ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 3: Sorting and Search Algorithms
Lecture 8: Integer sorting

Maxim Buzdalov
Saint Petersburg 2016

How can we speed up sorting, if we know that we are sorting integers?

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays
- We can perform mathematical operations with integers

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays
- We can perform mathematical operations with integers

We are no longer limited by the $\Omega(N \log N)$ lower bound...

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays
- We can perform mathematical operations with integers

We are no longer limited by the $\Omega(N \log N)$ lower bound...

- ...because we can perform operations which reveal more information

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays
- We can perform mathematical operations with integers

We are no longer limited by the $\Omega(N \log N)$ lower bound...

- ...because we can perform operations which reveal more information

In this video:

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays
- We can perform mathematical operations with integers

We are no longer limited by the $\Omega(N \log N)$ lower bound...

- ...because we can perform operations which reveal more information

In this video:

- Counting sort

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays
- We can perform mathematical operations with integers

We are no longer limited by the $\Omega(N \log N)$ lower bound...

- ...because we can perform operations which reveal more information

In this video:

- Counting sort
- Bucket sort

How can we speed up sorting, if we know that we are sorting integers?

- Integers are typically bounded by 2^W for some word size W
- We can use integers as indices for arrays
- We can perform mathematical operations with integers

We are no longer limited by the $\Omega(N \log N)$ lower bound...

- ...because we can perform operations which reveal more information

In this video:

- Counting sort
- Bucket sort
- Radix sort

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

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

1	2	3	4	5
0	0	0	0	0

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

	1	2	5	4	3	2	1	4	3	1	5

1	2	3	4	5
0	0	0	0	1

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

		2	5	4	3	2	1	4	3	1	5

1	2	3	4	5
1	0	0	0	1

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

			5	4	3	2	1	4	3	1	5

1	2	3	4	5
1	1	0	0	1

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

				4	3	2	1	4	3	1	5

1	2	3	4	5
1	1	0	0	2

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

					3	2	1	4	3	1	5

1	2	3	4	5
1	1	0	1	2

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

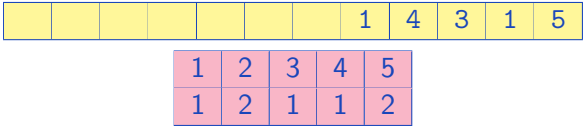
						2	1	4	3	1	5

1	2	3	4	5
1	1	1	1	2

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

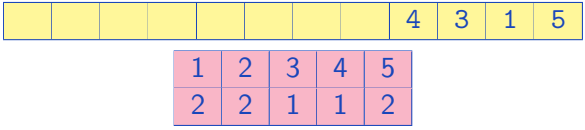
Idea: Just count how many times a certain value was seen
This is Counting sort!



Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

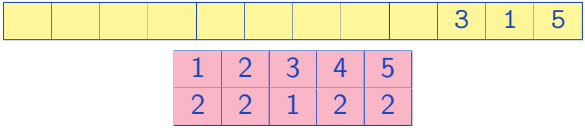
Idea: Just count how many times a certain value was seen
This is Counting sort!



Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!



Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	2	3	4	5
2	2	2	2	2

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	2	3	4	5
3	2	2	2	2

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	2	3	4	5
3	2	2	2	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

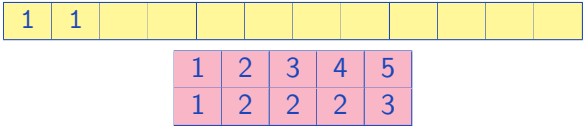
Idea: Just count how many times a certain value was seen
This is Counting sort!

1	2	3	4	5
2	2	2	2	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

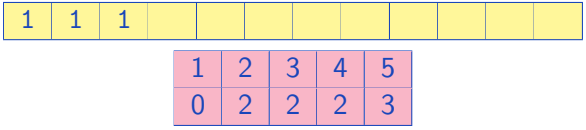
Idea: Just count how many times a certain value was seen
This is Counting sort!



Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!



Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2								

1	2	3	4	5
0	1	2	2	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2	2							

1	2	3	4	5
0	0	2	2	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2	2	3						

1	2	3	4	5
0	0	1	2	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2	2	3	3					

1	2	3	4	5
0	0	0	2	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2	2	3	3	4				

1	2	3	4	5
0	0	0	1	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2	2	3	3	4	4			

1	2	3	4	5
0	0	0	0	3

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2	2	3	3	4	4	5		

1	2	3	4	5
0	0	0	0	2

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

1	1	1	2	2	3	3	4	4	5	5	

1	2	3	4	5
0	0	0	0	1

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

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

1	2	3	4	5
0	0	0	0	0

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

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

1	2	3	4	5
0	0	0	0	0

Running time: $\Theta(N + M)$, additional space: $\Theta(M)$

Given: N integers, each from $[1; M]$, M is quite small. How to sort them efficiently?

- "Quite small" is "we can afford an array of M elements"

Idea: Just count how many times a certain value was seen
This is Counting sort!

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

1	2	3	4	5
0	0	0	0	0

Running time: $\Theta(N + M)$, additional space: $\Theta(M)$

- Faster than comparison-based sorting algorithms when M is small

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

5/Z	1/K	2/E	5/W	4/J	3/U	2/K	1/Z	4/A	3/Q	1/X	5/E

1				
2				
3				
4				
5				

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

	1/K	2/E	5/W	4/J	3/U	2/K	1/Z	4/A	3/Q	1/X	5/E

1				
2				
3				
4				
5	Z			

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

		2/E	5/W	4/J	3/U	2/K	1/Z	4/A	3/Q	1/X	5/E

1	K			
2				
3				
4				
5	Z			

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

			5/W	4/J	3/U	2/K	1/Z	4/A	3/Q	1/X	5/E

1	K			
2	E			
3				
4				
5	Z			

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

				4/J	3/U	2/K	1/Z	4/A	3/Q	1/X	5/E

1	K			
2	E			
3				
4				
5	Z	W		

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

					3/U	2/K	1/Z	4/A	3/Q	1/X	5/E

1	K			
2	E			
3				
4	J			
5	Z	W		

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

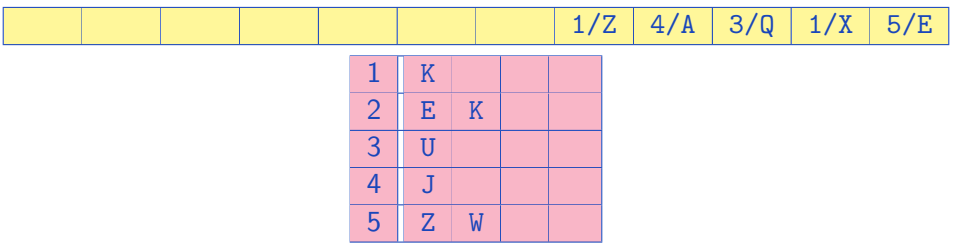
- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

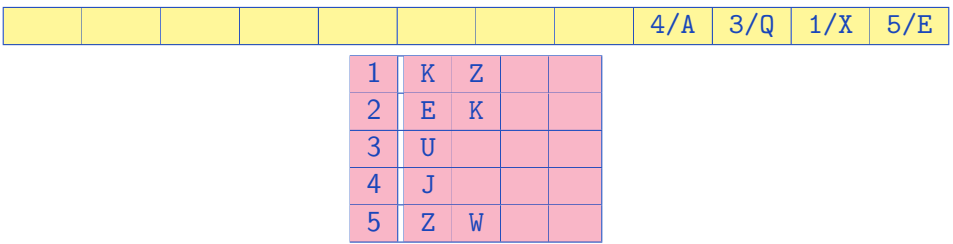
This is Bucket sort!



Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

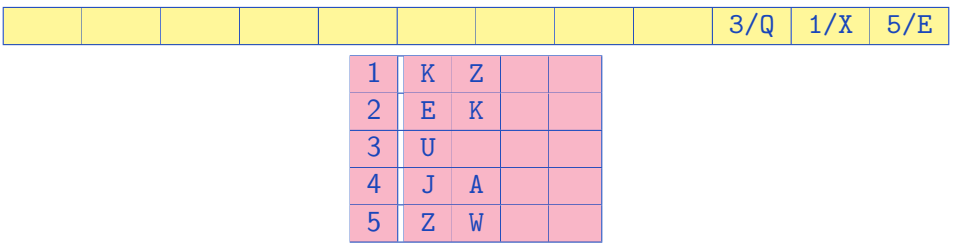
This is Bucket sort!



Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

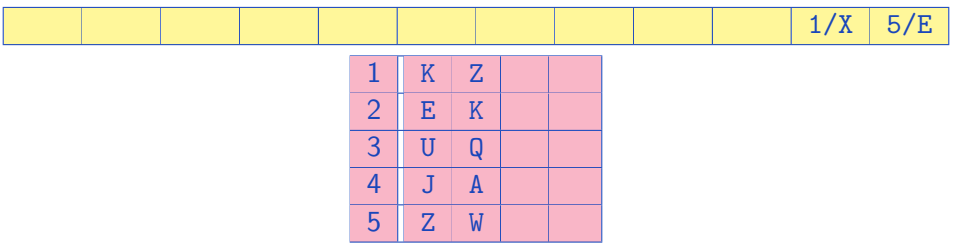
This is Bucket sort!



Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!



Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

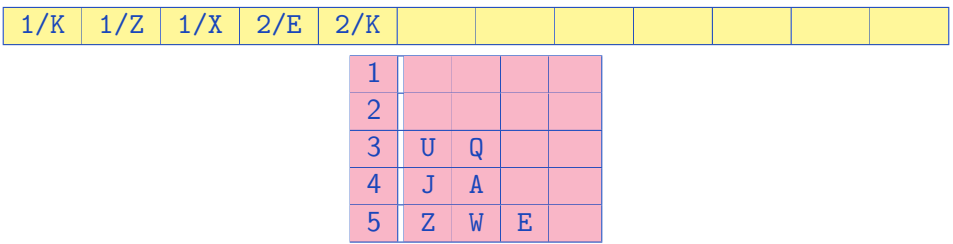
1/K	1/Z	1/X	2/E								

1				
2		K		
3	U	Q		
4	J	A		
5	Z	W	E	

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!



Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

1/K	1/Z	1/X	2/E	2/K	3/U						

1				
2				
3		Q		
4	J	A		
5	Z	W	E	

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q					

1				
2				
3				
4	J	A		
5	Z	W	E	

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q	4/J				

1				
2				
3				
4		A		
5	Z	W	E	

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q	4/J	4/A			

1				
2				
3				
4				
5	Z	W	E	

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q	4/J	4/A	5/Z		

1				
2				
3				
4				
5		W	E	

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q	4/J	4/A	5/Z	5/W	

1				
2				
3				
4				
5			E	

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort!

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q	4/J	4/A	5/Z	5/W	5/E

1				
2				
3				
4				
5				

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort! $\Theta(N + M)$ time, $\Theta(N + M)$ space.

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q	4/J	4/A	5/Z	5/W	5/E

1				
2				
3				
4				
5				

Given: N key-value pairs, keys are integers from $[1; M]$, M is quite small.
How to sort them efficiently?

- We cannot "just count" integers anymore
- But we still can profit from using the keys as array indices
- Have an array of lists of size M, and collect the pairs under their keys

This is Bucket sort! $\Theta(N + M)$ time, $\Theta(N + M)$ space. It is also stable.

1/K	1/Z	1/X	2/E	2/K	3/U	3/Q	4/J	4/A	5/Z	5/W	5/E

1				
2				
3				
4				
5				

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

- Strings are integer arrays. $M = 26$ for Latin alphabet, 256 for ASCII, ...

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

- Strings are integer arrays. $M = 26$ for Latin alphabet, 256 for ASCII, ...
- Big integers are integer arrays. $M = 10$ – or maybe 16, or 65536, or ...

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

- Strings are integer arrays. $M = 26$ for Latin alphabet, 256 for ASCII, ...
- Big integers are integer arrays. $M = 10$ – or maybe 16, or 65536, or ...

Solution:

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

- Strings are integer arrays. $M = 26$ for Latin alphabet, 256 for ASCII, ...
- Big integers are integer arrays. $M = 10$ – or maybe 16, or 65536, or ...

Solution:

- Use bucket sort by the L-th value $\rightarrow \Theta(N + M)$

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

- Strings are integer arrays. $M = 26$ for Latin alphabet, 256 for ASCII, ...
- Big integers are integer arrays. $M = 10$ – or maybe 16, or 65536, or ...

Solution:

- Use bucket sort by the L-th value $\rightarrow \Theta(N + M)$
- Use bucket sort by the $L - 1$-th value $\rightarrow \Theta(N + M)$

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

- Strings are integer arrays. $M = 26$ for Latin alphabet, 256 for ASCII, ...
- Big integers are integer arrays. $M = 10$ – or maybe 16, or 65536, or ...

Solution:

- Use bucket sort by the L-th value $\rightarrow \Theta(N + M)$
- Use bucket sort by the $L - 1$-th value $\rightarrow \Theta(N + M)$
- ...
- Use bucket sort by the 1-st value $\rightarrow \Theta(N + M)$

Given: N integer arrays, each of length L.
Each integer is from $[1; M]$, M is quite small.
How to sort these arrays lexicographically?

- Strings are integer arrays. $M = 26$ for Latin alphabet, 256 for ASCII, ...
- Big integers are integer arrays. $M = 10$ – or maybe 16, or 65536, or ...

Solution:

- Use bucket sort by the L-th value $\to \Theta(N + M)$
- Use bucket sort by the $L - 1$-th value $\to \Theta(N + M)$
- ...
- Use bucket sort by the 1-st value $\to \Theta(N + M)$

This is Radix sort! $\Theta((N + M)L)$ time, $\Theta(N + M)$ space.