MONASH University

MONASH
INFORMATION
TECHNOLOGY

# FIT2004
# Algorithms and Data Structures

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

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

GROUP OF EIGHT AUSTRALIA

# Faculty of Information Technology, Monash University

MONASH University

# Ready?

# Agenda

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

MONASH University

# Let us begin…

MONASH University

- We can sort without comparing elements in a list!

MONASH University

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

MONASH University

# Questions?

# Counting Sort

MONASH University

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list

# Counting Sort

MONASH University

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list


- What is the maximum number?

# Counting Sort

MONASH University

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list


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

# Counting Sort

MONASH University

- Very simple concept
- I am sure we all know this…

- Now let us begin with a list


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

# Counting Sort

MONASH University

- Our input


- We know max is 5

# Counting Sort

MONASH University

- Our input


Anyone noticed
the list is crooked?
#OCDtrigger

- We know max is 5

# Counting Sort

MONASH University

- Our input

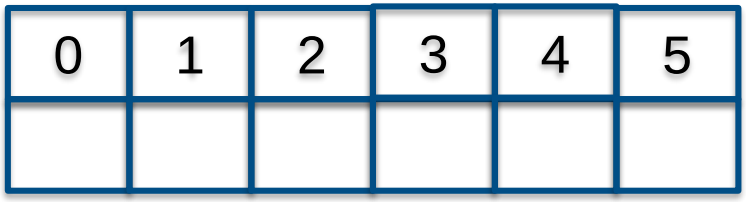

- We know max is 5

# Counting Sort

MONASH University

- Our input


- We know max is 5



# Counting Sort

MONASH University

- Out input

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

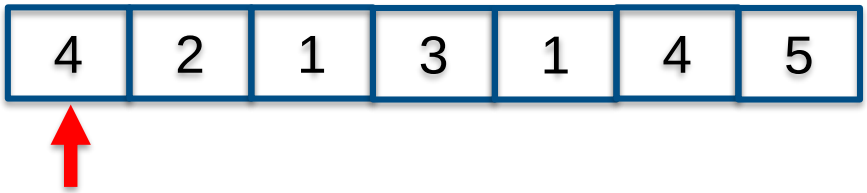
- We know max is 5

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

# Counting Sort

MONASH University

- Our input



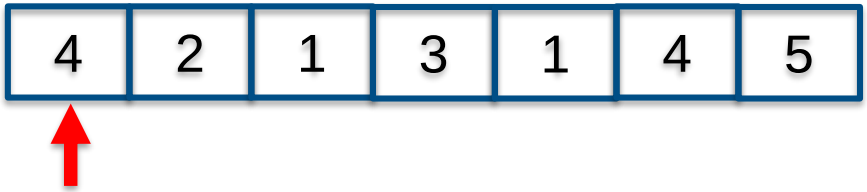

- We know max is 5

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

# Counting Sort

MONASH University

- Our input



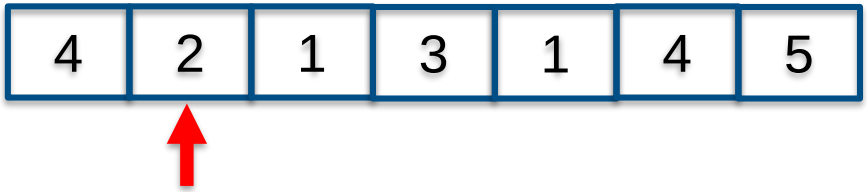

- We know max is 5

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

# Counting Sort

MONASH University

- Our input



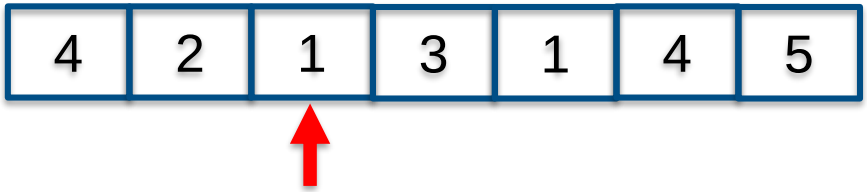

- We know max is 5

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

# Counting Sort

MONASH University

- Out input



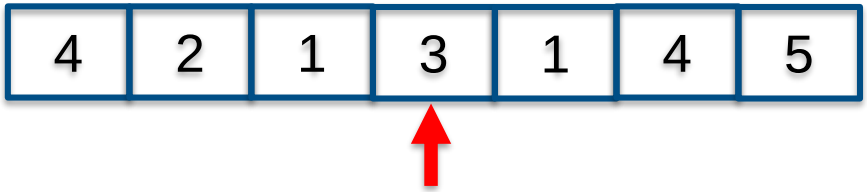

- We know max is 5

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

# Counting Sort

MONASH University

- Our input



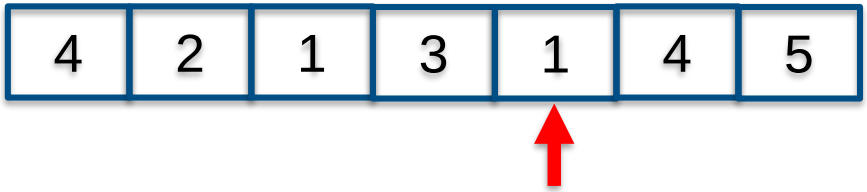

- We know max is 5

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

# Counting Sort

MONASH University

- Our input



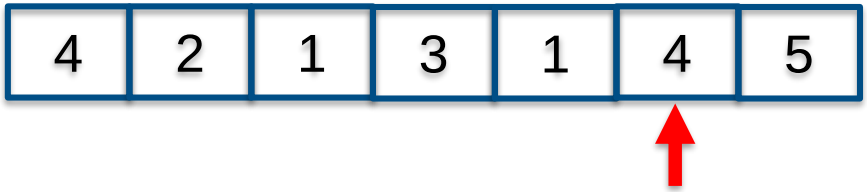

- We know max is 5

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

# Counting Sort

MONASH University

- Our input



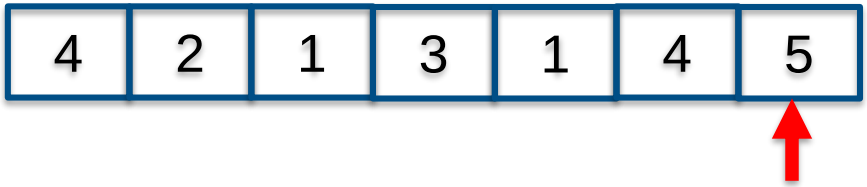

- We know max is 5

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

# Counting Sort

MONASH University

- Our input



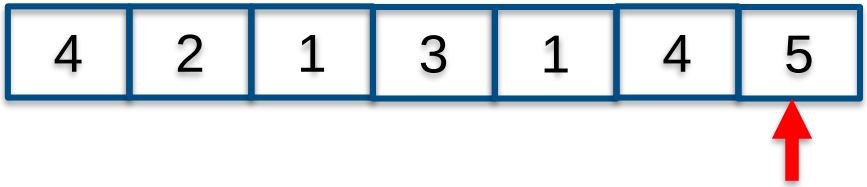

- We know max is 5

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

# Counting Sort

MONASH University

- Our input



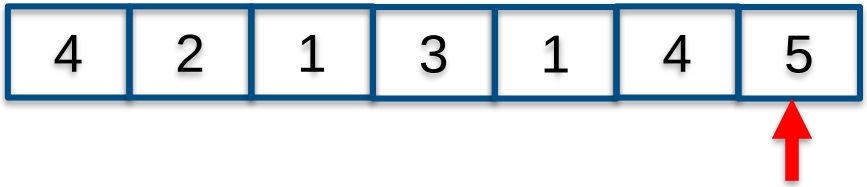

- We know max is 5

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

# Counting Sort

MONASH University

- Our input




- 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

MONASH University

- 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?

# Counting Sort

MONASH University

- Our input

- 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

MONASH University

- Our input

- We know max is 5

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

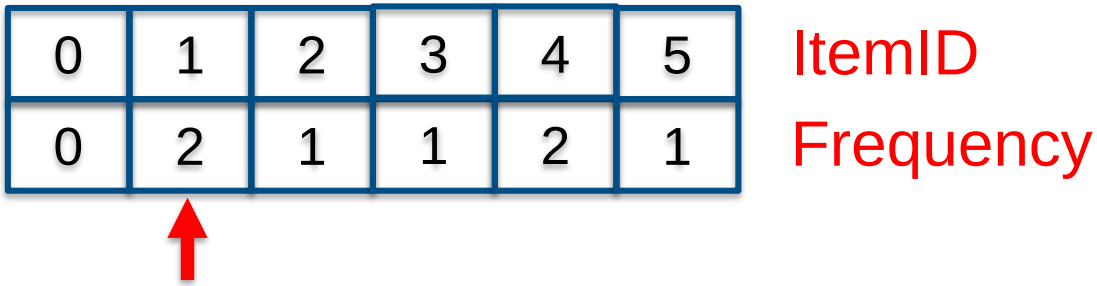
- So how do we sort it now then?

# Counting Sort

MONASH University

- Our input


- We know max is 5



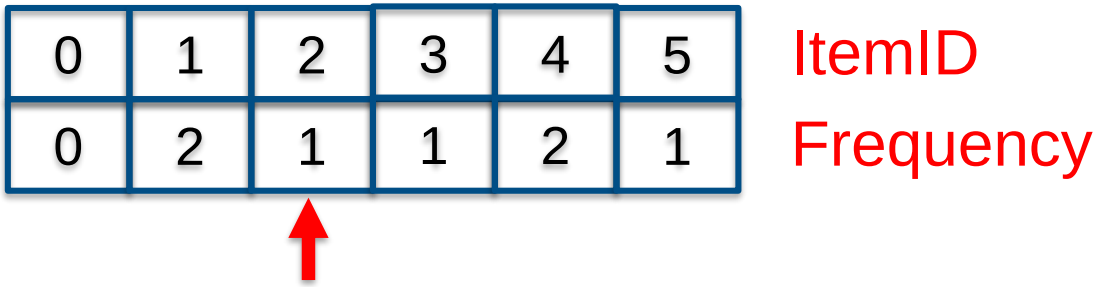

- So how do we sort it now then?

# Counting Sort

MONASH University

- Our input


- We know max is 5



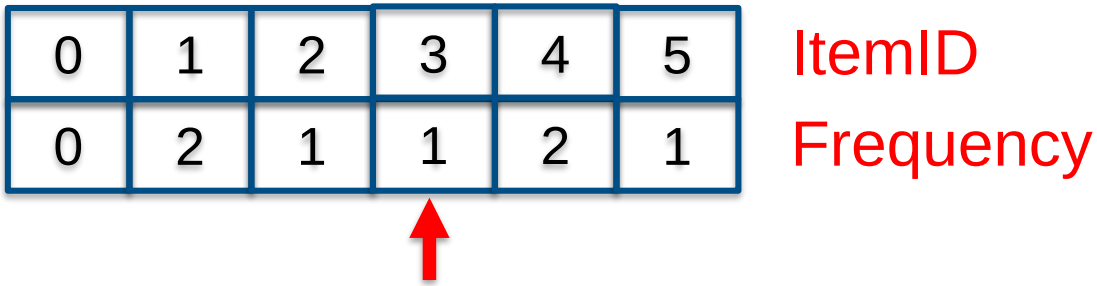

- So how do we sort it now then?

# Counting Sort

MONASH University

- Our input


- We know max is 5



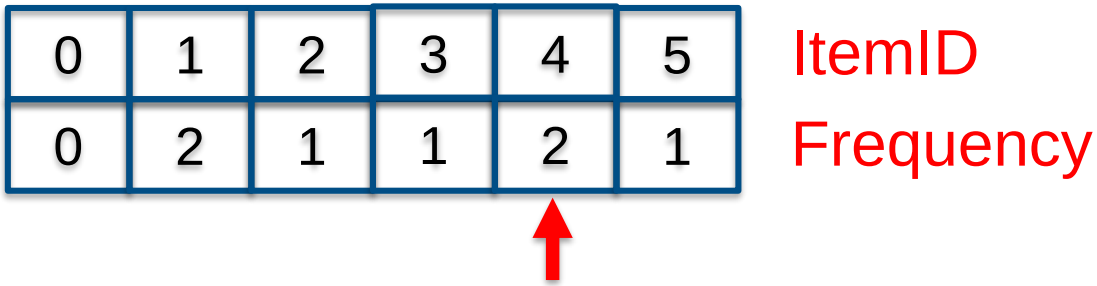

- So how do we sort it now then?

# Counting Sort

MONASH University

- Our input


- We know max is 5




- So how do we sort it now then?

# Counting Sort

MONASH University

- 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?

# Counting Sort

MONASH University

- 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?

# Counting Sort
## Complexity

- Time?

# Counting Sort
## Complexity

MONASH University

- Time?
  - Find the maximum O(N)

# Counting Sort
## Complexity

MONASH University

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

# Counting Sort
## Complexity

MONASH University

- 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

MONASH University

- 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

MONASH University

- 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 | Index |
|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 1 | 2 | 1 | Frequency |

# Counting Sort
## Complexity

MONASH University

- 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

# Counting Sort
## Complexity

MONASH University

- 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)

# Counting Sort
## Complexity

MONASH University

- 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)

# Counting Sort
## Complexity

MONASH University

- 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 << N for this to be good
    - Else even N log N < M

# Counting Sort
## Complexity

MONASH University

- 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 << N for this to be good
  - If we are doing alphabets only, then the M = 26 for the 26 character (after ascii conversion + maths)

MONASH University

# Questions?

# Counting Sort
## Complexity

- Space?

# Counting Sort
## Complexity

MONASH University

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

# Counting Sort
## Complexity

MONASH University

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

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

MONASH University

# Questions?

# Counting Sort

MONASH University

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

# Counting Sort

MONASH University

- 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

MONASH University

# Questions?

# Counting Sort

## Issue…

- Now imagine the following:

# Counting Sort

## Issue…

MONASH University

- Now imagine the following:


  - What is my complexity?

# Counting Sort

## Issue…

MONASH University

- Now imagine the following:

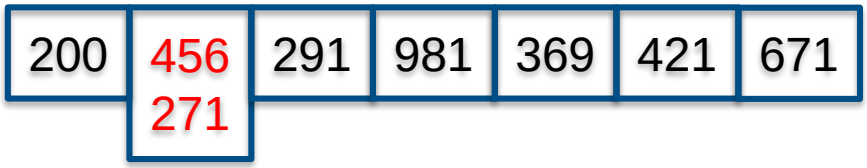

  - What is my complexity?
    - Time…
    - Space…

# Counting Sort

## Issue…

MONASH University

- Now imagine the following:



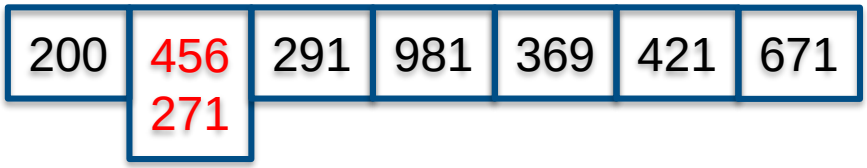

  - What is my complexity?
    - Time…
    - Space…

  - What if one of the value is LARGE

# Counting Sort

## Issue…

- Now imagine the following:




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

  - What if one of the value is LARGE

M is large!!!

# Counting Sort

## Issue…

MONASH University

- Now imagine the following:


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

MONASH University

# Questions?

# Counting Sort

MONASH University

- Stable?

# Counting Sort

MONASH University

- Stable?
  - No
  - We only remember the frequency

# Counting Sort

MONASH University

- Stable?
  - No
  - We only remember the frequency
- But can we make it stable?

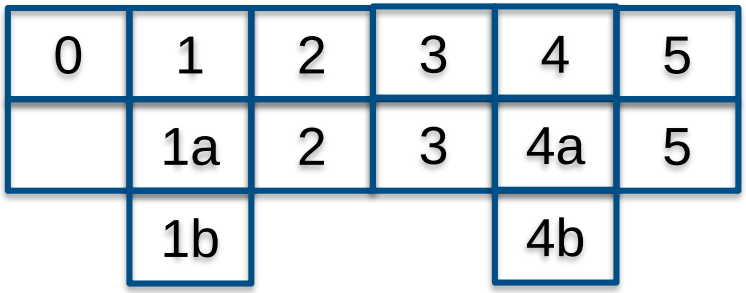
# Counting Sort

MONASH University

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

# Counting Sort

MONASH University


Index

Frequency

# Counting Sort

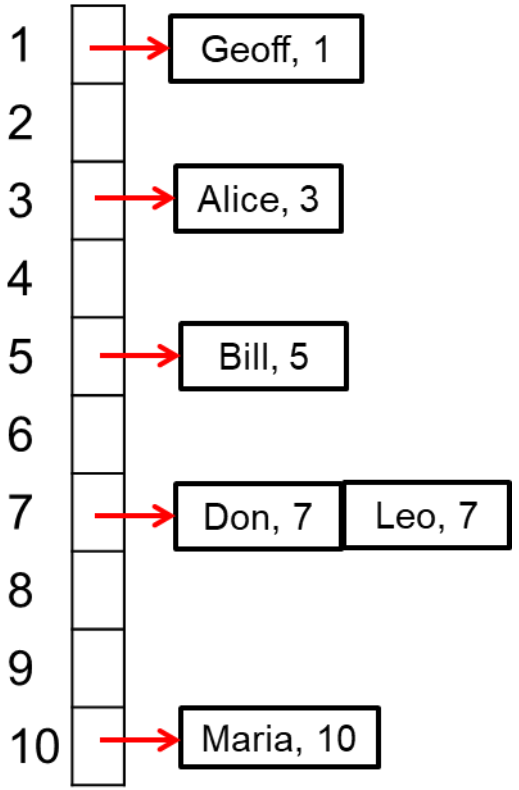
MONASH University

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

# Counting Sort

MONASH University

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

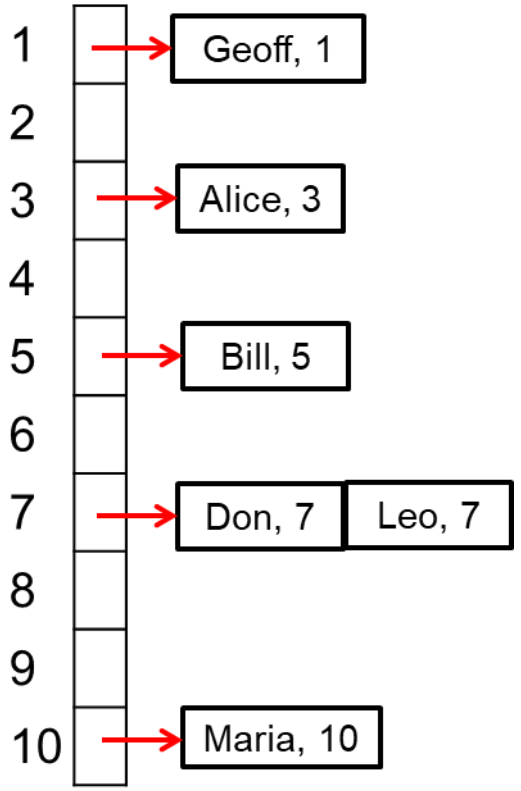



| Marks | 3 | 5 | 7 | 1 | 7 | 10 |
|---|---|---|---|---|---|---|
| Name | Alice | Bill | Don | Geoff | Leo | Maria |

# Counting Sort

MONASH University

- 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

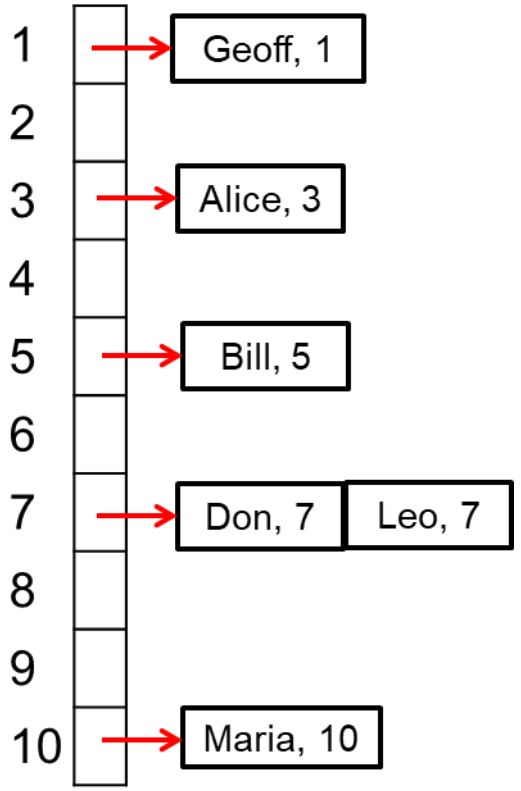



| Marks | 3 | 5 | 7 | 1 | 7 | 10 |
|---|---|---|---|---|---|---|
| Name | Alice | Bill | Don | Geoff | Leo | Maria |

# Counting Sort

MONASH University

- 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?

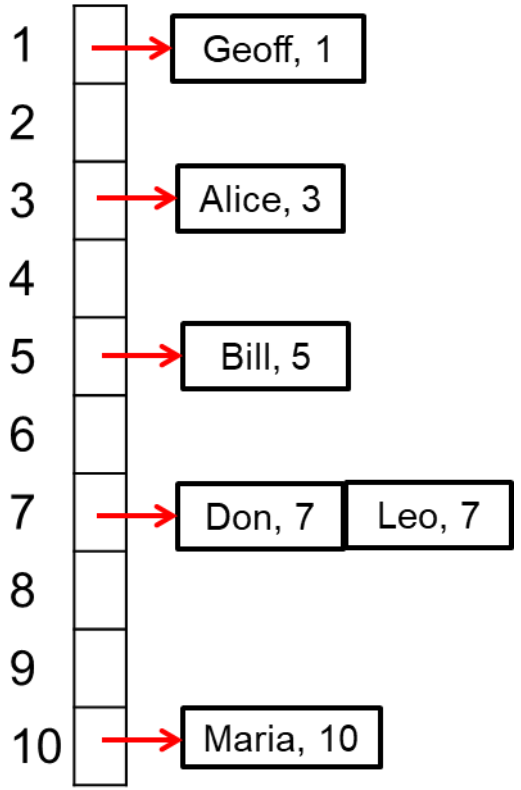



| Marks | 3 | 5 | 7 | 1 | 7 | 10 |
|---|---|---|---|---|---|---|
| Name | Alice | Bill | Don | Geoff | Leo | Maria |

# Counting Sort

MONASH University

- 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?



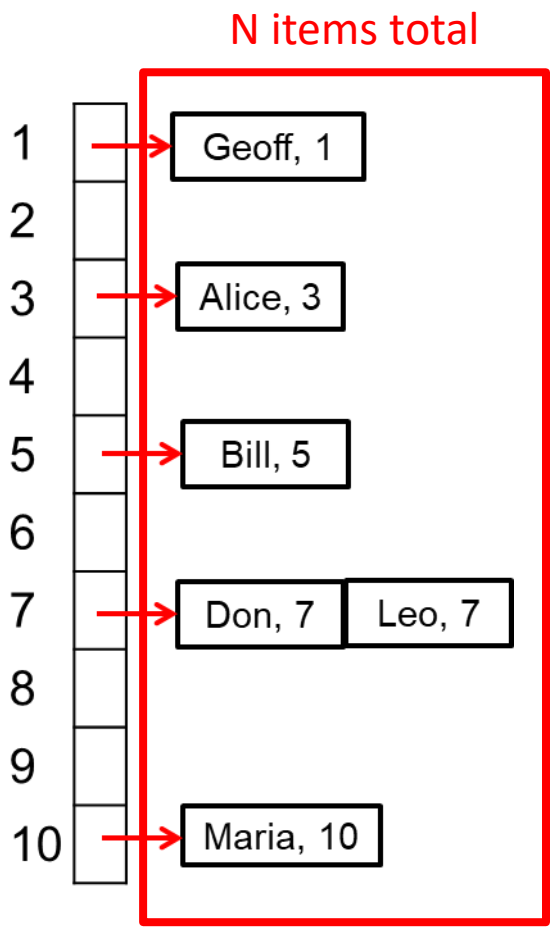

| Marks | 3 | 5 | 7 | 1 | 7 | 10 |
|---|---|---|---|---|---|---|
| Name | Alice | Bill | Don | Geoff | Leo | Maria |

N items

# Counting Sort

MONASH University

- 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

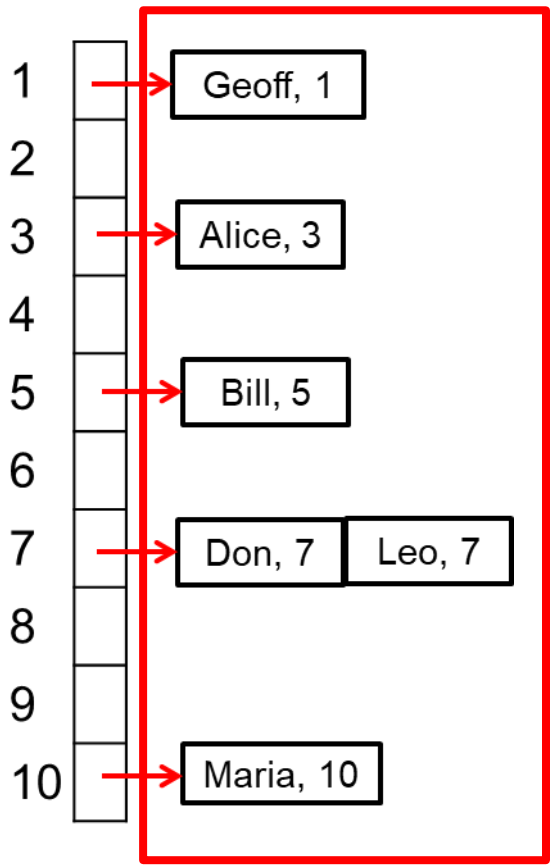
# Counting Sort

MONASH University

Not O(N*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

MONASH University

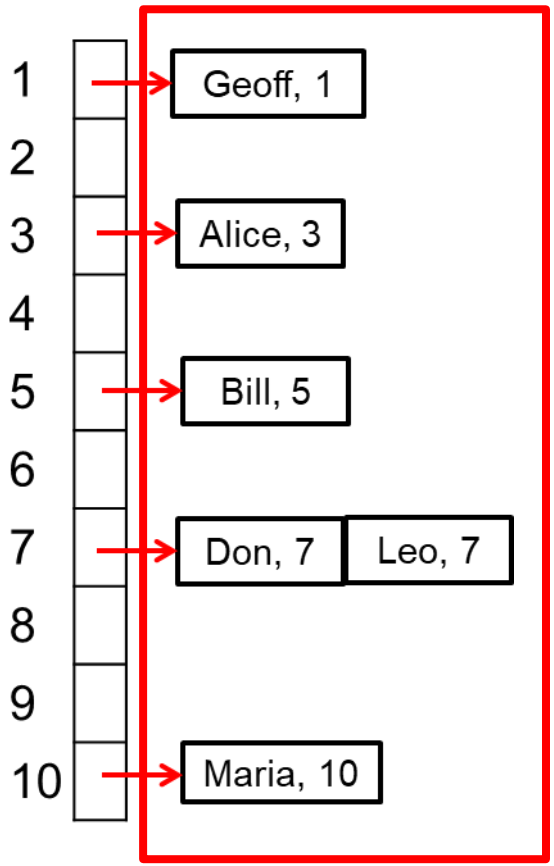
Not O(N*M)

N items total

- Stable?
  - No
  - We only remember the frequency

**VERY COMMON MISCONCEPTION**

- 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

MONASH University

# Questions?

# Counting Sort

MONASH University

- 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,
- count[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,
- count[key] += 1

Construct position:

- Initialise first position as a 1

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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,
- count[key] += 1

Construct position:

- Initialise first position as a 1
- position[i] = position[i-1] + count[i-1]

Construct output

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

count

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

position

| | |
|---|---|
| 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 |

MONASH University

# Questions?

# Counting Sort

MONASH University

- 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?

MONASH University

# Questions?

MONASH University

Have a break again!

# Counting Sort

## Remember this issue…

MONASH University

- Now imagine the following:


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

# Counting Sort

## Remember this issue…

MONASH University

- Now imagine the following:


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

# Counting Sort

## Remember this issue…

MONASH University

- Now imagine the following:


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

MONASH University

# Questions?

# Radix Sort

## A different outlook…

MONASH University

- With this input…

# Radix Sort

## A different outlook…

MONASH University

- With this input…


  - What if we view it differently?

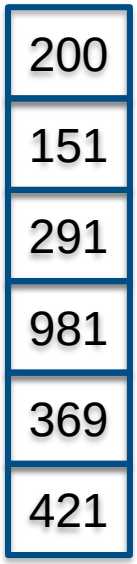
# Radix Sort

## A different outlook…

MONASH University

- With this input…


  - What if we view it differently?



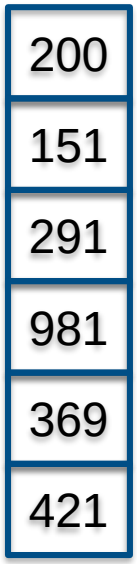
# Radix Sort

## A different outlook…

MONASH University

- With this input…


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



# Radix Sort

## A different outlook…

MONASH University

- 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…

MONASH University

- 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…

MONASH University

- 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…

MONASH University

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

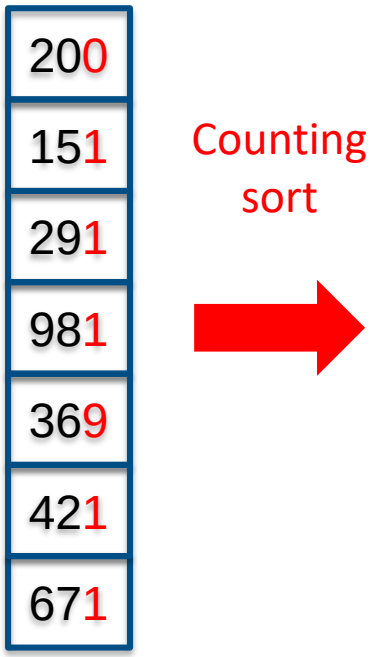


| |
|---|
| 200 |
| 151 |
| 291 |
| 981 |
| 369 |
| 421 |
| 671 |

# 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

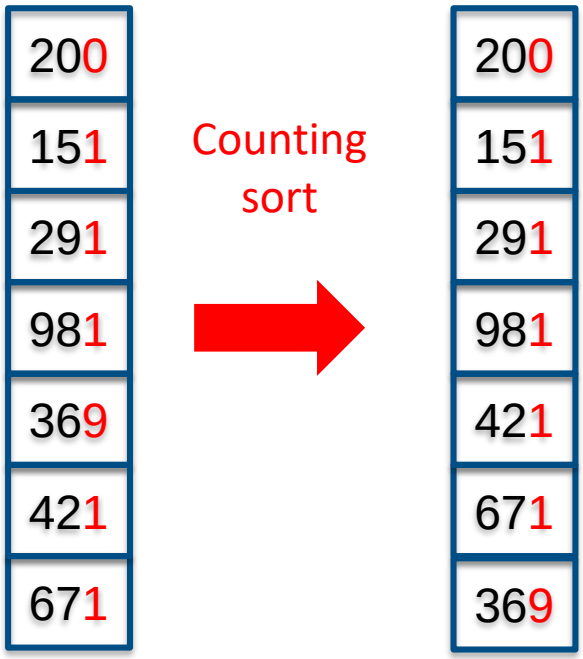



Counting sort

# 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

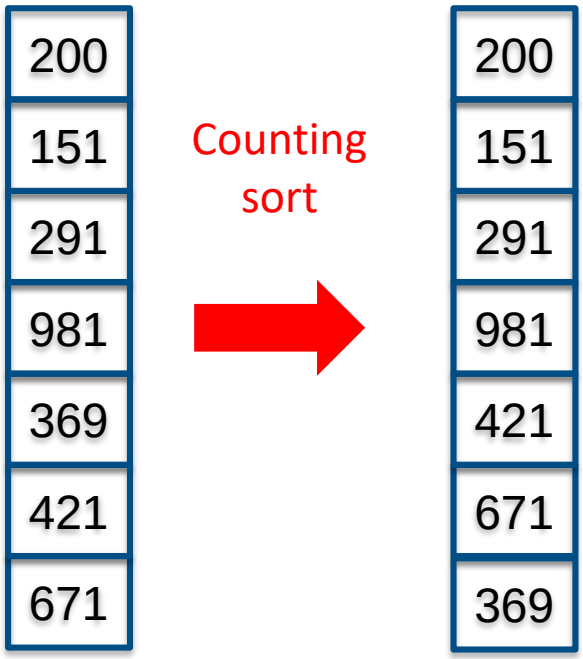


# Radix Sort

## A different outlook…

MONASH University

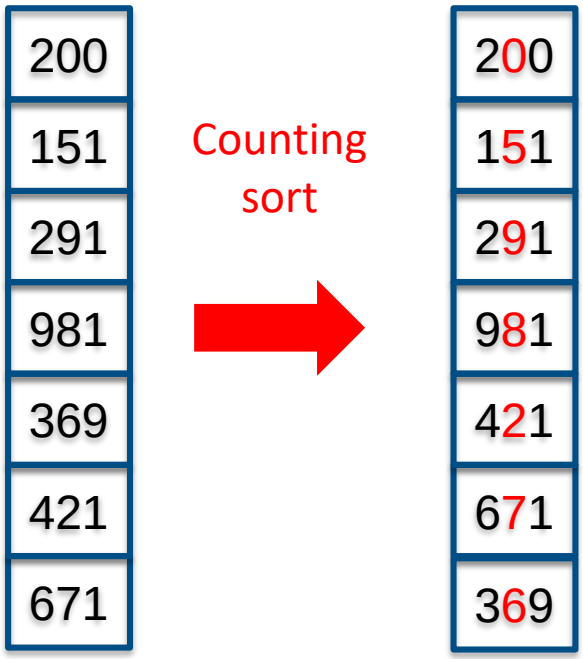
- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



# 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

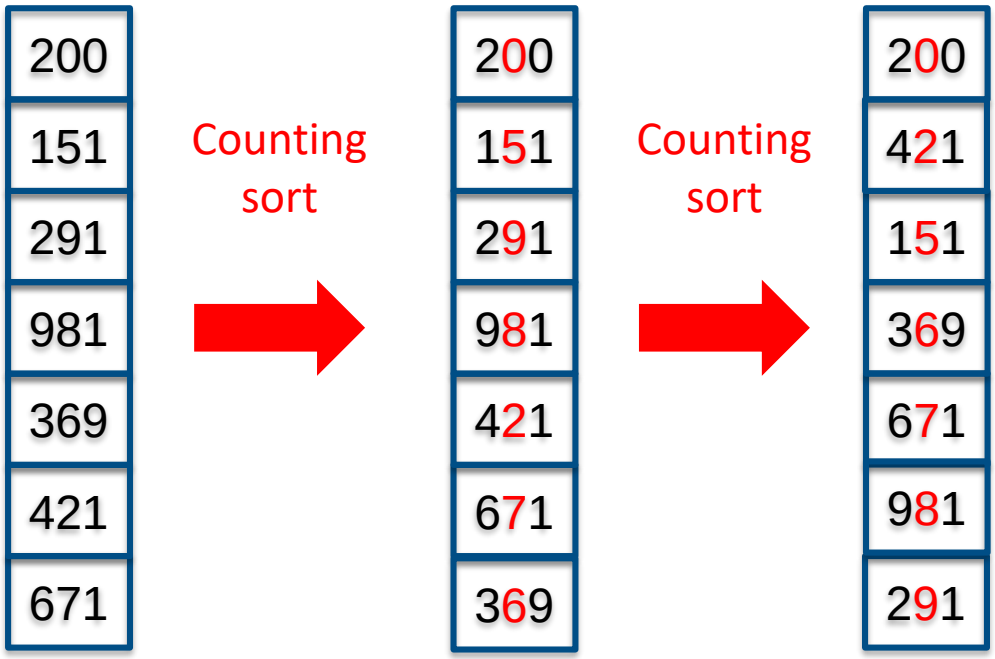


# Radix Sort

## A different outlook…

MONASH University

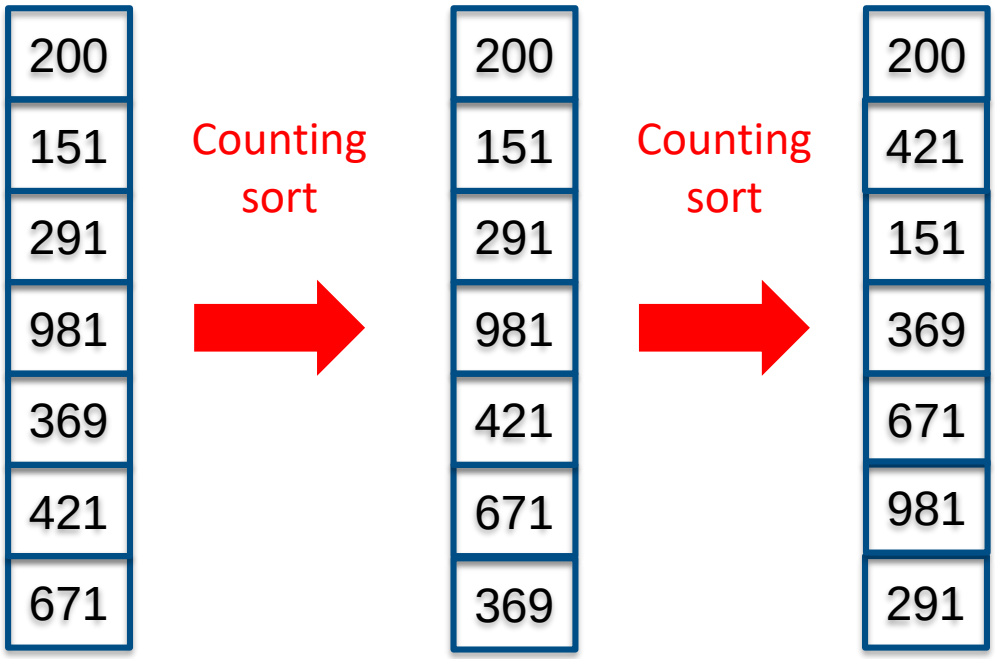
- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



# 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

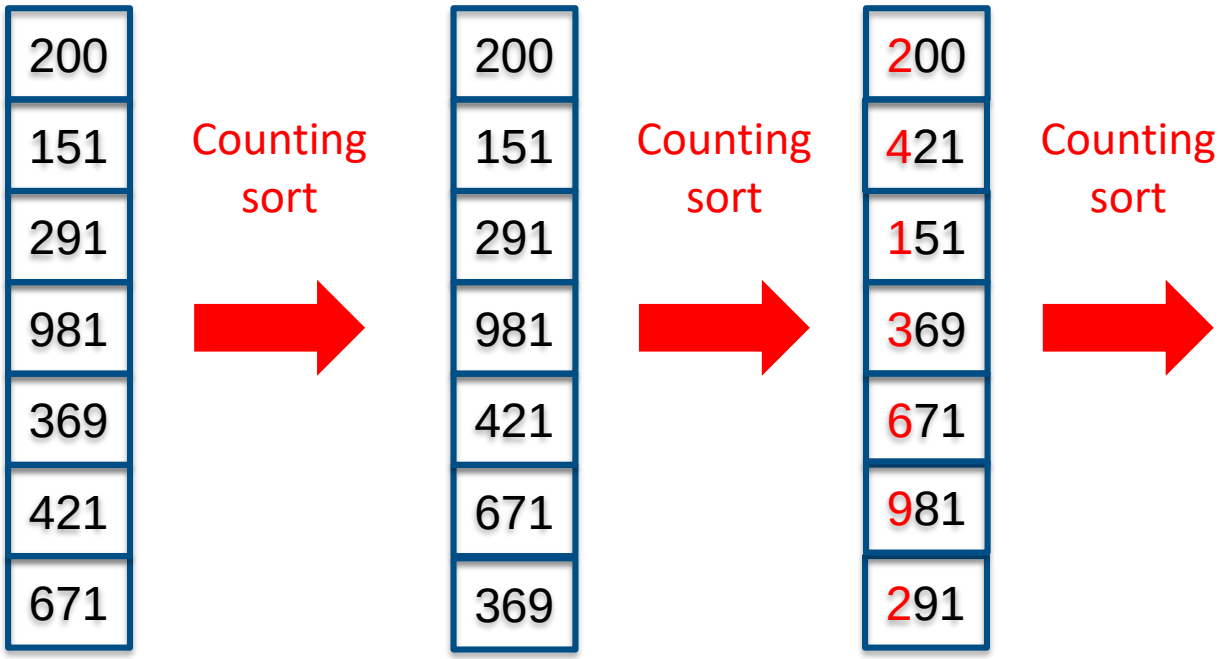


# Radix Sort

## A different outlook…

MONASH University

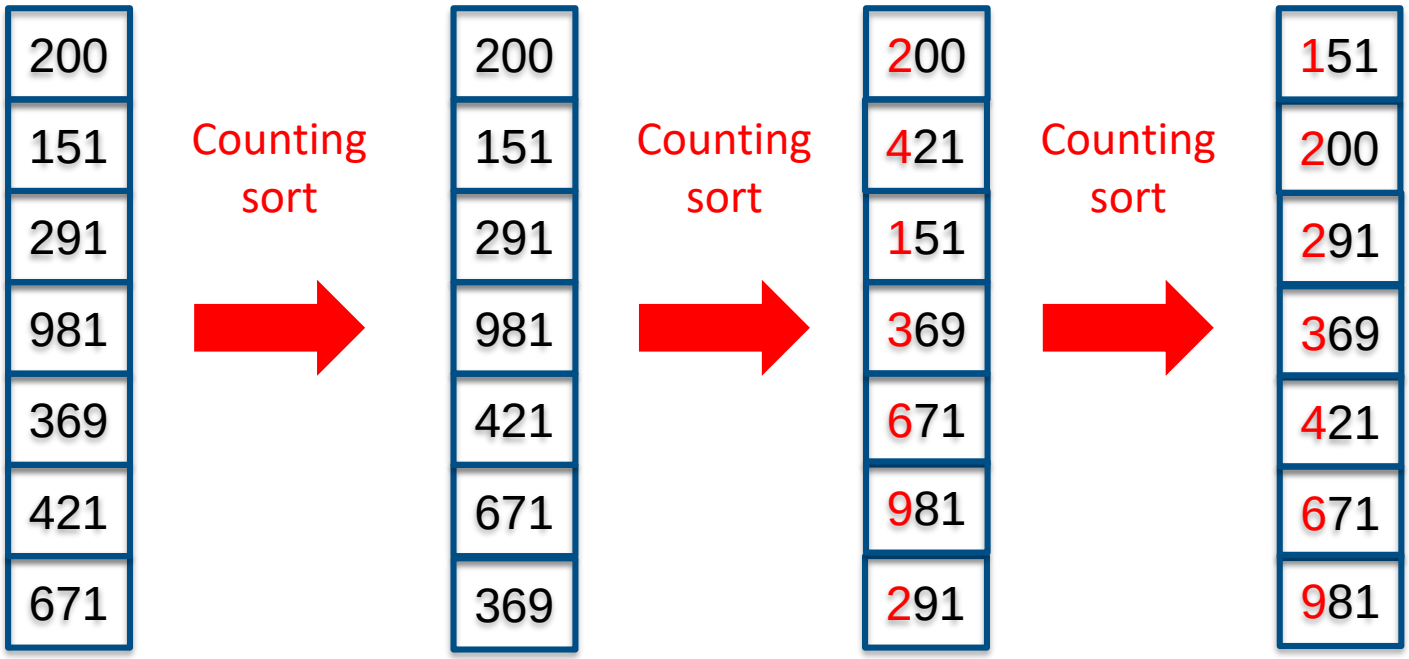
- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



# Radix Sort

## A different outlook…

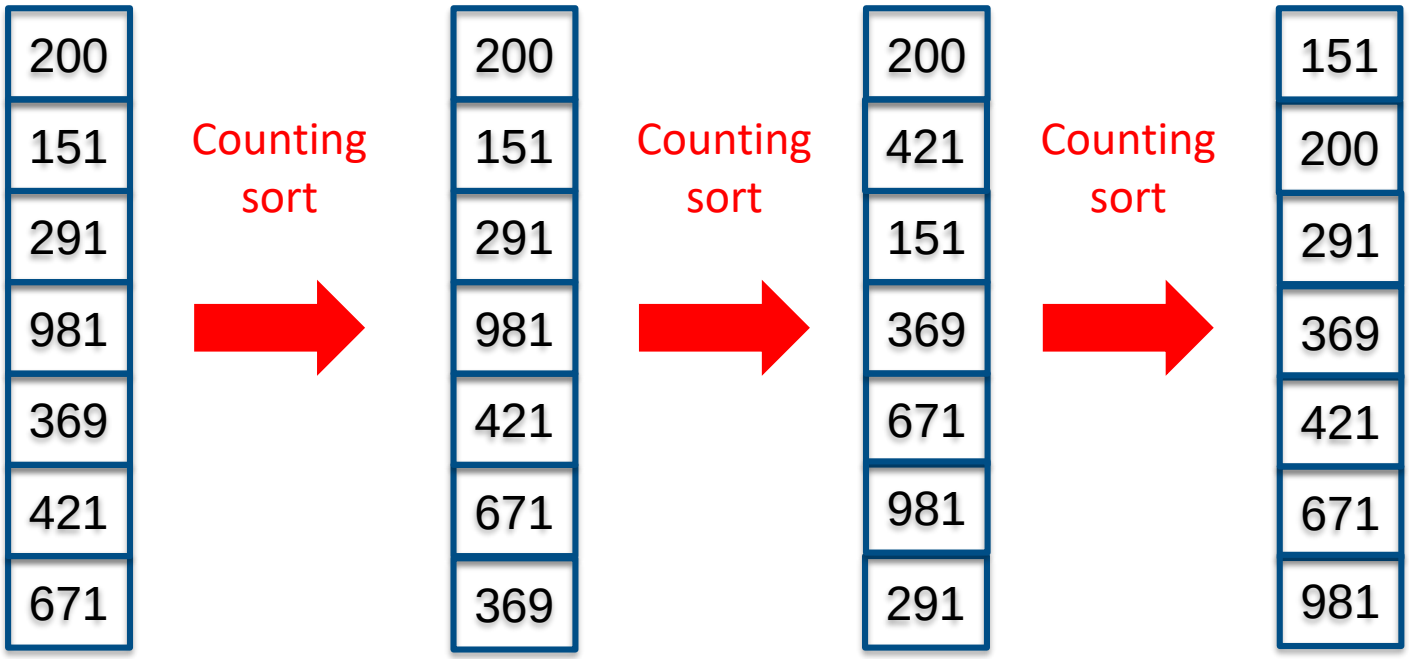
- With this input…
  - What if we view it differently? How would we sort it?
    - Right most digit = least significant
    - Left most digit = most significant



# 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



MONASH University

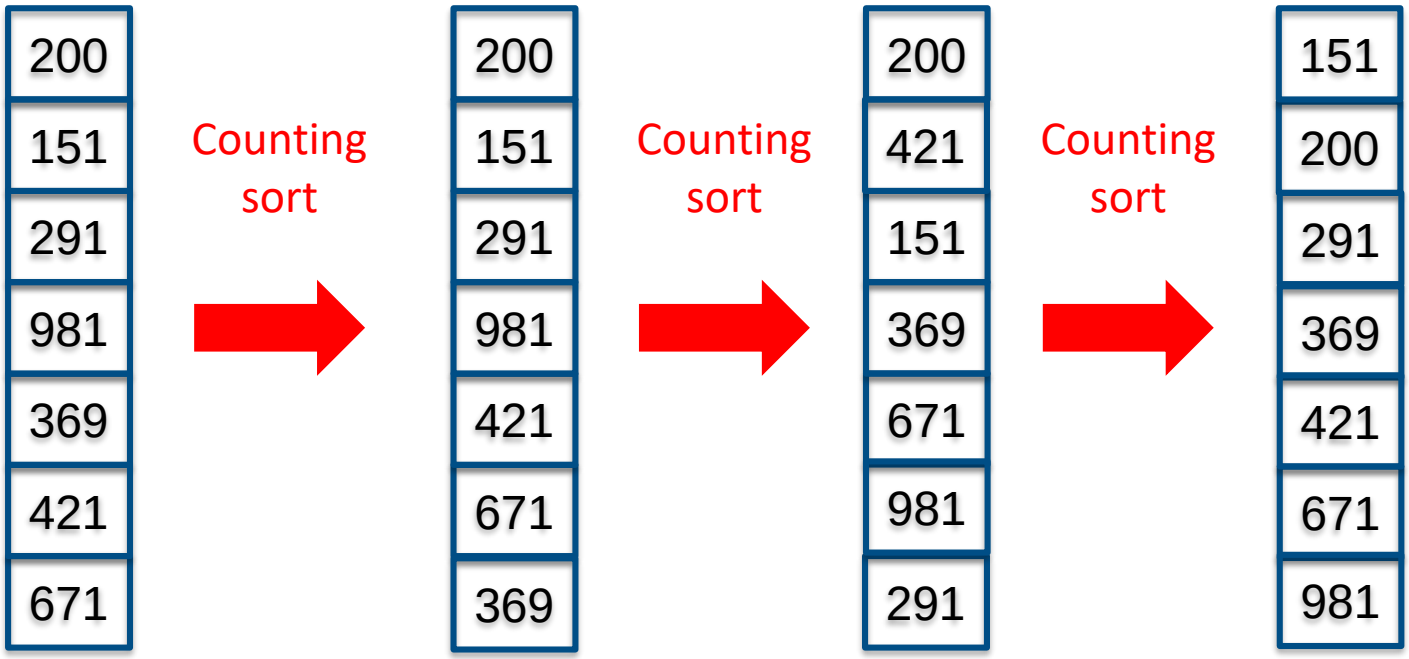
# Questions?

# Radix Sort

## A different outlook…

MONASH University

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

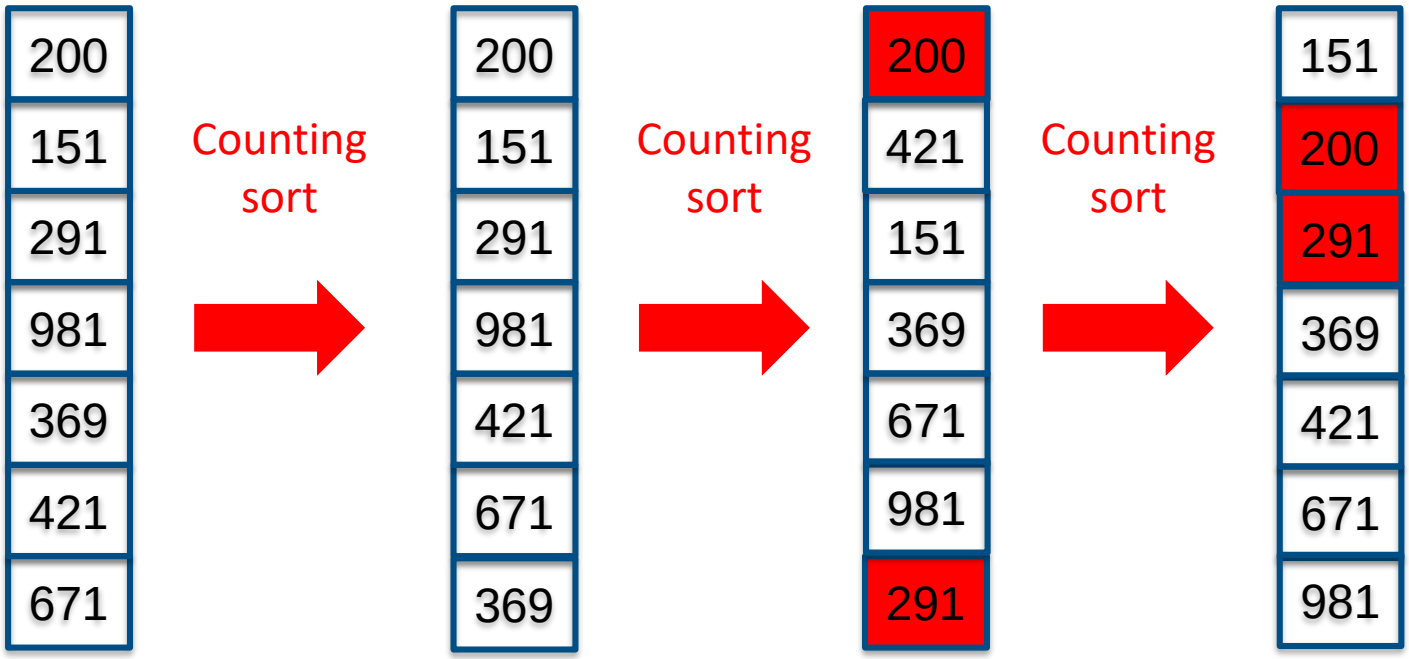


# Radix Sort

## A different outlook…

MONASH University

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

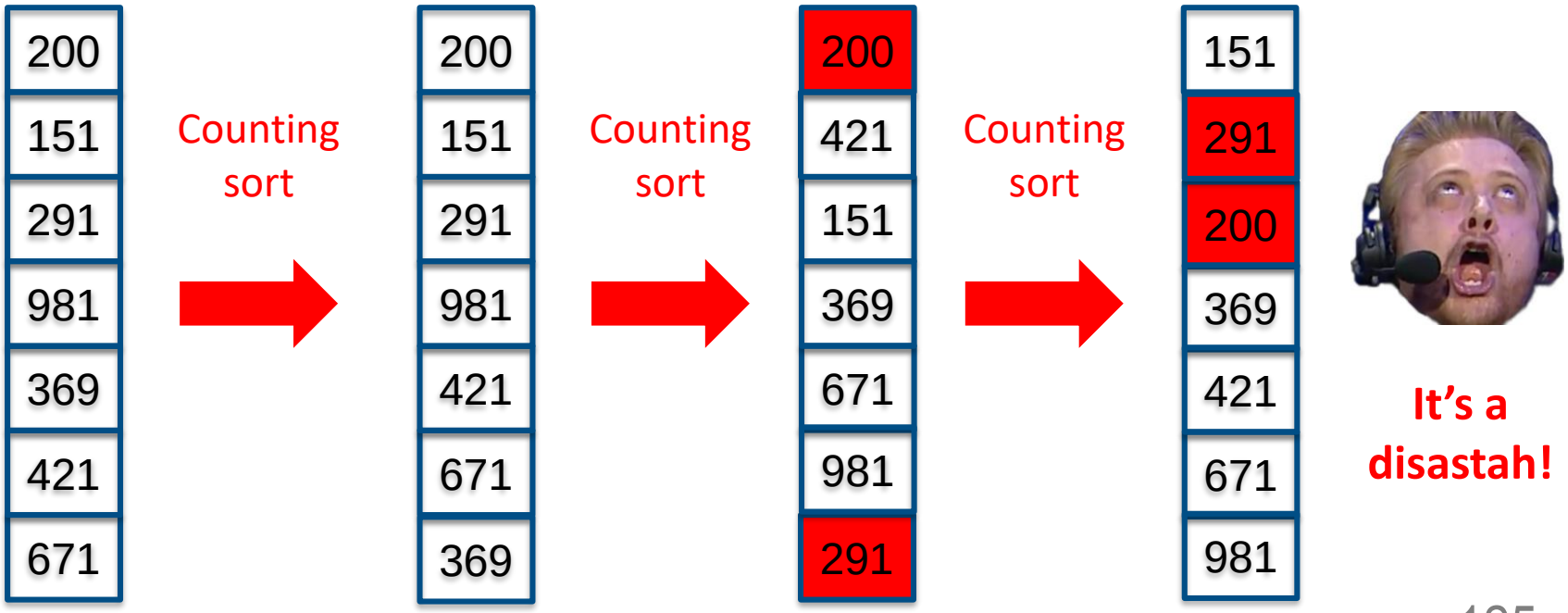


# Radix Sort

## A different outlook…

MONASH University

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



MONASH University

# Questions?

# Radix Sort
## Complexity

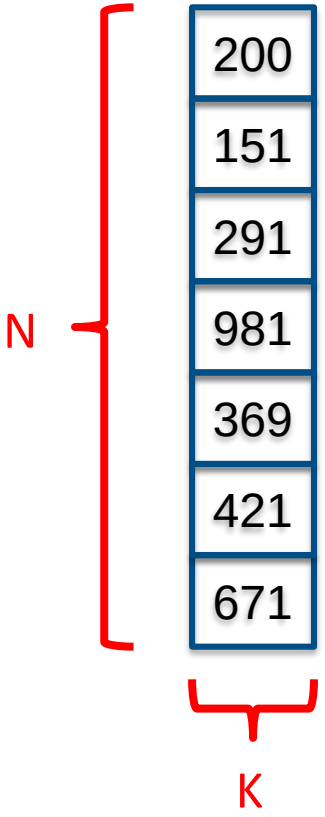
MONASH University

- What is the complexity?
  - Time
  - Space

# Radix Sort
## Complexity

MONASH University

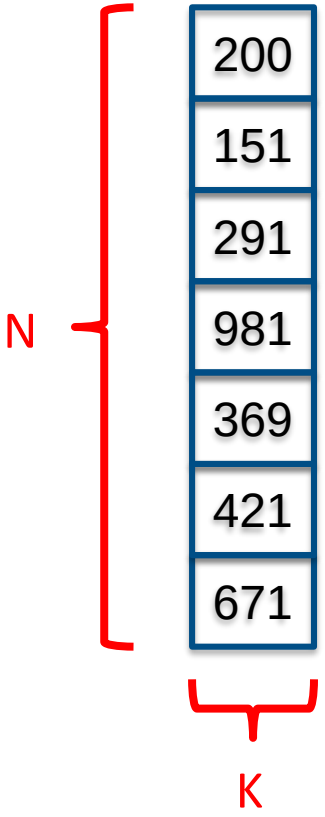
- What is the complexity?
  - Time
  - Space



# Radix Sort
## Complexity

MONASH University

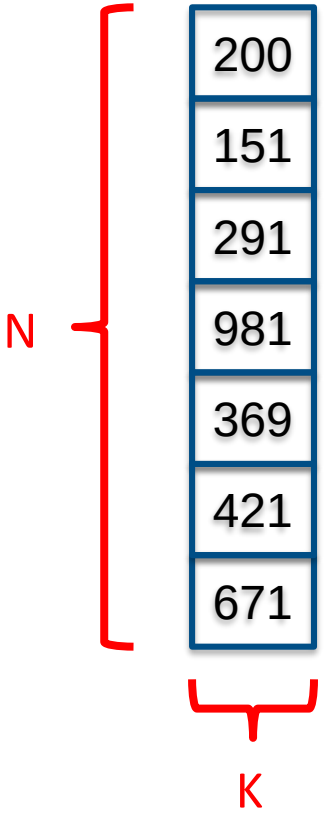
- What is the complexity?
  - Time
    - O(KN)?
  - Space



# Radix Sort
## Complexity

MONASH University

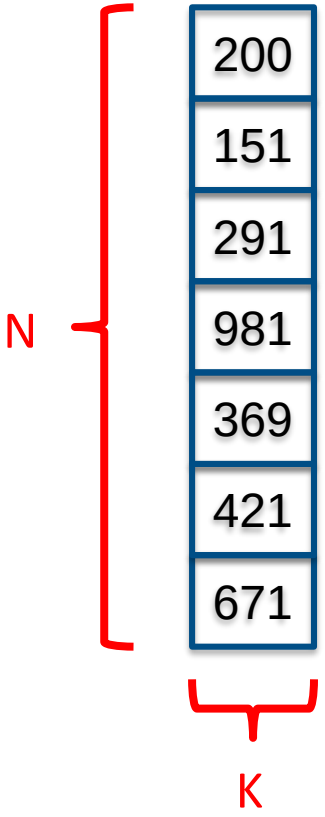
- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
  - Space



# Radix Sort
## Complexity

MONASH University

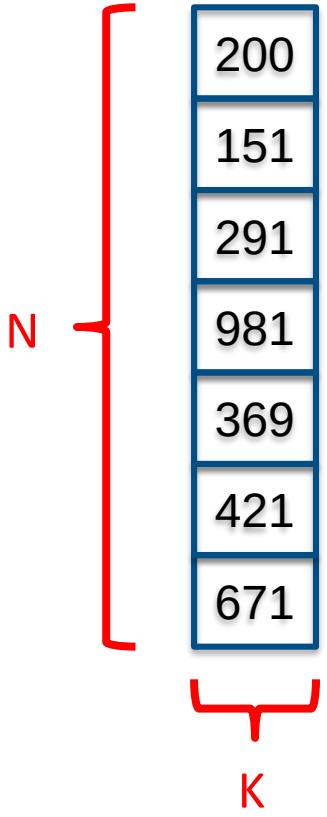
- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max
  - Space



# Radix Sort
## Complexity

MONASH University

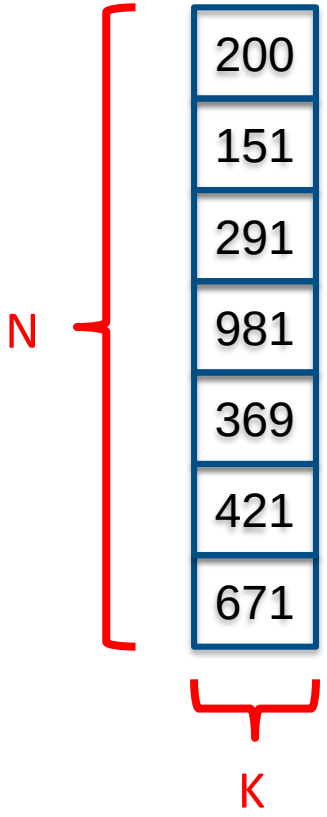
- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
  - Space



# Radix Sort
## Complexity

MONASH University

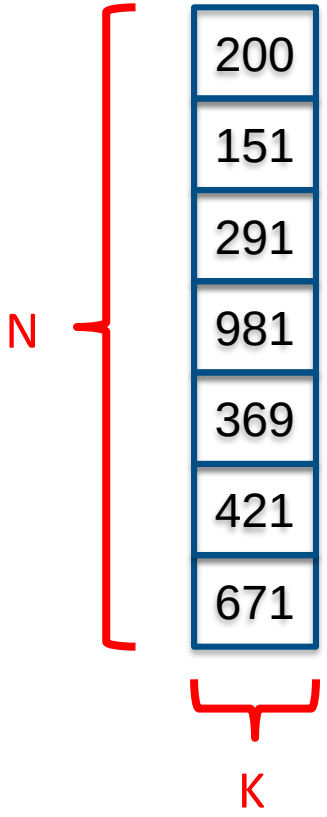
- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
  - Space



# Radix Sort
## Complexity

MONASH University

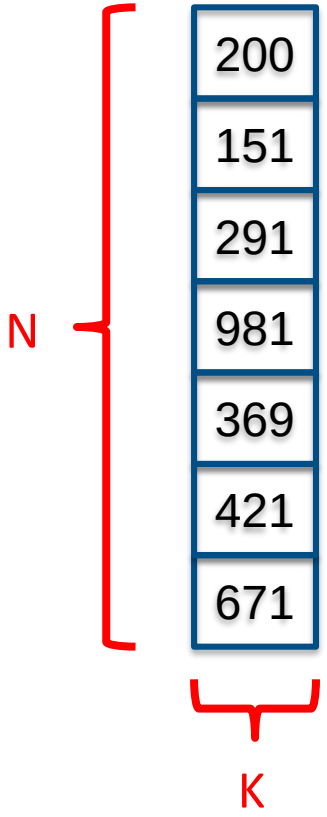
- What is the complexity?
  - Time
    - O(KN) + O(KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max
      giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space



# Radix Sort
## Complexity

MONASH University

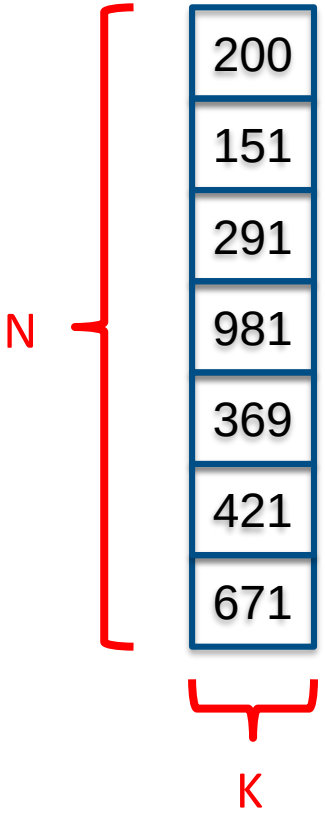
- What is the complexity?
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space



# Radix Sort
## Complexity

MONASH University

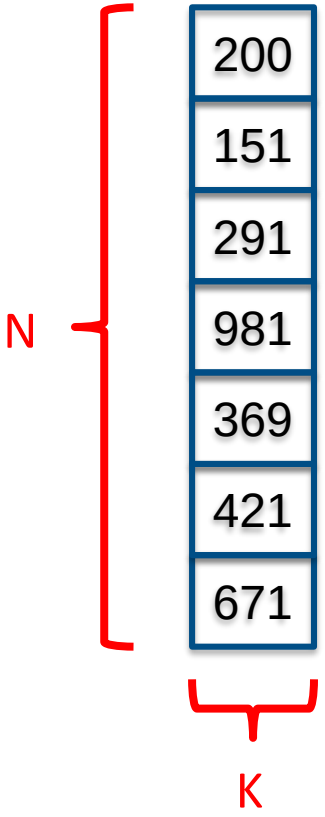
- What is the complexity?
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)



# Radix Sort
## Complexity

MONASH University

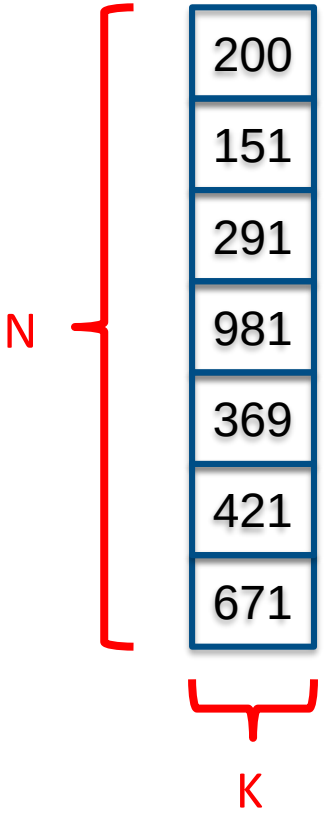
- What is the complexity?
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)
    - Each counting sort needs O(M+N)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



# Radix Sort
## Complexity

MONASH University

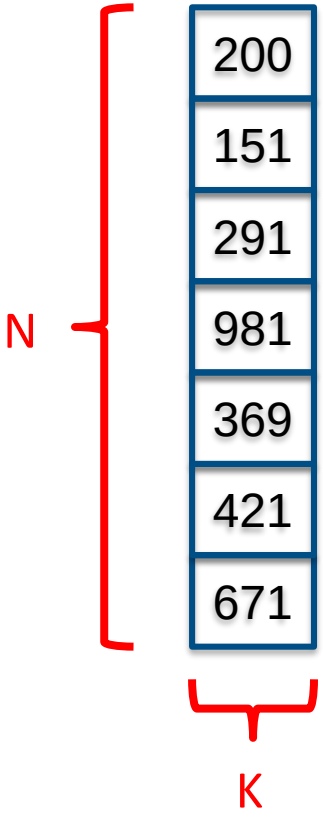
- What is the complexity?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input is O(KN)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



# Radix Sort
## Complexity

MONASH University

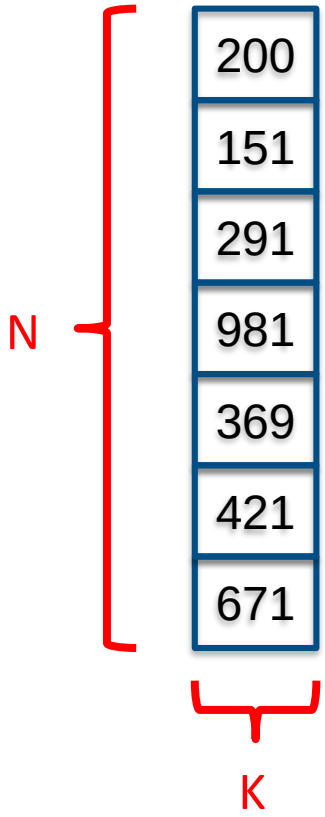
- What is the complexity?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)
  - Space
    - Input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



# Radix Sort
## Complexity

MONASH University

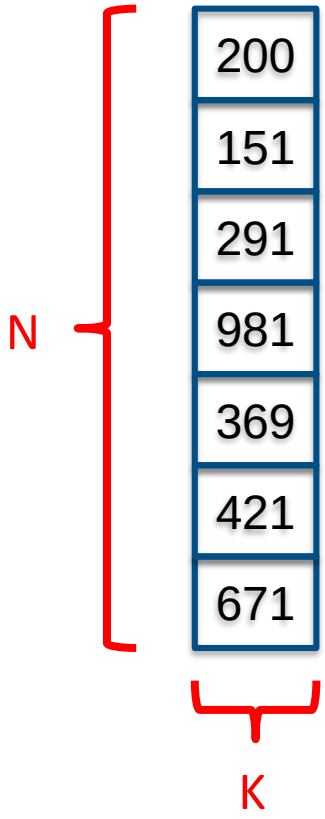
- What is the complexity?
  - Better than merge sort O(k N log N)!
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving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



# Radix Sort
## Complexity

MONASH University

- 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) ≈ 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) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N) <- why no K? Come ask me if interested…



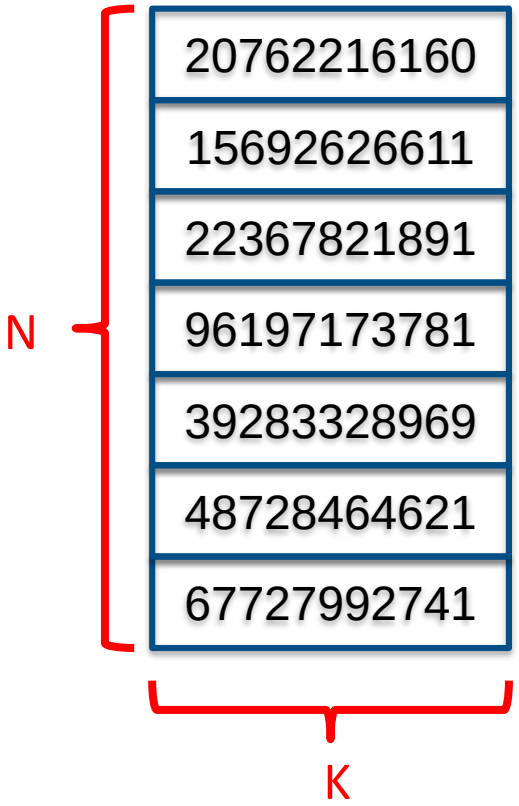
MONASH University

# Questions?

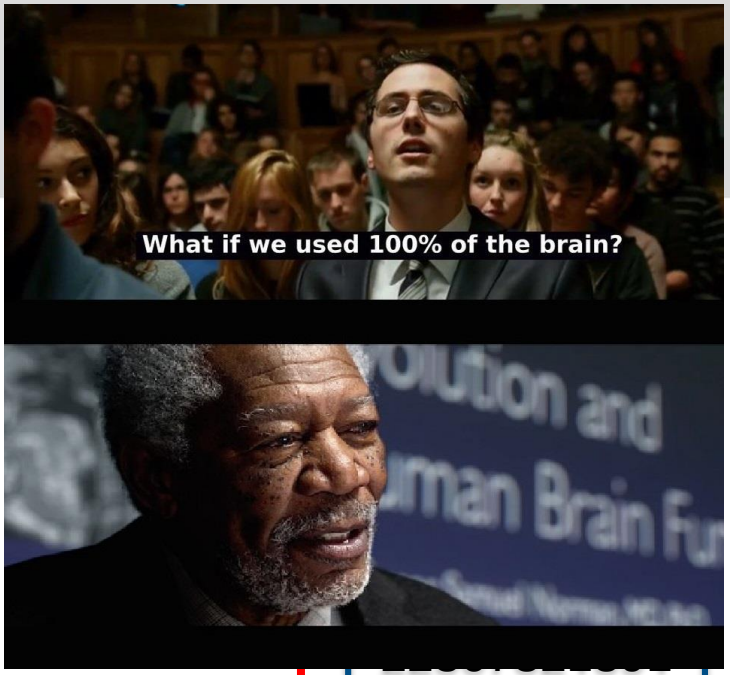
# Radix Sort
## Complexity

MONASH University

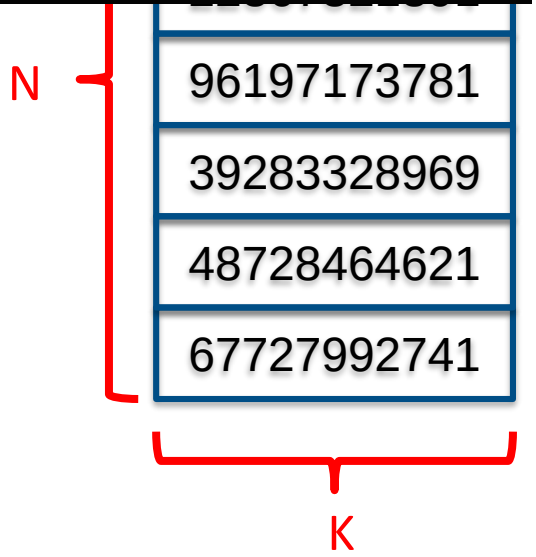
- What is the complexity?
  - What if k is bigger?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)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



# Radix Sort
## Complexity



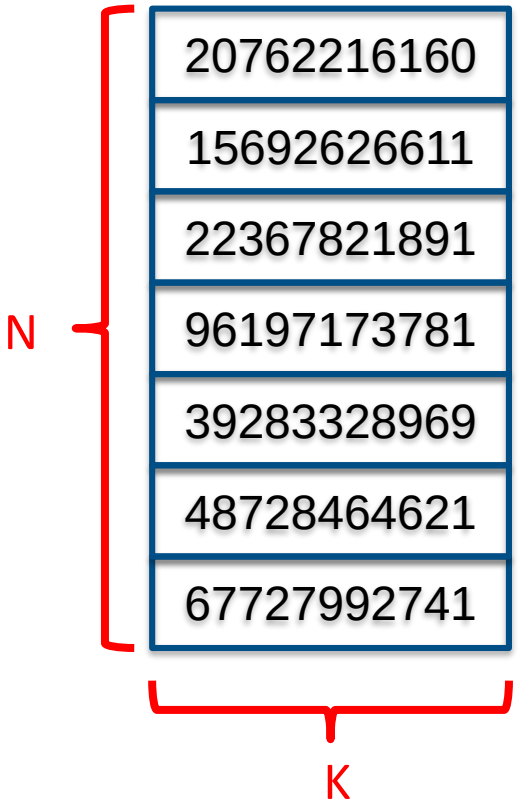

- What is the complexity?
  - What if k is bigger?
  - But we know M = 10 for 0, 1, …, 9
  - Time
    - O(KN + KM) ≈ O(KN)
      where M is the number of unique characters
    - Why? Recall counting sort, we account for the max giving us O(N+M)
    - Then we have K columns
      giving us O(K) * O(N+M)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



# Radix Sort
## Complexity

MONASH University

- What is the complexity?
  - What if k is bigger?
  - We increase M = 100 for 0, 1, …, 99
  - Time
    - O(KN + KM) ≈ 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) ≈ O(KN)
    - Auxiliary is O(M + N) ≈ O(N)



# Radix Sort
## Complexity

MONASH University

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

# Radix Sort
## Complexity

MONASH University

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

- M is the base

# Radix Sort
## Complexity

MONASH University

- 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

MONASH University

- 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

MONASH University

- 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

MONASH University

- 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

MONASH University

- 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

# Radix Sort
## Complexity

MONASH University

- 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

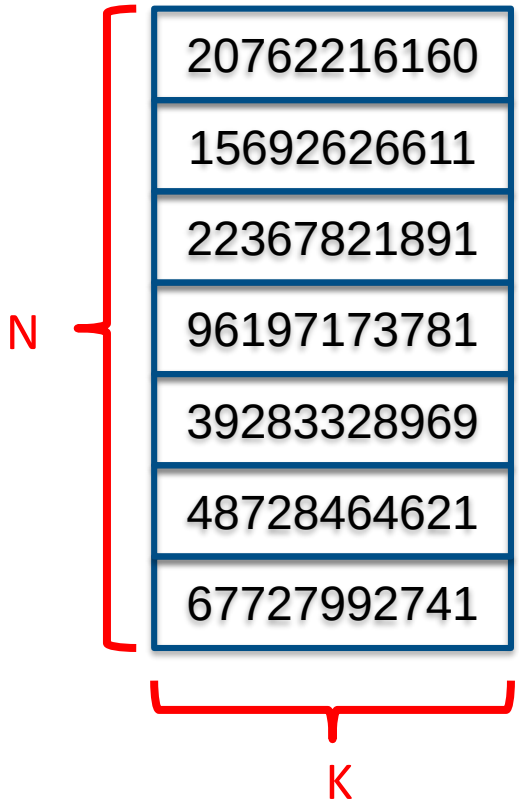
MONASH University

# Questions?

# Radix Sort

## TL;DR

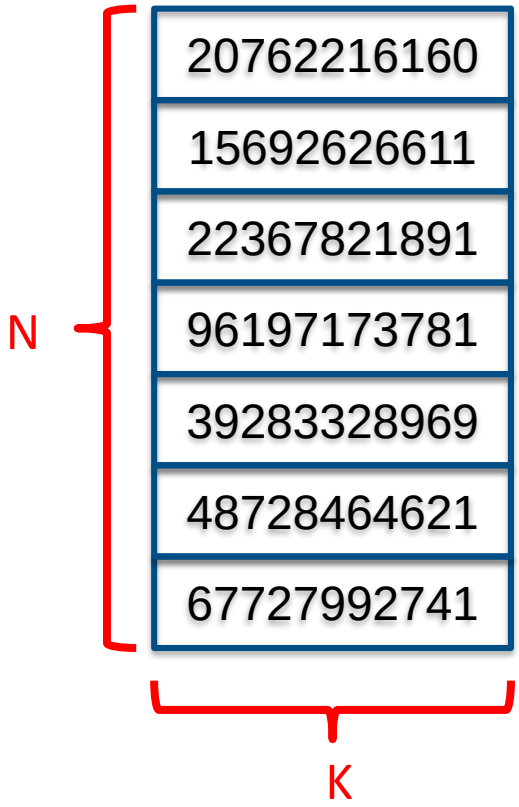
- So you know radix sort
- What have you notice?



# Radix Sort

## TL;DR

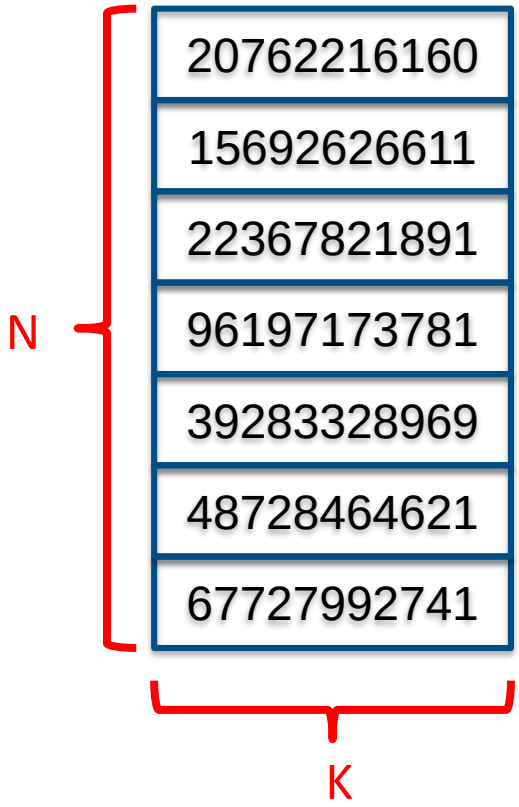
- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times



# Radix Sort

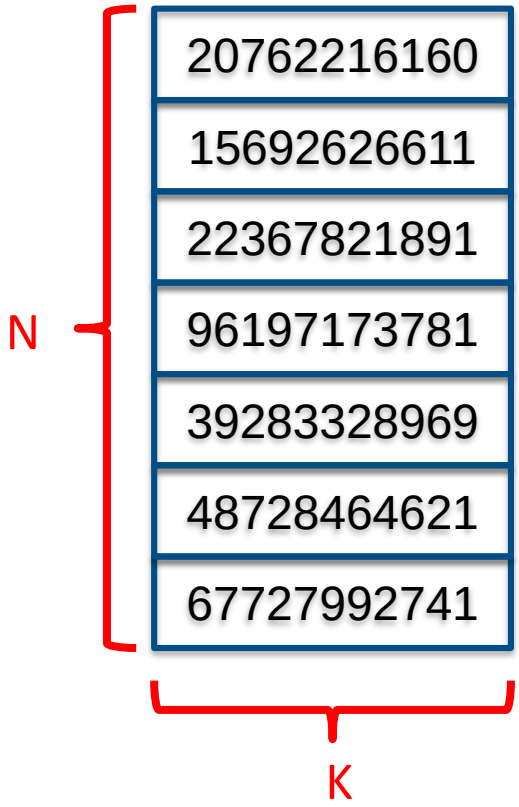
## TL;DR

- 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)



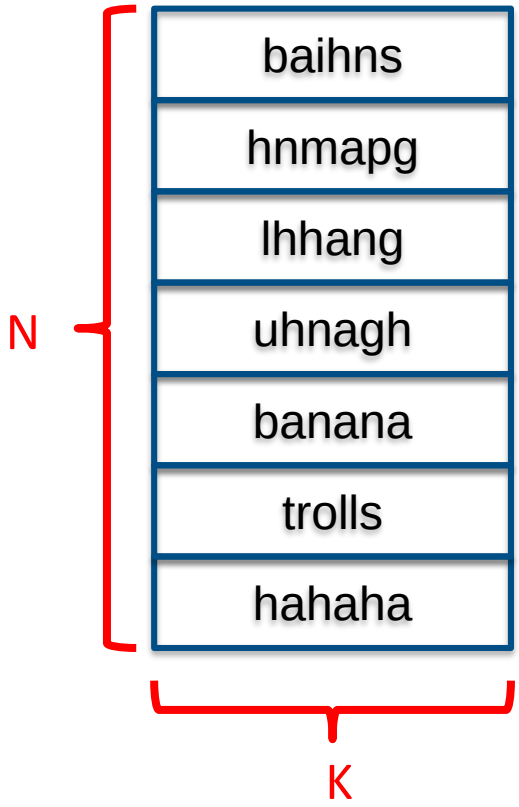
# Radix Sort
## TL;DR

- 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

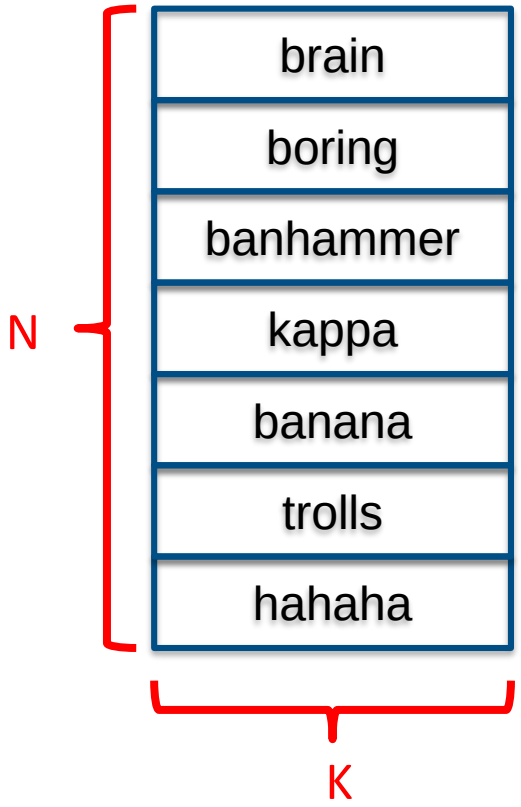
- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
    - Works well for characters as well
  - Usually least significant (right) to most significant (left)



# Radix Sort
## TL;DR

MONASH University

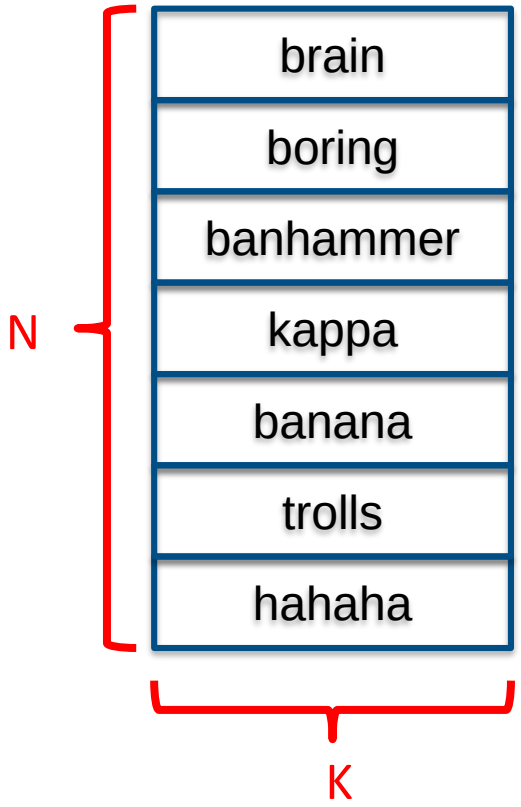
- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
    - Works well for characters as well
  - Usually least significant (right) to most significant (left)
- But what if they are not the same length?



# Radix Sort
## TL;DR

MONASH University

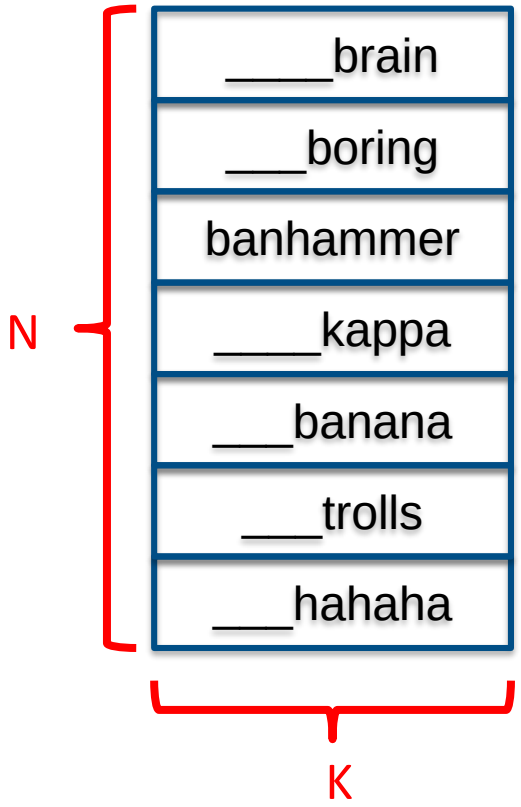
- So you know radix sort
- What have you notice?
  - It is counting sort really, done multiple times
    - We can reduce this by increasing the base
    - Works well for characters as well
  - Usually least significant (right) to most significant (left)
- But what if they are not the same length?
  -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



# Radix Sort
## TL;DR

MONASH University

- 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?



MONASH University

# Questions?

MONASH University

# Thank You