

SORTING IN LINEAR TIME WITH ASSUMPTIONS

COUNTING SORT: *assume* input: n integers within range of size k

$n=5$

9	6	8	9	8
---	---	---	---	---

$k: 1 \dots 9$



COUNTING SORT: assume input: n integers within range of size k

$n=5$

9	6	8	9	8
---	---	---	---	---

$k: 1 \dots 9$

or

$6 \dots 9$

$k=4$

COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

$k=4$

6	7	8	9
			1

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

$k=4$

6 7 8 9

			1
--	--	--	---

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

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

$k=4$

6 7 8 9

			1
--	--	--	---

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

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

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

$k=4$

6 7 8 9

			1
--	--	--	---

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

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

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

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

$k=4$

6 7 8 9

			1
--	--	--	---

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

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

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

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

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

$k=4$

6	7	8	9
			1

6	7	8	9
1			1

6	7	8	9
1		1	1

6	7	8	9
1		1	2

6	7	8	9
1		2	2

6 7 8 9

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

6				
---	--	--	--	--

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

$k=4$

6	7	8	9
			1

6	7	8	9
1			1

6	7	8	9
1		1	1

6	7	8	9
1		1	2

6	7	8	9
1		2	2

6 7 8 9

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

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

6				
---	--	--	--	--

COUNTING SORT: *assume* input: n integers within range of size k

Use an *array* with k indices matching range

↳ keep track of how many integers have each value

$n=5$

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

9	6	8	9	8
---	---	---	---	---

$k=4$

6	7	8	9
			1

6	7	8	9
1			1

6	7	8	9
1		1	1

6	7	8	9
1		1	2

6	7	8	9
1		2	2

6 7 8 9

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

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

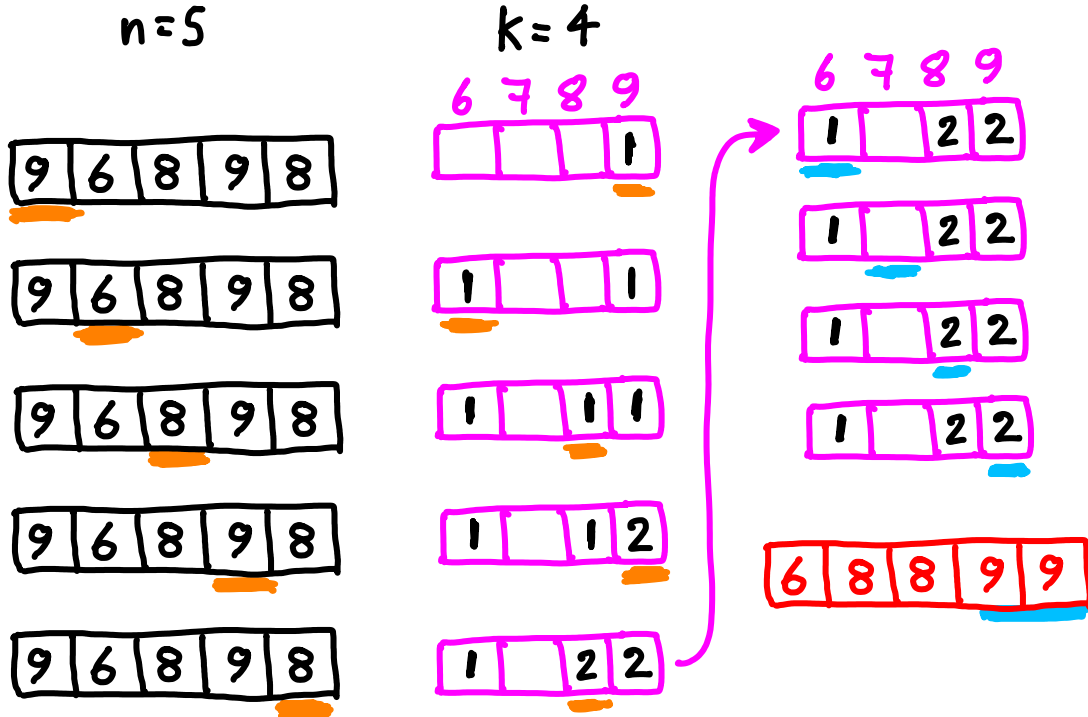
1		2	2
---	--	---	---

6	8	8		
---	---	---	--	--

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value



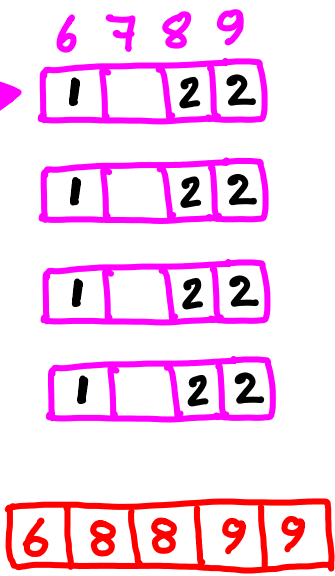
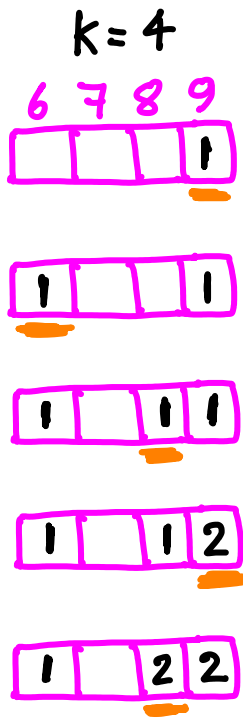
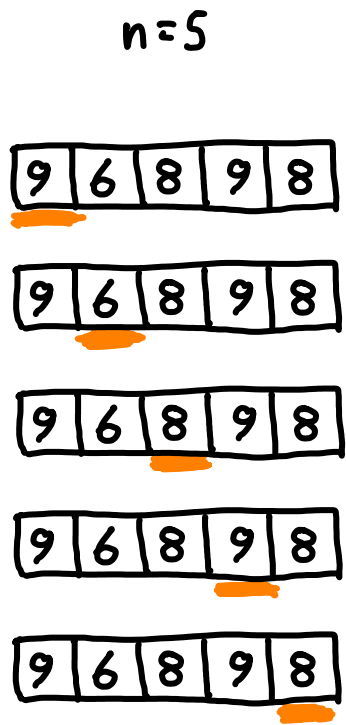
time: $\Theta(n) + \Theta(k)$

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

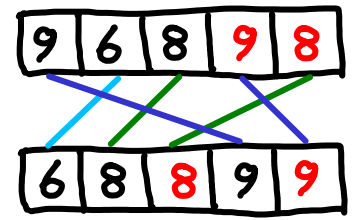
↳ keep track of how many integers have each value

stable sorting



not stable

time: $\Theta(n) + \Theta(k)$



COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value

$n=5$

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

$k=4$

6 7 8 9
[] [] [] [1]

[1] [] [] [1]

[1] [] [1] [1]

[1] [] [1] [2]

[1] [] [2] [2]

6 7 8 9

[1] [] [2] [2]

[1] [] [2] [2]

[1] [] [2] [2]

[1] [] [2] [2]

6 8 8 9 9

6 7 8 9

[1] [] [] []



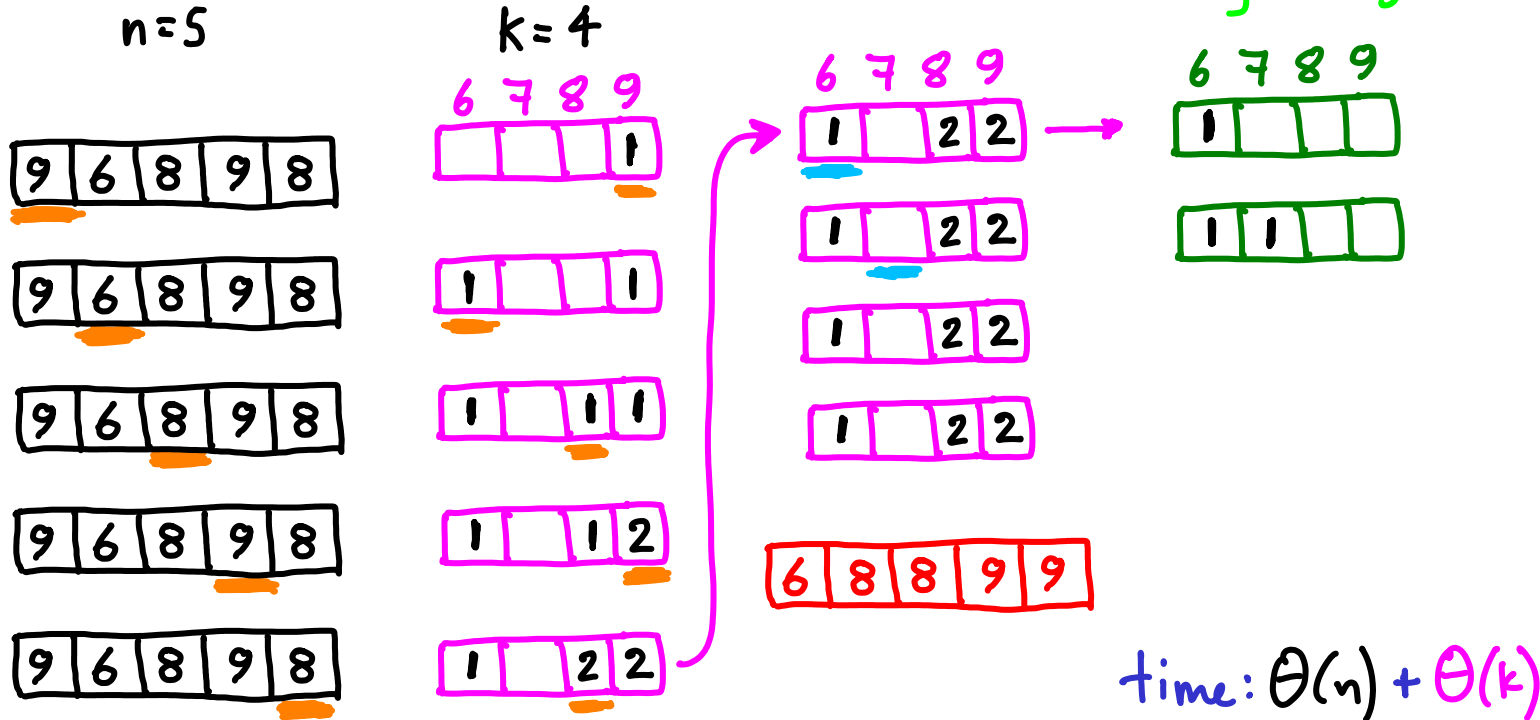
time: $\Theta(n) + \Theta(k)$

COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting
↳ how many integers are \leq each value



COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting
↳ how many integers are \leq each value

$n=5$

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

$k=4$

6 7 8 9
[] [] [] [1]

[1] [] [] [1]

[1] [] [1] [1]

[1] [] [1] [2]

[1] [] [2] [2]

6 7 8 9

[1] [] [2] [2]

[1] [] [2] [2]

[1] [] [2] [2]

[1] [] [2] [2]

6 8 8 9 9

6 7 8 9

[1] [] [] []

[1] [1] [] []

[1] [1] [3] []

time: $\Theta(n) + \Theta(k)$

COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting
↳ how many integers are \leq each value

$n=5$

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

9 6 8 9 8

$k=4$

6 7 8 9
[] [] [] [1]

[1] [] [] [1]

[1] [] [1] [1]

[1] [] [1] [2]

[1] [] [2] [2]

6 7 8 9

[1] [] [2] [2]

[1] [] [2] [2]

[1] [] [2] [2]

[1] [] [2] [2]

6 8 8 9 9

6 7 8 9

[1] [] [] []

[1] [1] [] []

[1] [1] [3] []

[1] [1] [3] [5]

time: $\Theta(n) + \Theta(k)$

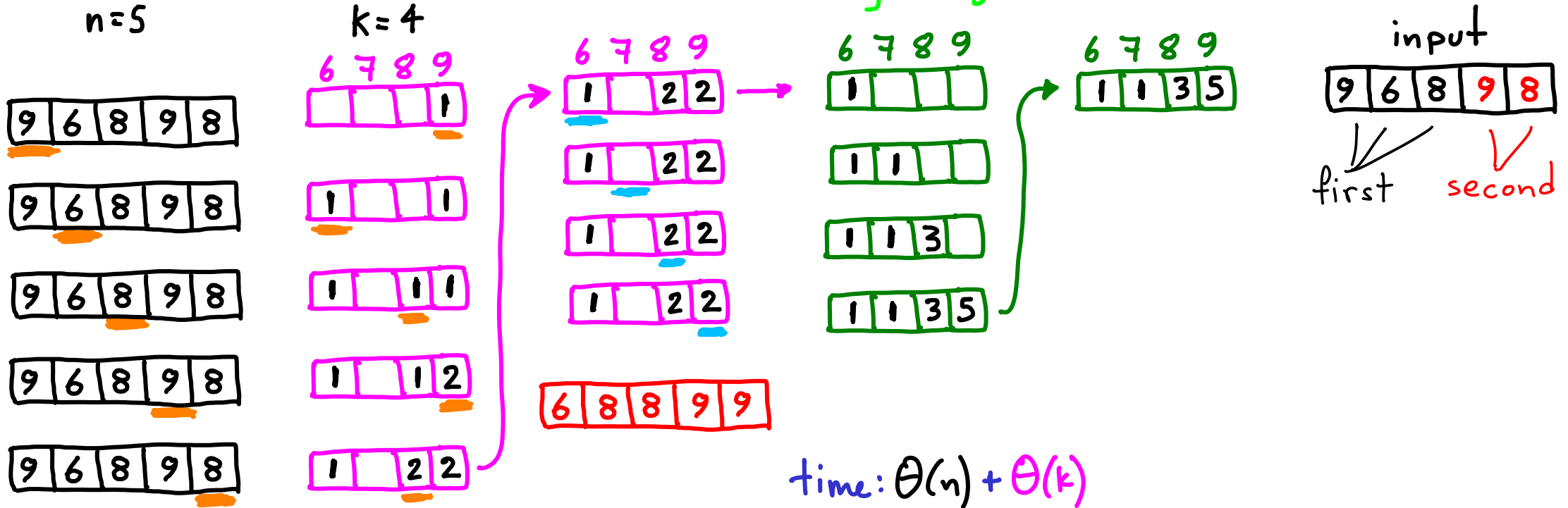
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



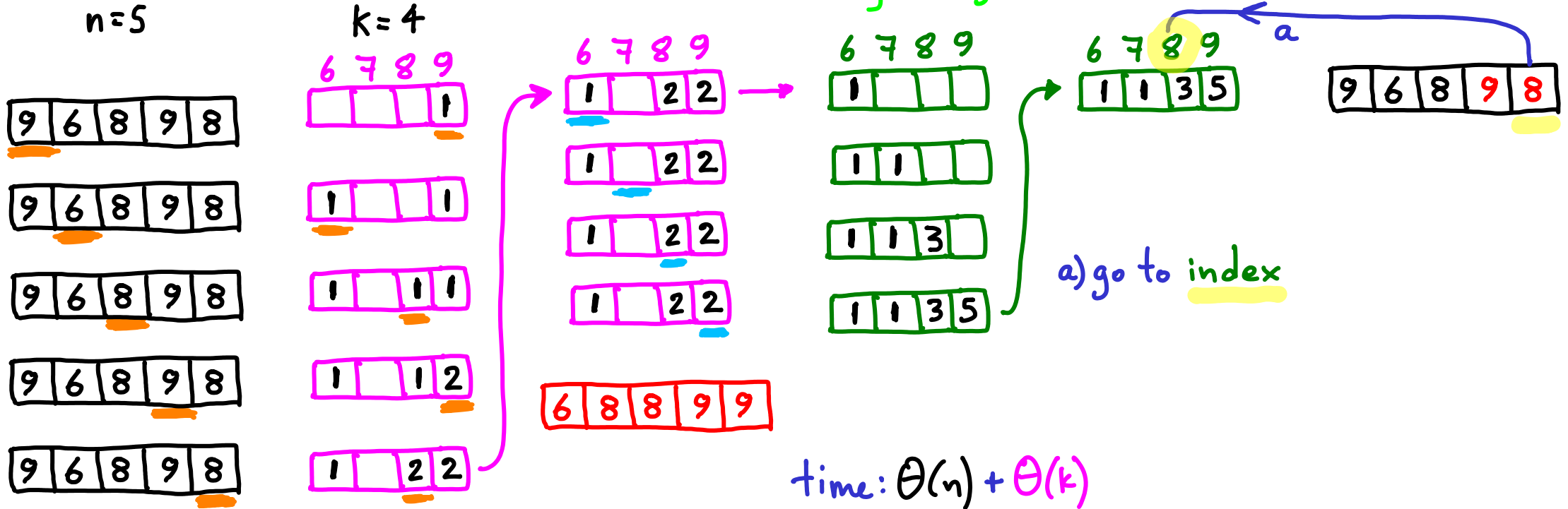
COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



COUNTING SORT: **assume** input: n integers within range of size k

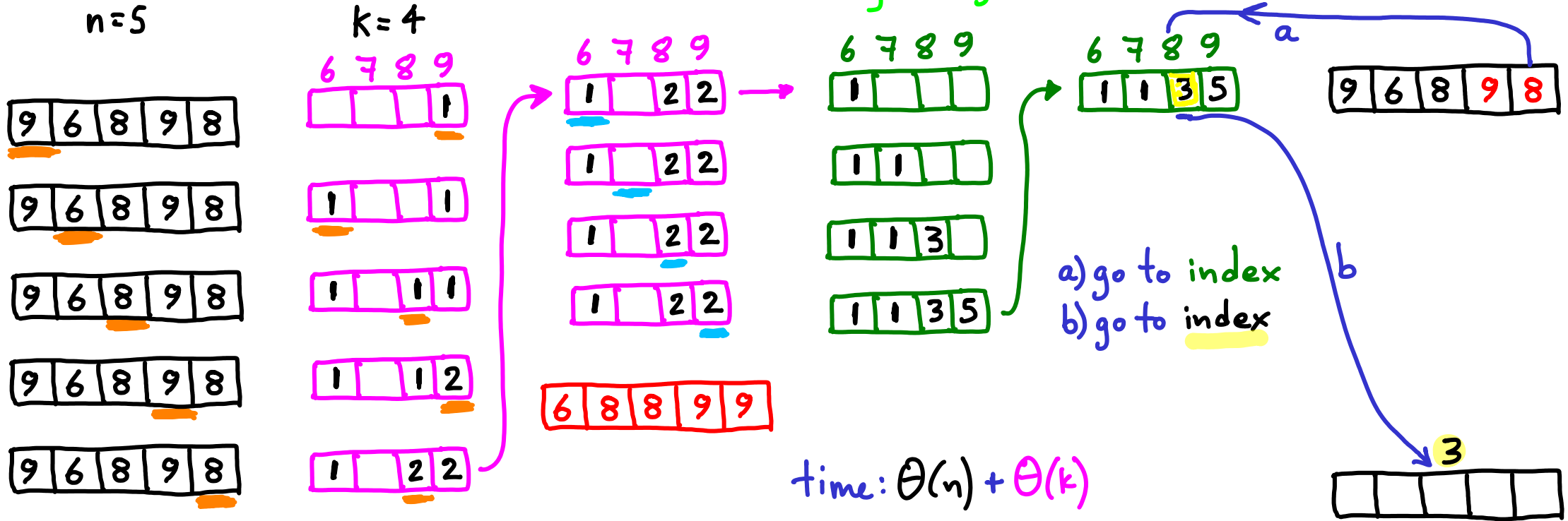
Use an **array** with k indices matching range

↳ keep track of how many integers have each value



Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



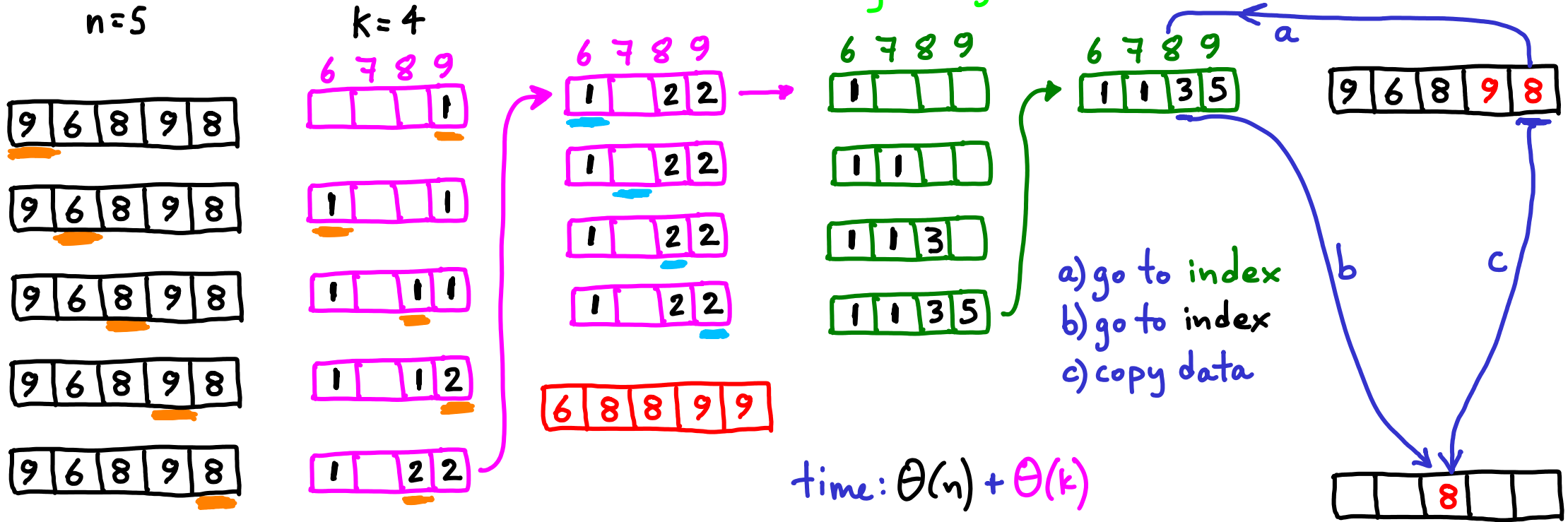
COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



COUNTING SORT: *assume* input: n integers within range of size k

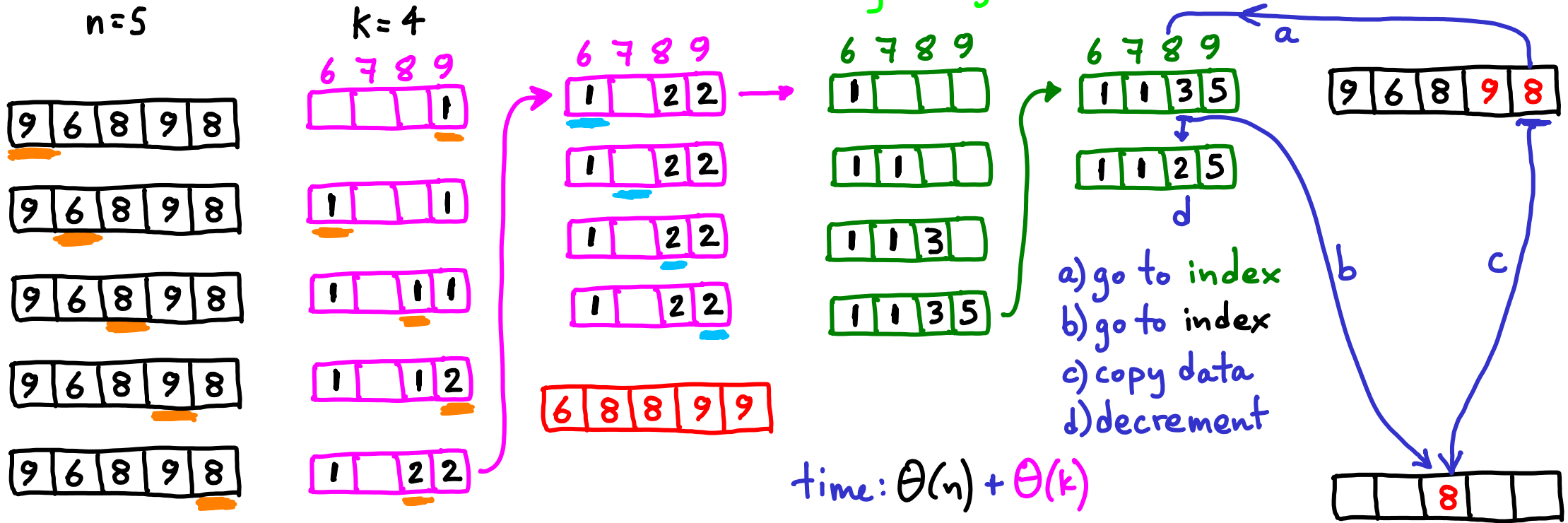
Use an *array* with k indices matching range

↳ keep track of how many integers have each value



Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



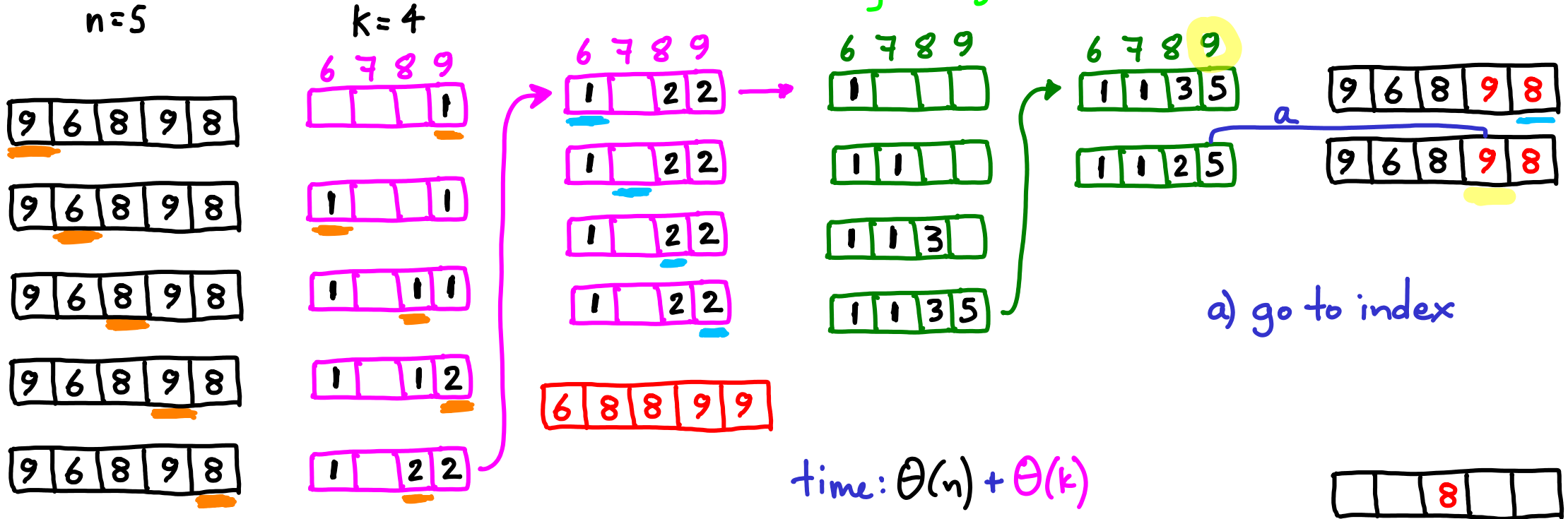
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



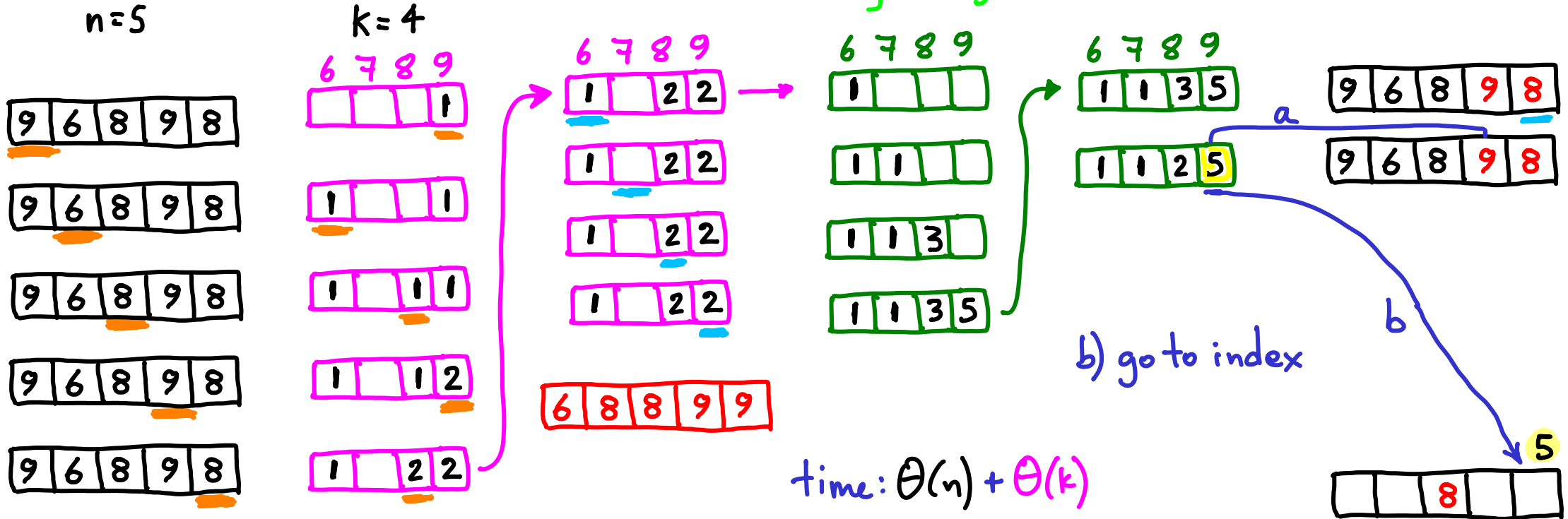
COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



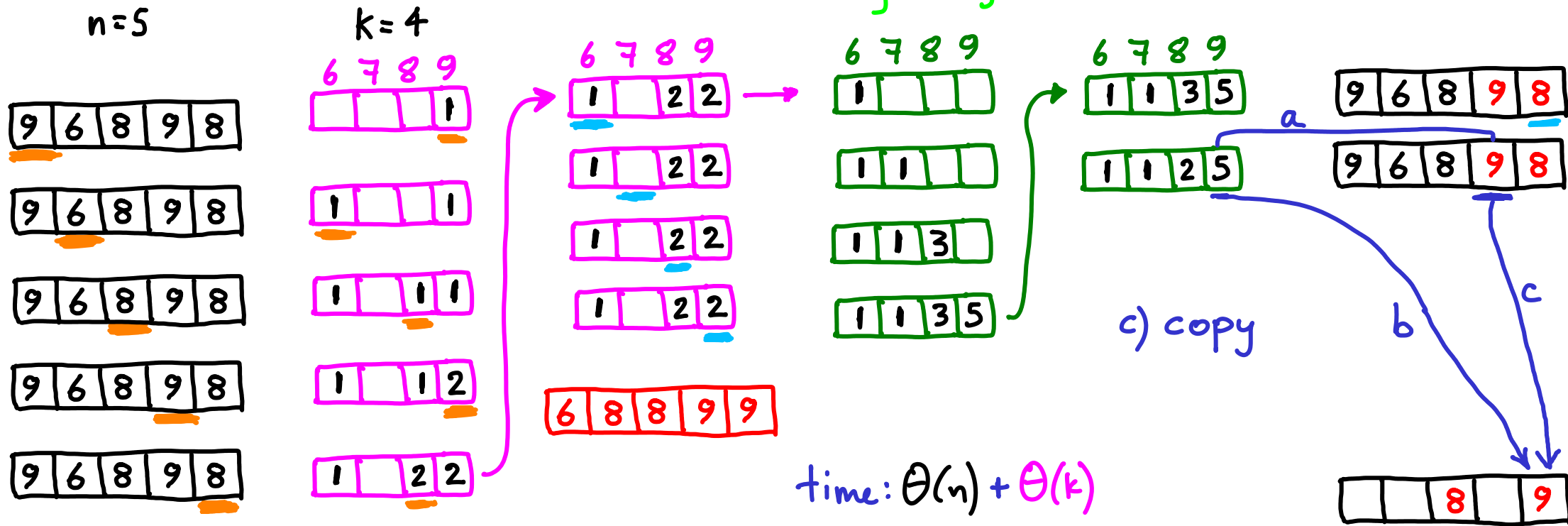
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



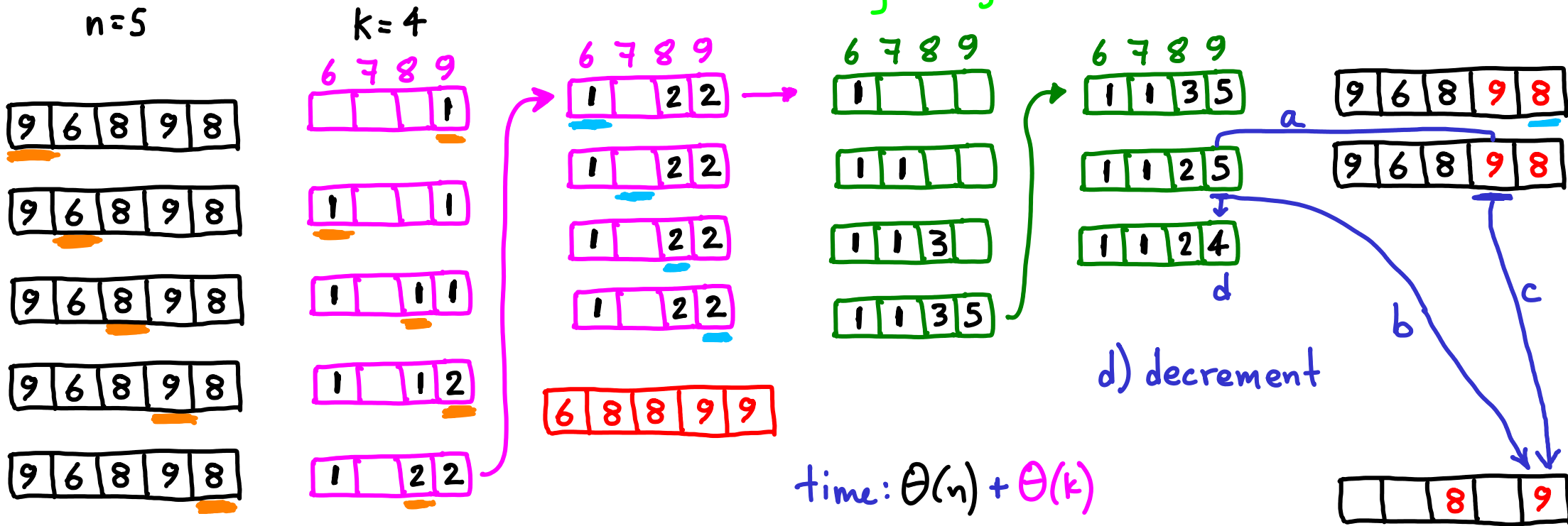
COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



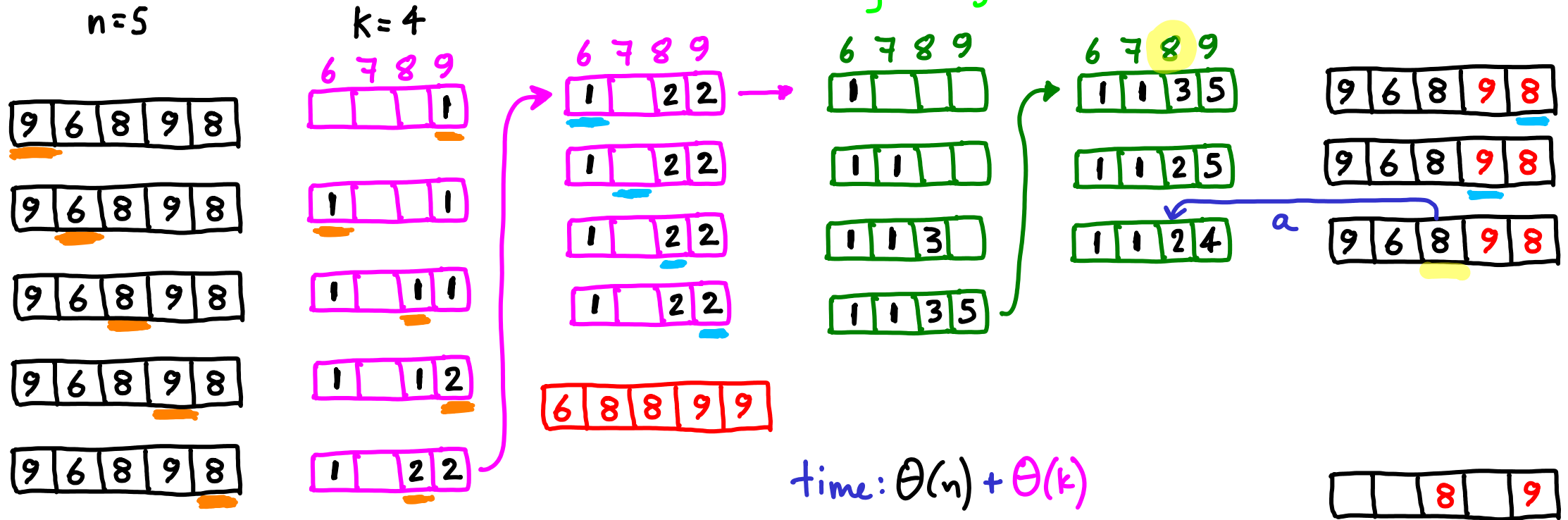
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



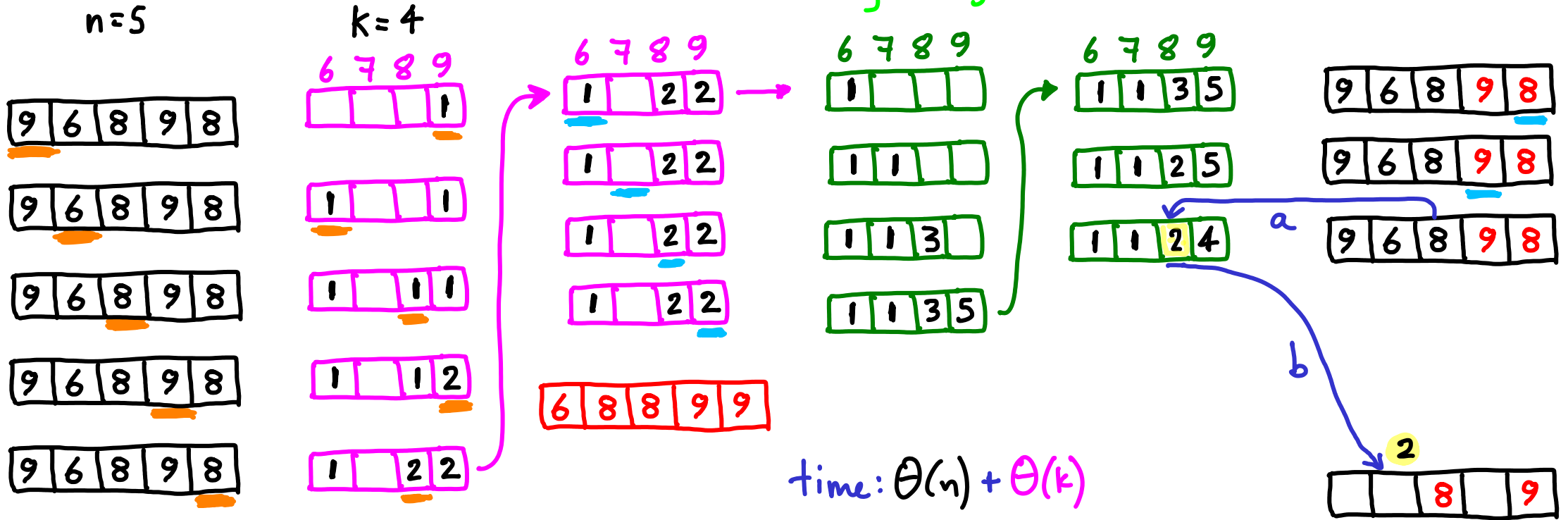
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



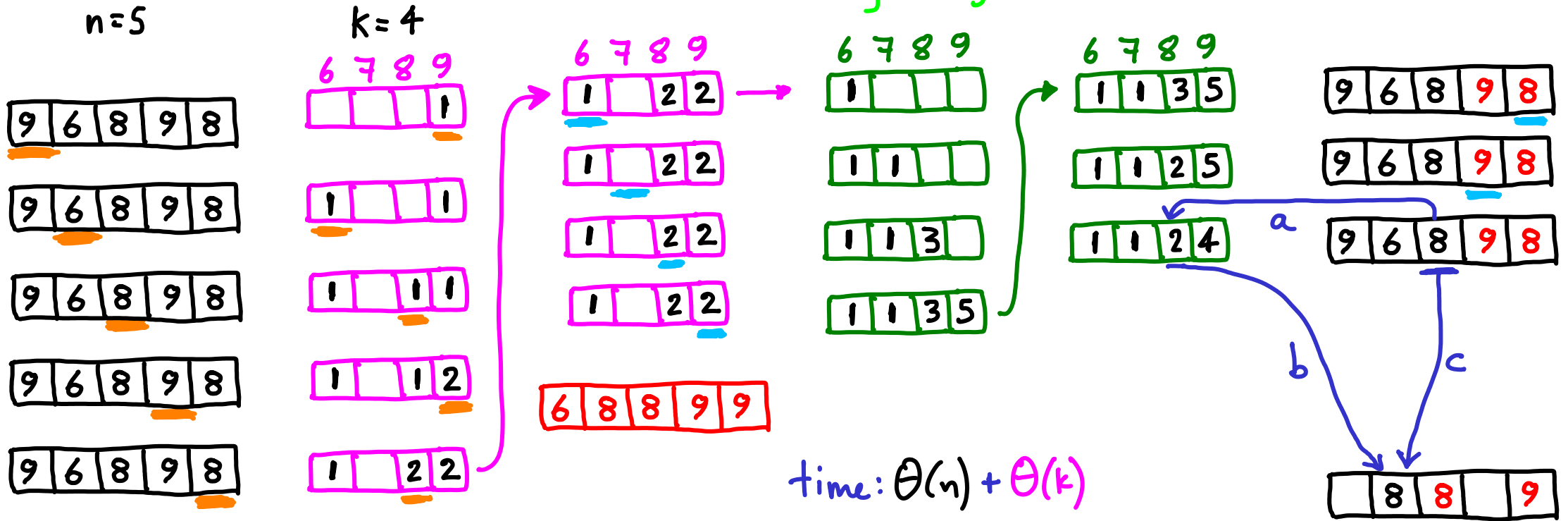
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



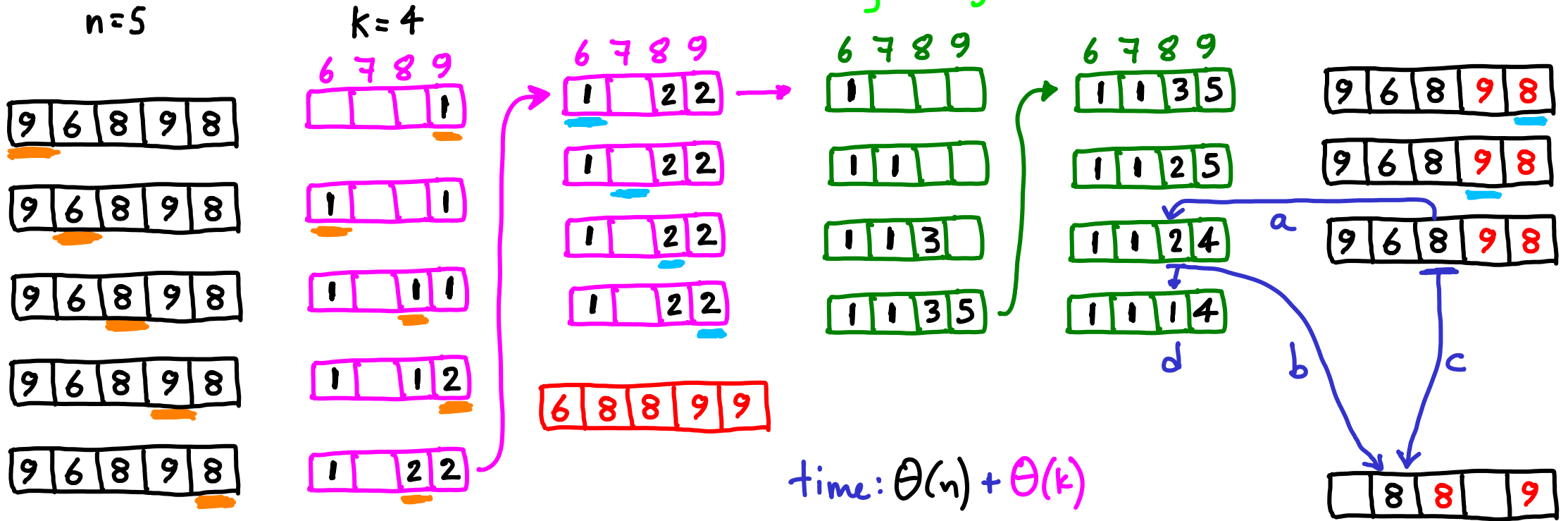
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



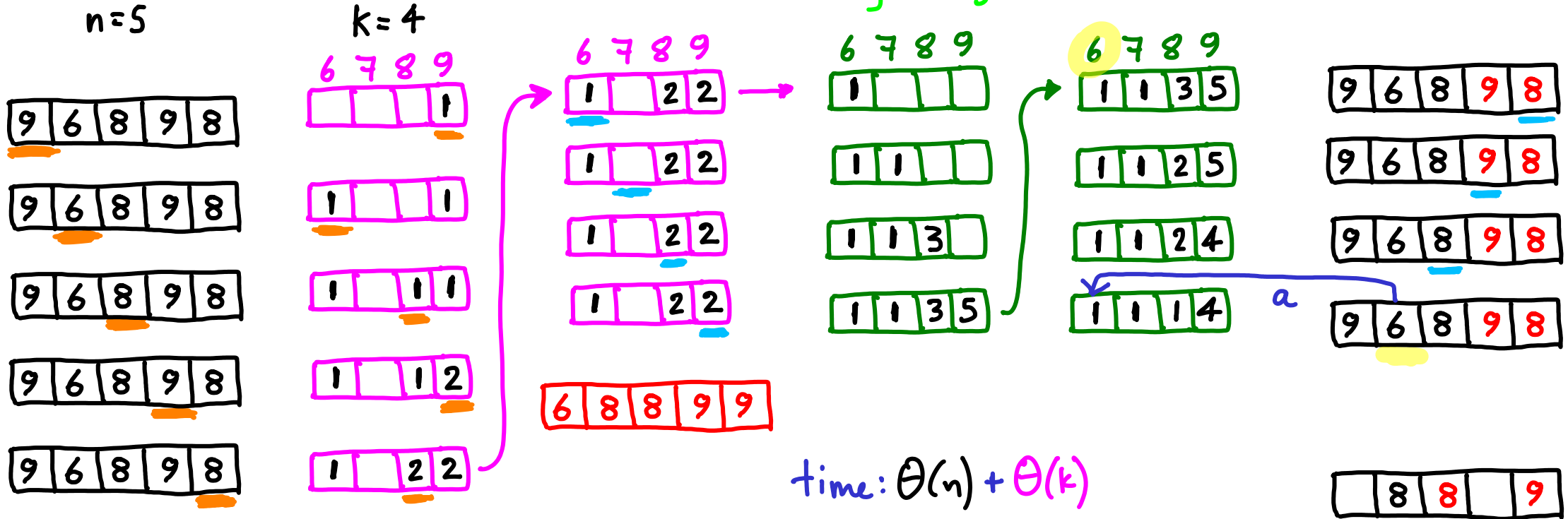
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



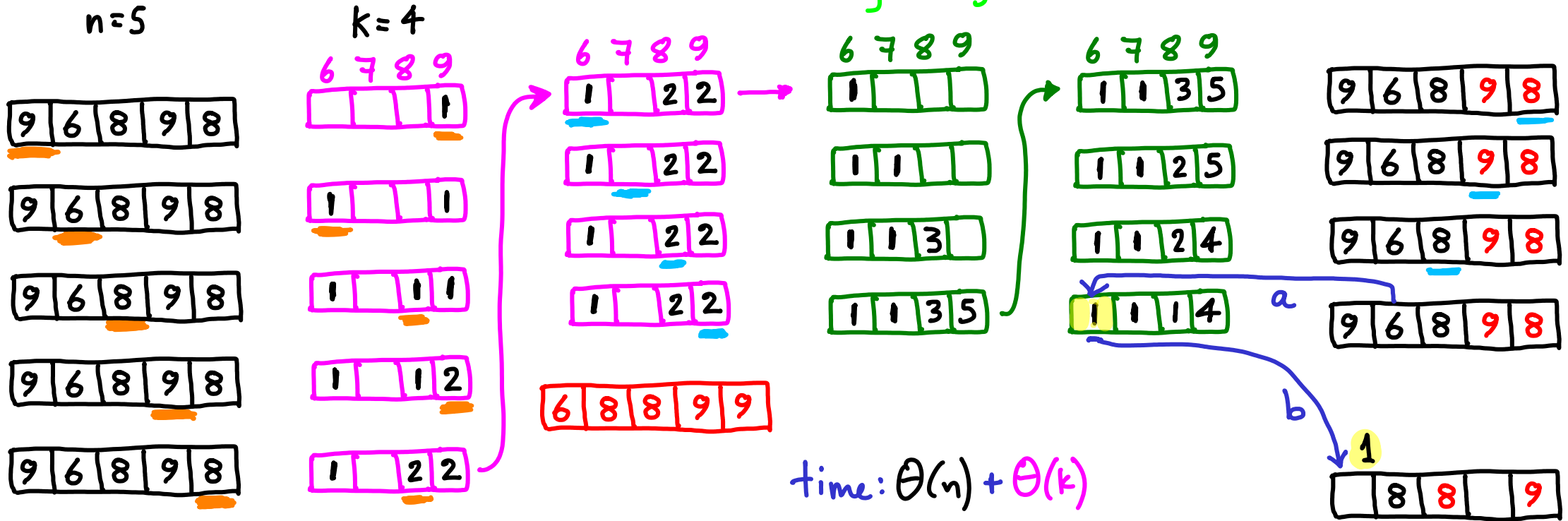
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



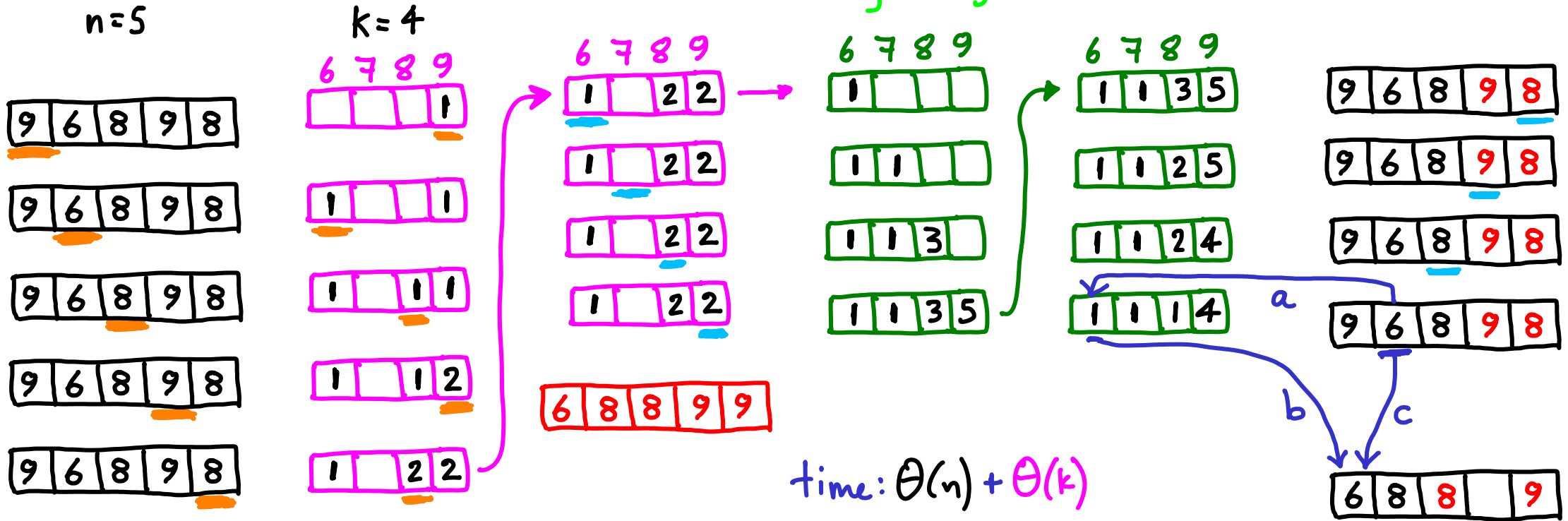
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



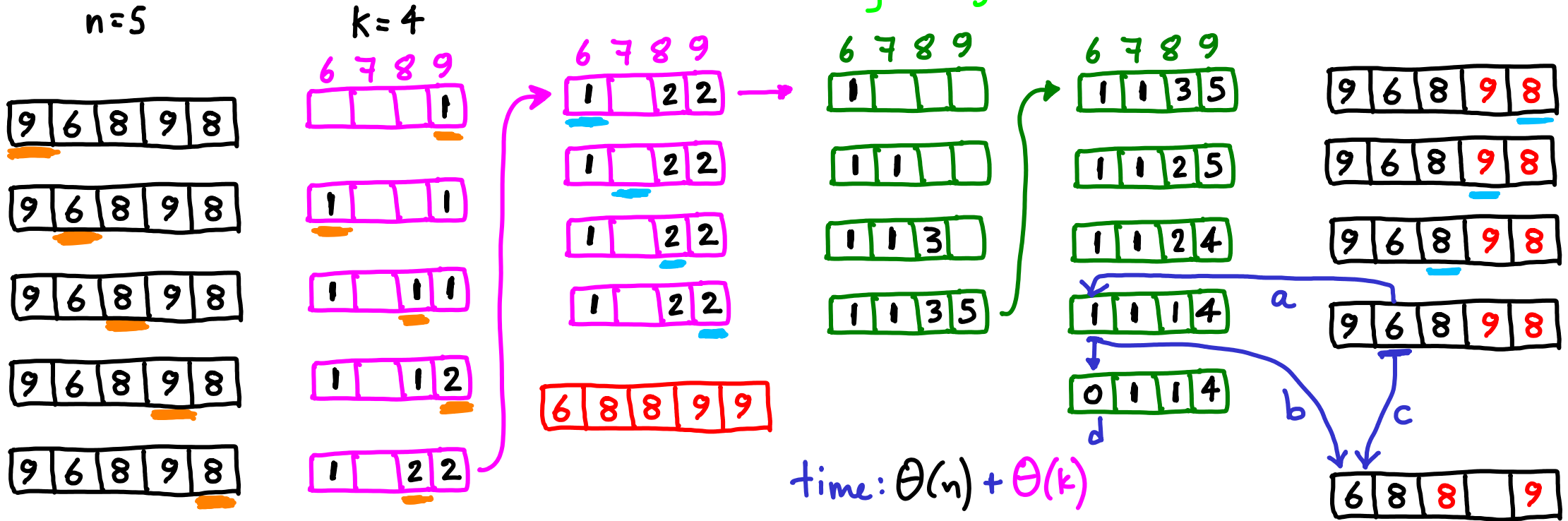
COUNTING SORT: assume input: n integers within range of size k

Use an array with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



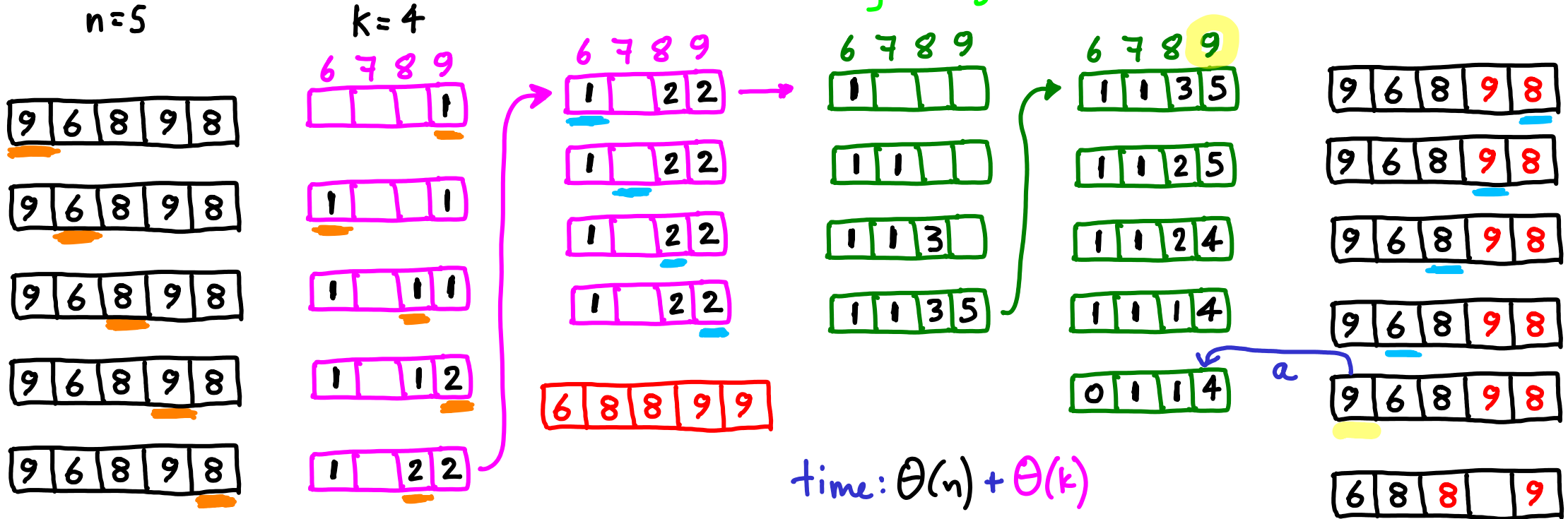
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



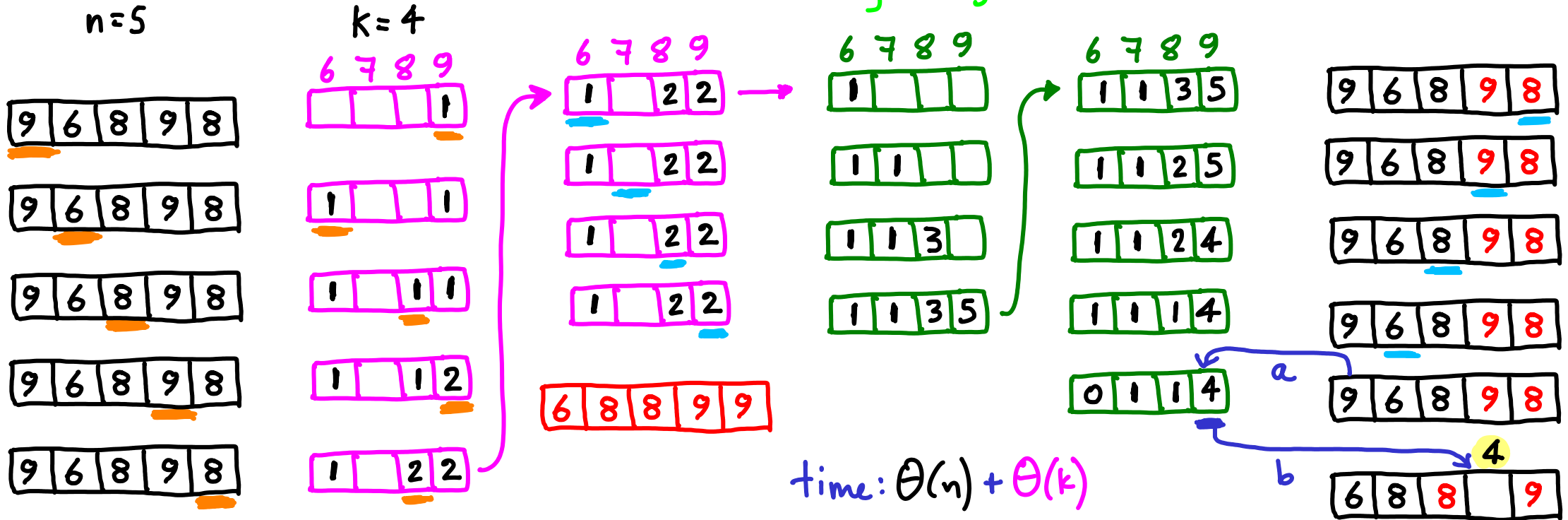
COUNTING SORT: *assume* input: n integers within range of size k

Use an *array* with k indices matching range

↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value



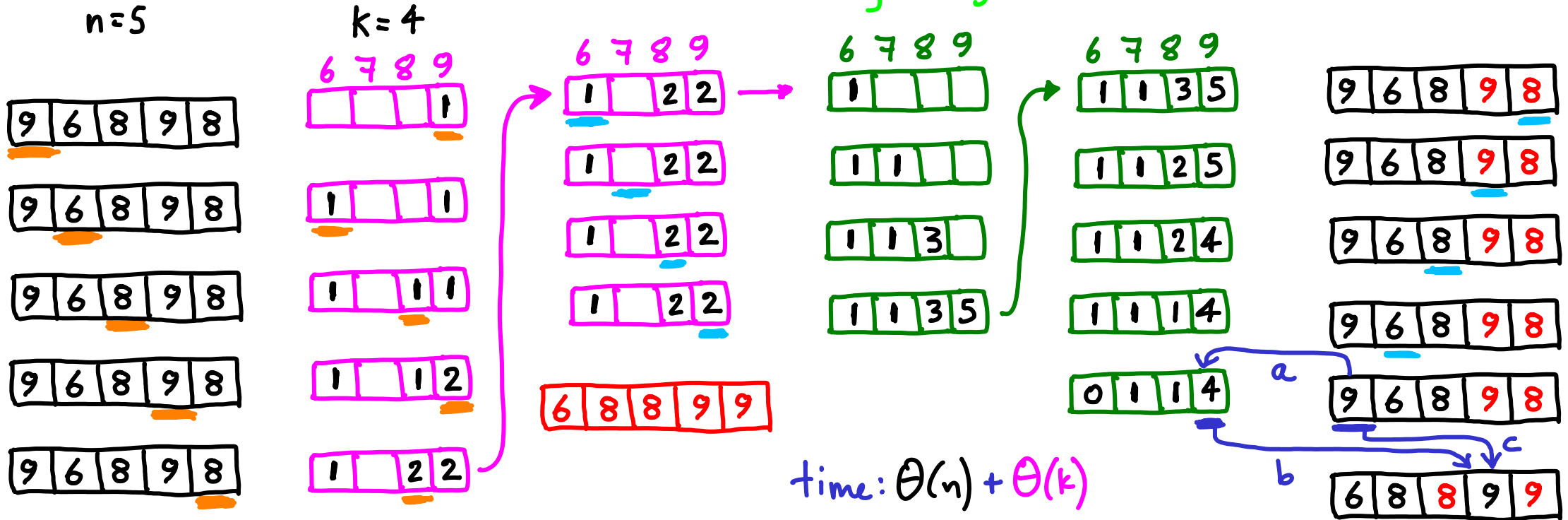
COUNTING SORT: **assume** input: n integers within range of size k

Use an **array** with k indices matching range

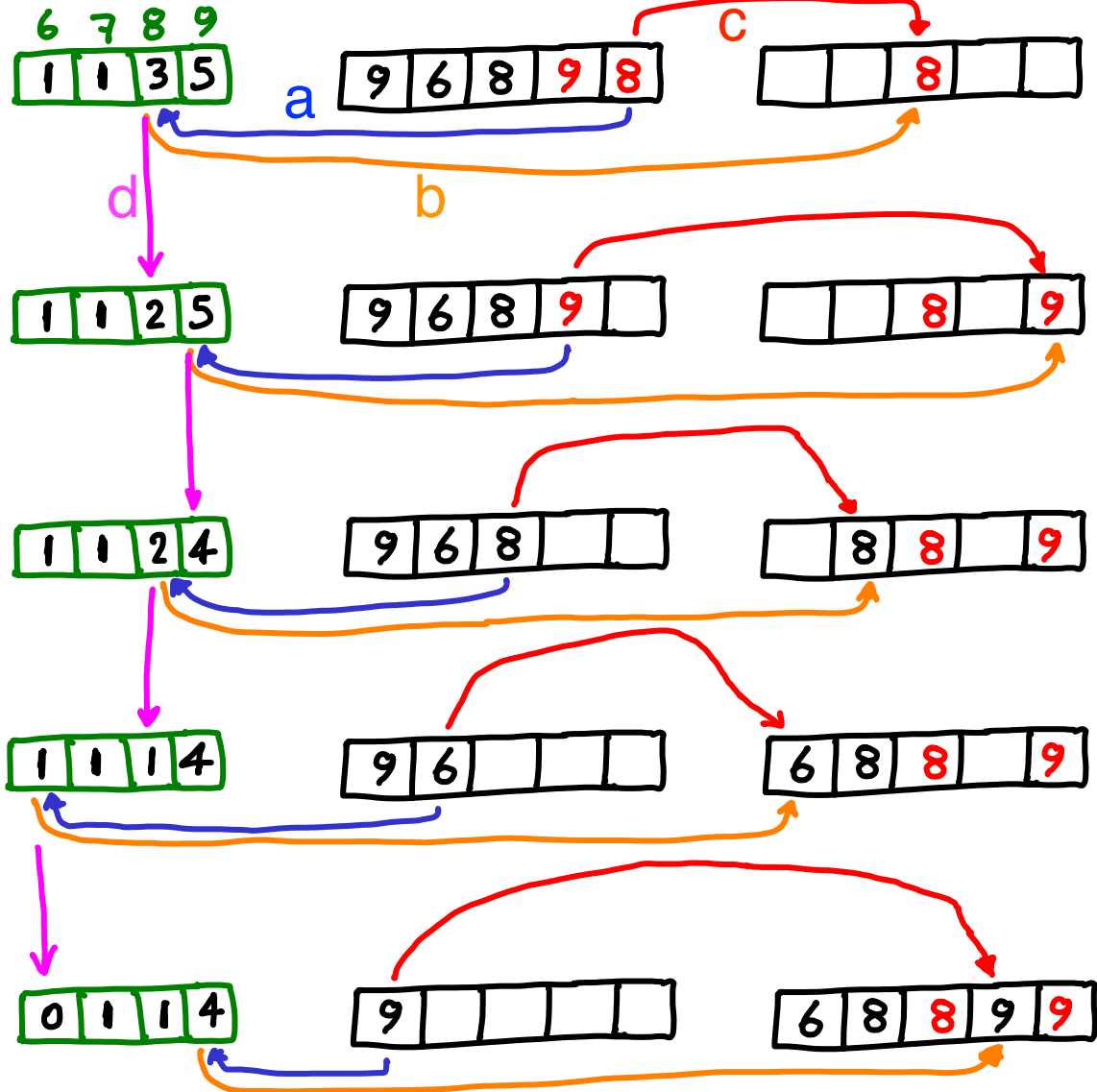
↳ keep track of how many integers have each value

Use a second array of size k to get stable sorting

↳ how many integers are \leq each value

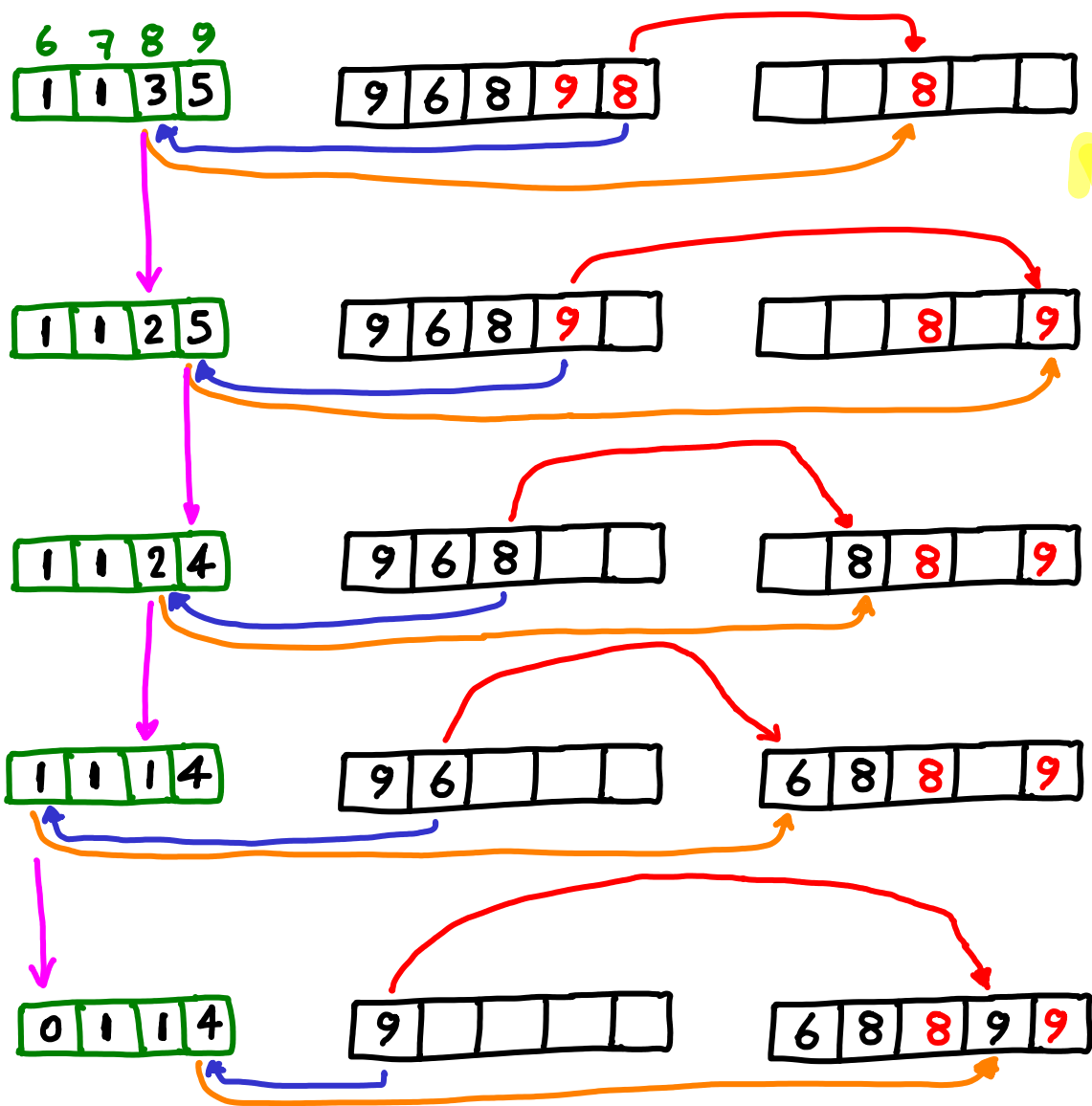


STABLE COUNTING SORT



Recap

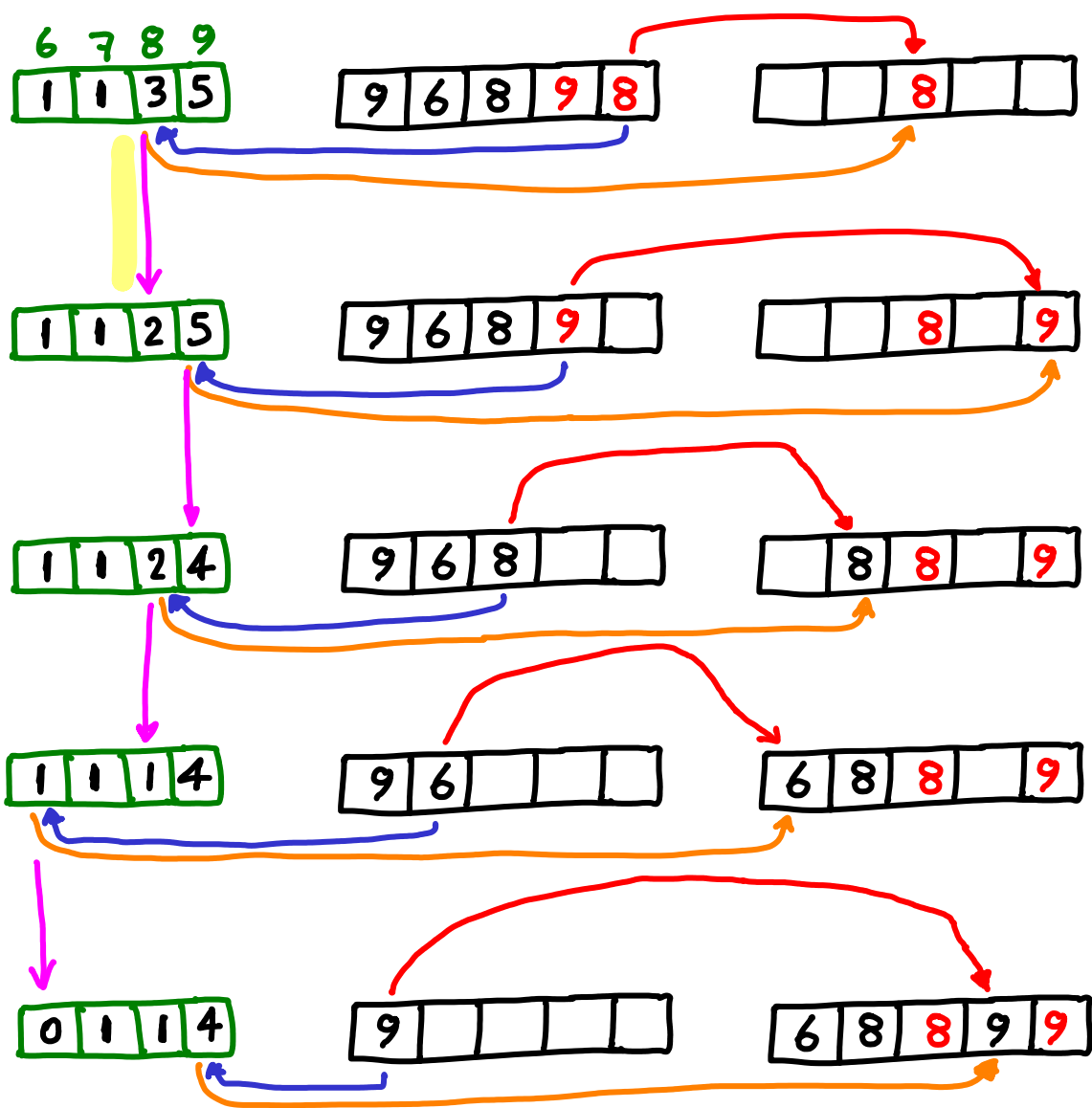
STABLE COUNTING SORT



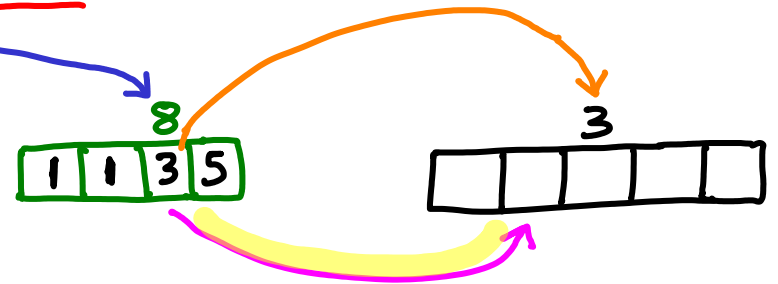
the first time we point to the count array at index i ...
 ... it tells us where the last element with value i must go



STABLE COUNTING SORT

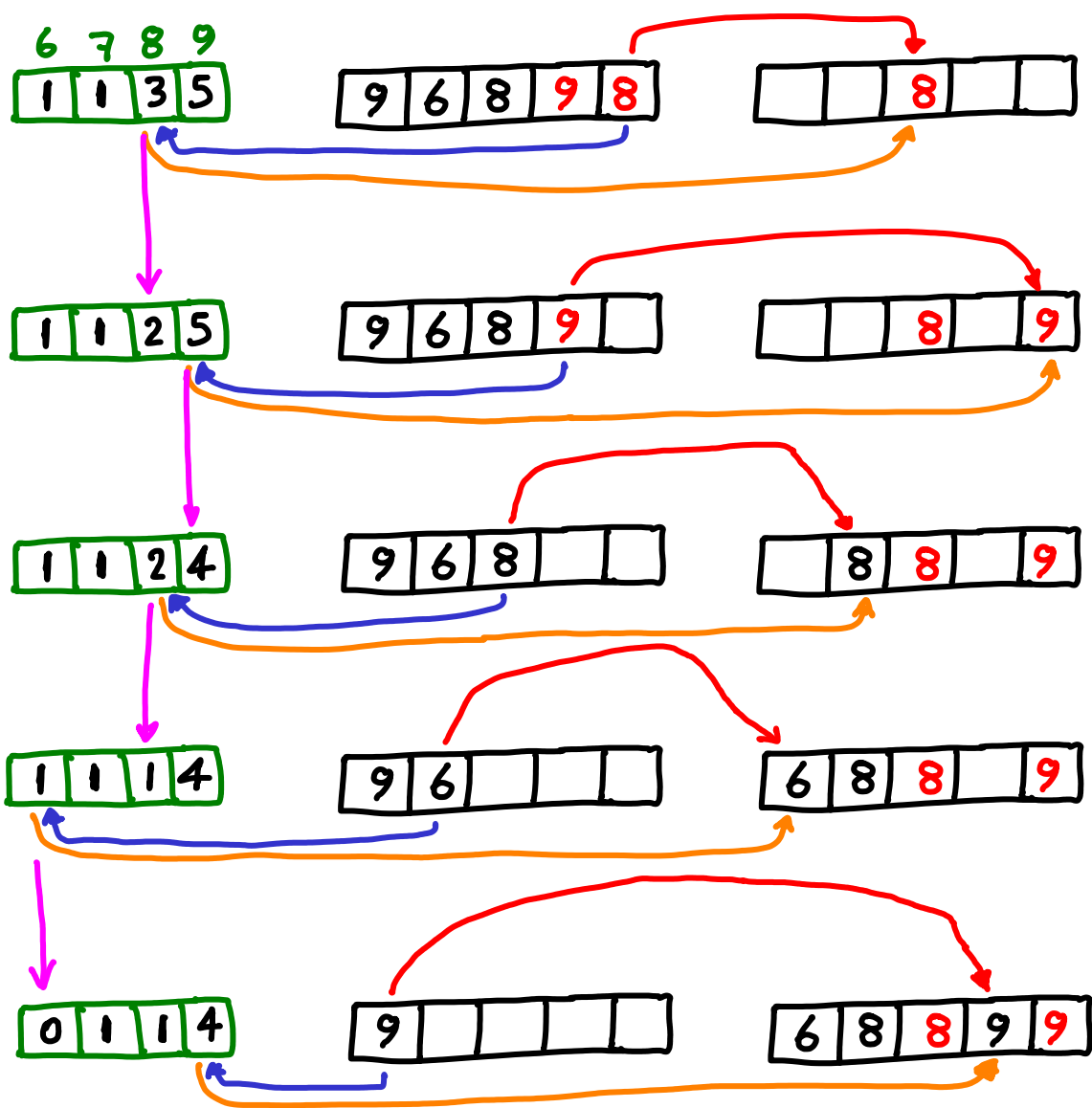


the first time we point to the count array at index i ...
 ... it tells us where the last element with value i must go

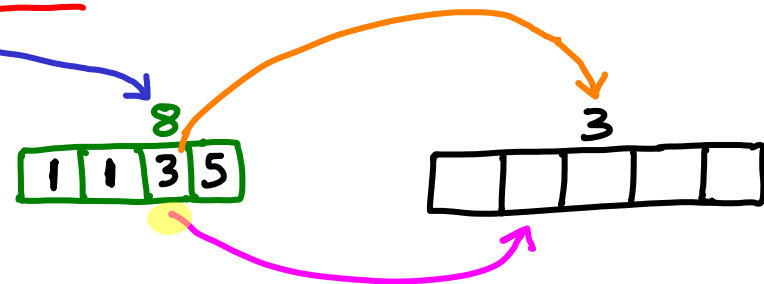


Then we decrement so the next element with value i (to the left in input array) will have its correct empty place.

STABLE COUNTING SORT



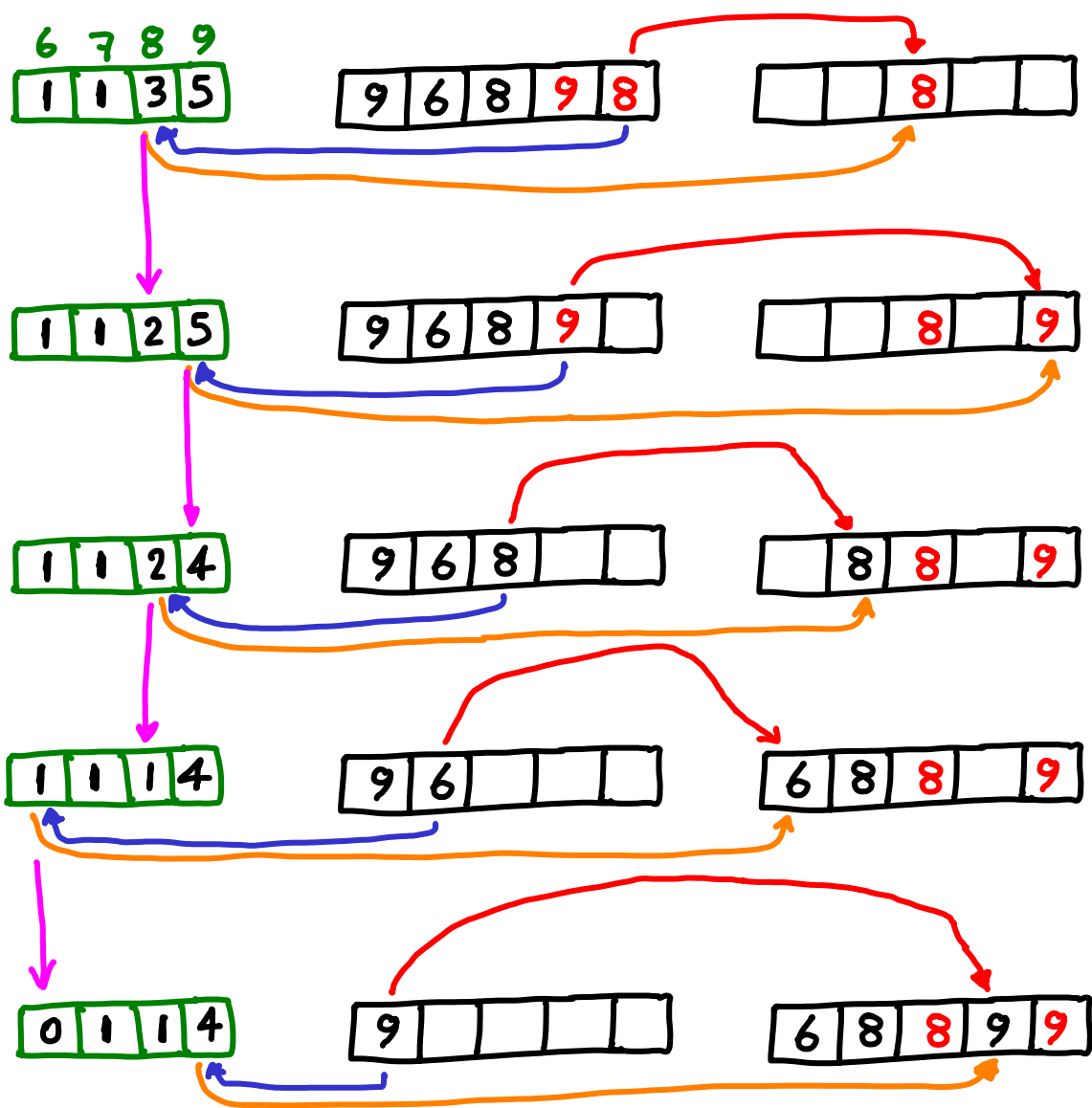
the first time we point to the count array at index i ...
 ... it tells us where the last element with value i must go



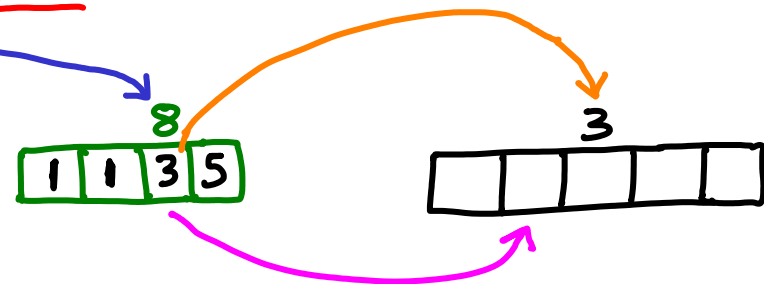
Then we decrement so the next element with value i (to the left in input array) will have its correct empty place.

We will only decrement once per element with value i

STABLE COUNTING SORT



the first time we point to the count array at index i ...
 ... it tells us where the last element with value i must go



Then we decrement so the next element with value i (to the left in input array) will have its correct empty place.

We will only decrement once per element with value i , so we maintain $\#$ in index $i > \max \#$ in index $i-1$
 no conflicts

There is also a similar variant in which we use a helper array to record the leftmost target position for each value. (Omitted here)

A different version follows.

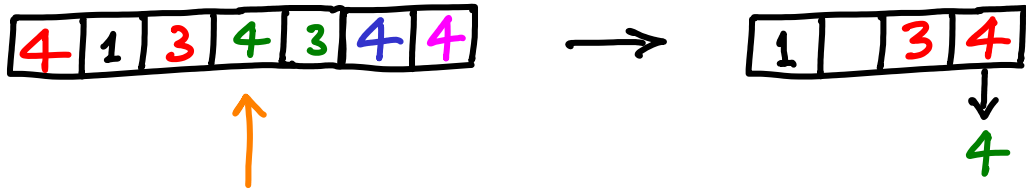




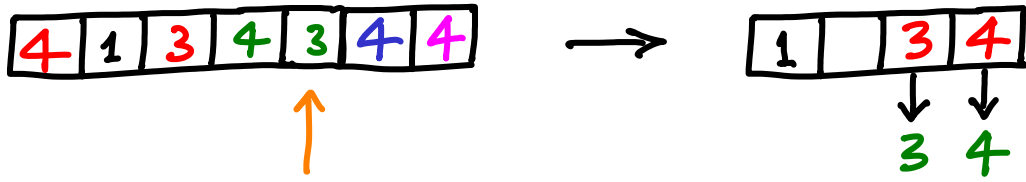
4	1	3	4	3	4	4
---	---	---	---	---	---	---



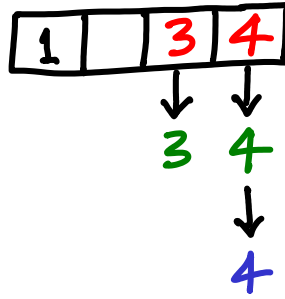
1		3	4
---	--	---	---



linked list

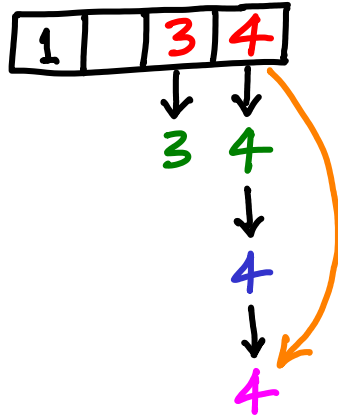
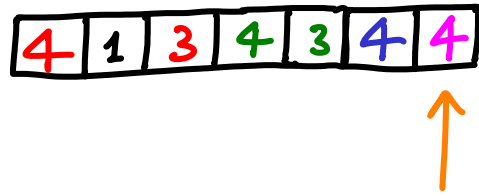


linked list



time?

linked list



need a pointer
to last element
in linked list
otherwise $O(n^2)$

trivial
→



$O(n)$ for all
linked lists
+ $O(k)$ just to
create the
counter array

RADIX SORT

RADIX SORT

n : number of elements

1073	284	5	8261	2714	382
------	-----	---	------	------	-----

$$n=6$$

RADIX SORT

n : number of elements

l : (max) length of each element

1073	284	5	8261	2714	382
------	-----	---	------	------	-----

1073	0284	0005	8261	2714	0382
------	------	------	------	------	------

$l=4$

$n=6$

RADIX SORT

n : number of elements

l : (max) length of each element

r : radix (#symbols available at each digit) e.g., binary, decimal, hex

1073	284	5	8261	2714	382
------	-----	---	------	------	-----

1073	0284	0005	8261	2714	0382
------	------	------	------	------	------

$r = 9$
(0...8)

$n = 6$

$l = 4$

There is a version of Radix sort
that does what most of us find intuitive

There is a version of Radix sort
that does what most of us find intuitive:

Group all elements by most significant digit
& recurse within each group.

There is a version of Radix sort
that does what most of us find intuitive:

Group all elements by most significant digit
& recurse within each group.

There is another way that is easier to implement.

RADIX SORT

3 2 9

4 5 7

6 5 7

8 3 9

4 3 6

7 2 0

3 5 5

$$n = 7$$

$$l = 3$$

$$r = 10$$

RADIX SORT

uses the least significant digit.
↓
usually

3 2 9

4 5 7

6 5 7

8 3 9

4 3 6

7 2 0

3 5 5

RADIX SORT

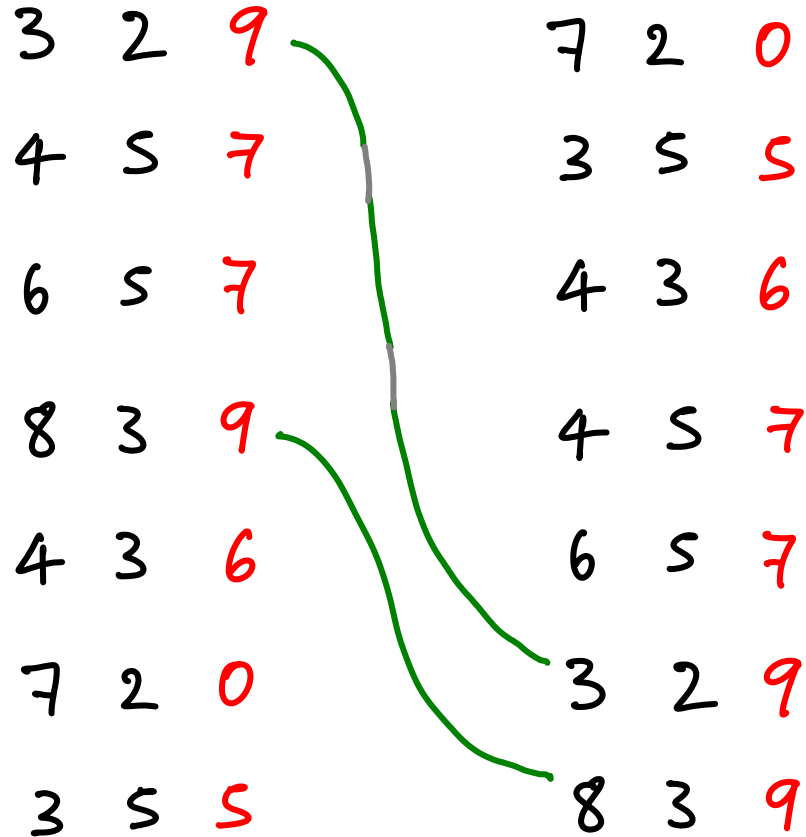
uses the least significant digit.

3	2	9		7	2	0
4	5	7		3	5	5
6	5	7		4	3	6
8	3	9	⇒	4	5	7
4	3	6		6	5	7
7	2	0		3	2	9
3	5	5		8	3	9

RADIX SORT

uses the least significant digit.

don't cross the streams

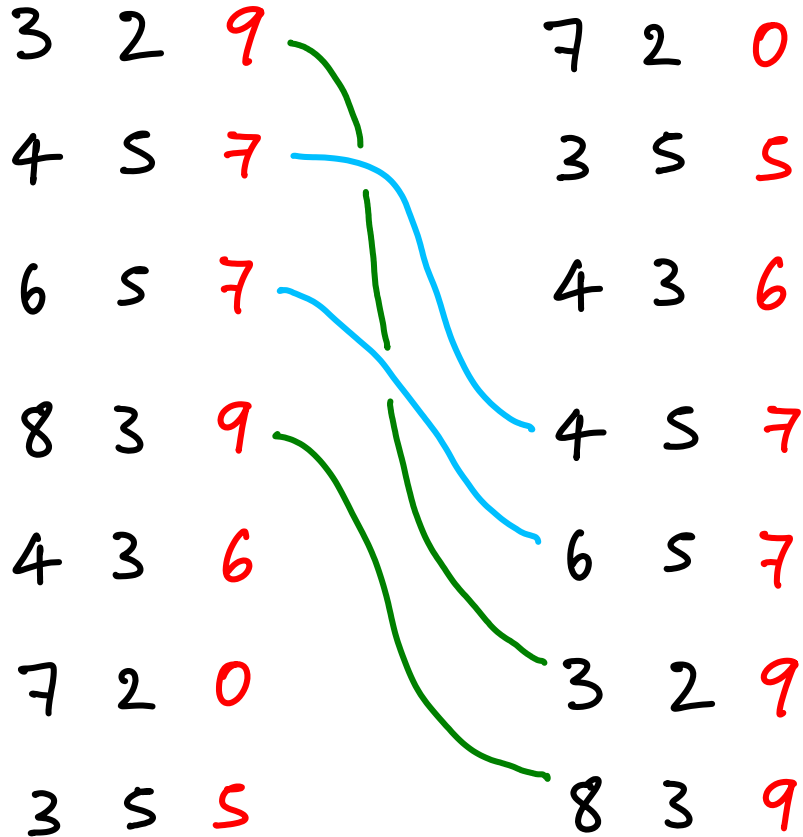


RADIX SORT

uses the least significant digit.

use stable counting sort

→ $\Theta(n+r)$



RADIX SORT

uses the least significant digit.

iteration 2

3 2 9

4 5 7

6 5 7

8 3 9

4 3 6

7 2 0

3 5 5

7 2 0

3 5 5

4 3 6

4 5 7

6 5 7

3 2 9

8 3 9

7 2 0

3 2 9

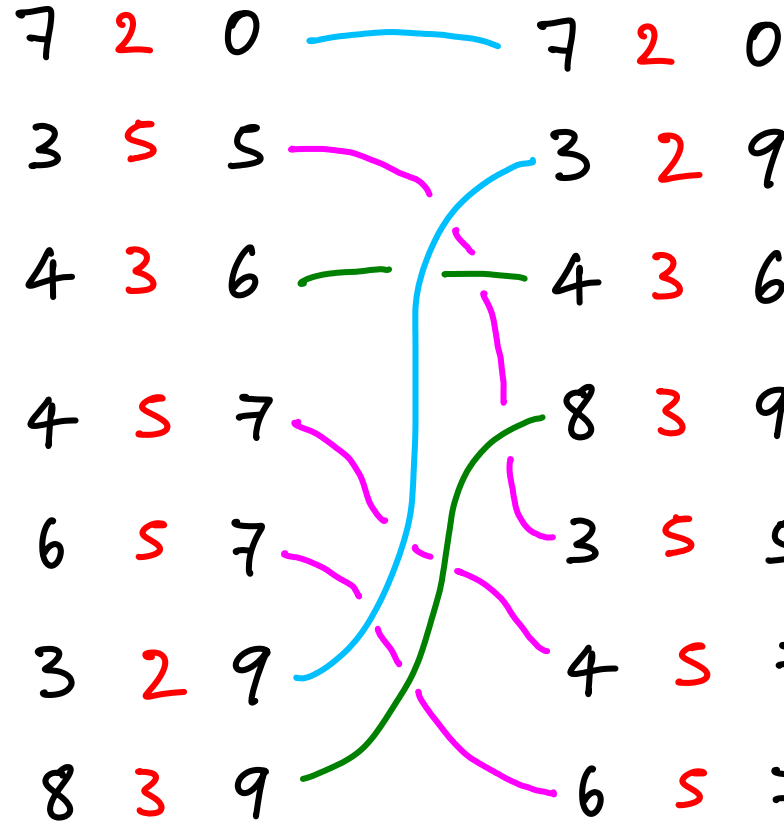
4 3 6

8 3 9

3 5 5

4 5 7

6 5 7



RADIX SORT

uses the least significant digit.

3 2 9

4 5 7

6 5 7

8 3 9

4 3 6

7 2 0

3 5 5

7 2 0

3 5 5

4 3 6

4 5 7

6 5 7

3 2 9

8 3 9

iteration 3

7 2 0

3 2 9

4 3 6

8 3 9

3 5 5

4 5 7

6 5 7

3 2 9

3 5 5

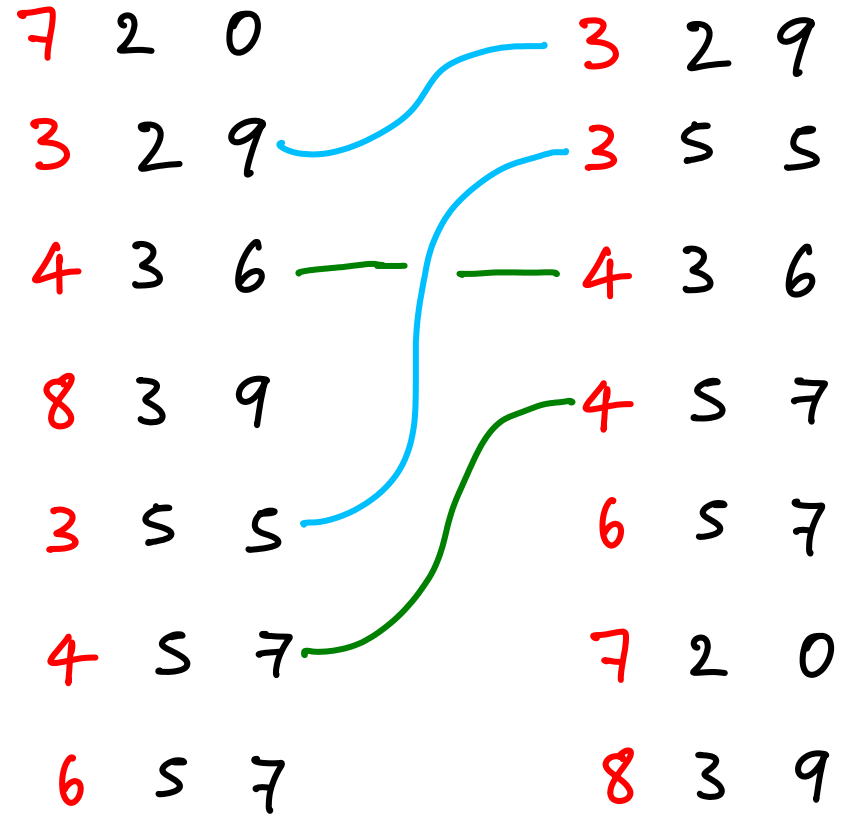
4 3 6

4 5 7

6 5 7

7 2 0

8 3 9



RADIX SORT

Time = ?

uses the least significant digit.

3 2 9

4 5 7

6 5 7

8 3 9

4 3 6

7 2 0

3 5 5

7 2 0

3 5 5

4 3 6

4 5 7

6 5 7

3 2 9

8 3 9

7 2 0

3 2 9

4 3 6

8 3 9

3 5 5

4 5 7

6 5 7

3 2 9

3 5 5

4 3 6

4 5 7

6 5 7

7 2 0

8 3 9

RADIX SORT

$$\Theta(l \cdot (n+r))$$

uses the least significant digit.

3 2 9

4 5 7

6 5 7

8 3 9

4 3 6

7 2 0

3 5 5

7 2 0

3 5 5

4 3 6

4 5 7

6 5 7

3 2 9

8 3 9

7 2 0

3 2 9

4 3 6

8 3 9

3 5 5

4 5 7

6 5 7

3 2 9

3 5 5

4 3 6

4 5 7

6 5 7

7 2 0

8 3 9

Assume by induction:

after iteration i you have sorted all elements by the last i digits.

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

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

$i \leftarrow$
7 2 0
3 2 9
4 3 6
8 3 9
3 5 5
4 5 7
6 5 7

