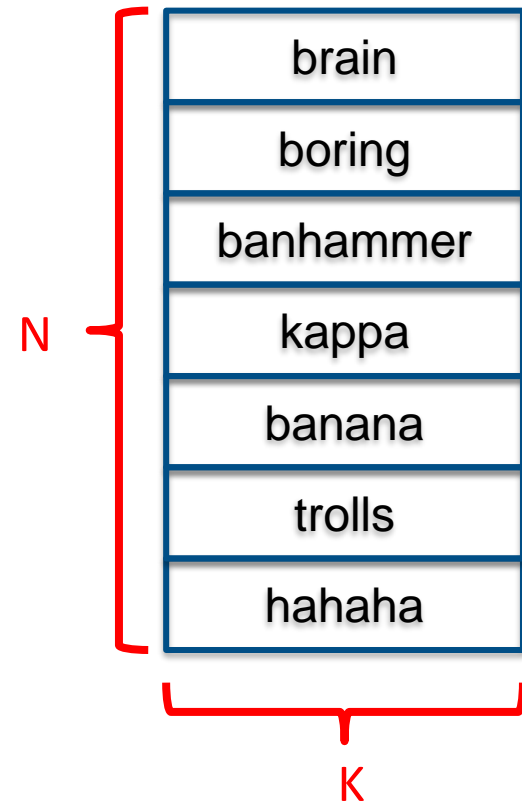
- So you know radix sort
- What have you notice?
 - It is counting sort really, done multiple times
 - We can reduce this by increasing the base
 - Works well for characters as well
 - Usually least significant (right) to most significant (left)



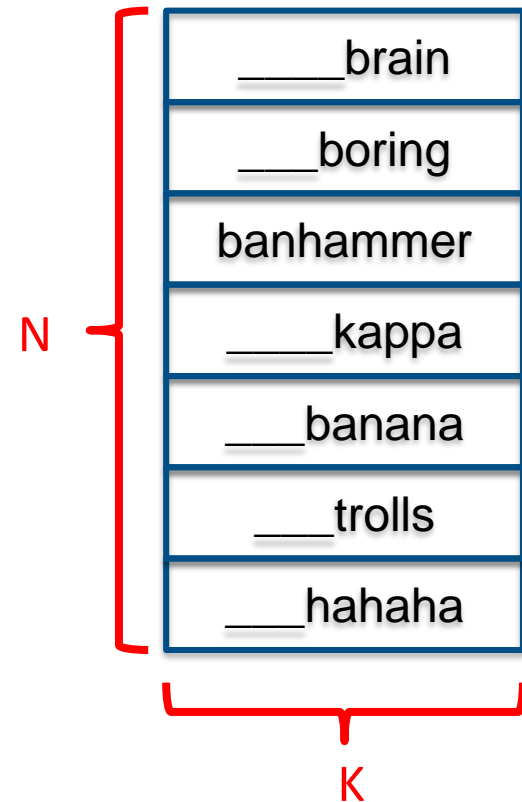
- So you know radix sort
- What have you notice?
 - It is counting sort really, done multiple times
 - We can reduce this by increasing the base
 - Works well for characters as well
 - Usually least significant (right) to most significant (left)
- But what if they are not the same length?



- So you know radix sort
- What have you notice?
 - It is counting sort really, done multiple times
 - We can reduce this by increasing the base
 - Works well for characters as well
 - Usually least significant (right) to most significant (left)
- But what if they are not the same length?
 - Left-aligned?
 - Right-aligned?



- So you know radix sort
- What have you notice?
 - It is counting sort really, done multiple times
 - We can reduce this by increasing the base
 - Works well for characters as well
 - Usually least significant (right) to most significant (left)
- But what if they are not the same length? **Add spaces!**
 - Left-aligned?
 - Right-aligned?



Questions?

Thank You