Faculty of Computer Science
Data Science and Business Analytics (DSBA)

Algorithms and Data Structures

Seminar 3. Counting Sort and Radix Sort

Idea: sorting algorithms that avoid to compare elements (A[i] < A[i + 1]) in the array to obtain lower complexity.

September 2021

J.C. Carrasquel

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T*(*n*).
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T*(*n*).
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n)\leq c\,n\log(n)\ \exists c>0$ `// definition of Big-O`

(2). $T(m)\leq c\,m\log(m)\ \forall m\leq n$ `// inductive step (assumption)`

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T*(*n*).
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n) \leq c\,n\log(n)$ $\exists c>0$

(2). $T(m) \leq c\,m\log(n)$ $\forall m<n$

$Let\ m=\frac{n}{2}$

(3). $T(\frac{n}{2}) \leq c\frac{n}{2}\log(\frac{n}{2})$ `// placing m = n/2 in (2)`

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T(n)*.
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n) \leq c\,n\log(n)$ $\exists c>0$

(2). $T(m) \leq c\,m\log(n)$ $\forall m<n$

$Let\ m=\frac{n}{2}$

(3). $T(\frac{n}{2}) \leq c\frac{n}{2}\log(\frac{n}{2})$

(4). $T(n) \leq 2c\frac{n}{2}\log(\frac{n}{2}) + n$ `// substitution of T(n/2) in definition of T(n)`

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T*(*n*).
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n) \le c\,n\log(n)$ $\exists c>0$

(2). $T(m) \le c\,m\log(n)$ $\forall m<n$

$Let\ m=\frac{n}{2}$

(3). $T(\frac{n}{2}) \le c\frac{n}{2}\log(\frac{n}{2})$

// log(a/b) = log(a) - log(b)

(4). $T(n) \le \cancel{2}c\frac{n}{\cancel{2}}\log(\frac{n}{2}) + n = c\,n\log(\frac{n}{2}) + n = c\,n(\log(n)-\log(\cancel{2}^{1})) + n$

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T*(*n*).
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n) \le c\,n\log(n)\ \exists c>0$

(2). $T(m) \le c\,m\log(n)\ \forall m<n$

$Let\ m=\frac{n}{2}$

(3). $T(\frac{n}{2}) \le c\frac{n}{2}\log(\frac{n}{2})$

(4). $T(n) \le \cancel{2}c\frac{n}{\cancel{2}}\log(\frac{n}{2}) + n = c\,n\log(\frac{n}{2}) + n = c\,n(\log(n)-\log(\cancel{2}^{1})) + n$

$T(n) \le c\,n\log(n)-cn + n$

$T(n) \le c\,n\log(n)-(c-1)n$

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T(n)*.
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n) \le cn\log(n)$ $\exists c>0$

(2). $T(m) \le cm\log(n)$ $\forall m<n$

Let $m=\frac{n}{2}$

(3). $T(\frac{n}{2}) \le c\frac{n}{2}\log(\frac{n}{2})$

(4). $T(n) \le 2c\frac{n}{2}\log(\frac{n}{2}) + n = cn\log(\frac{n}{2}) + n = cn(\log(n)-\log(2^1)) + n$

$T(n) \le cn\log(n)-cn + n$

$T(n) \le cn\log(n)-(c-1)n$

(5). $cn\log(n)-(c-1)n \le cn\log(n)$ `// for which values of c?`

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T(n)*.
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n) \le cn\log(n)$ $\exists c>0$

(2). $T(m) \le cm\log(n)$ $\forall m<n$

Let $m=\frac{n}{2}$

(3). $T(\frac{n}{2}) \le c\frac{n}{2}\log(\frac{n}{2})$

(4). $T(n) \le \cancel{2}c\frac{n}{\cancel{2}}\log(\frac{n}{2}) + n = cn\log(\frac{n}{2}) + n = cn(\log(n)-\log(\cancel{2}^{1})) + n$

$T(n) \le cn\log(n)-cn + n$

$T(n) \le cn\log(n)-(c-1)n$

(5). $cn\log(n)-(c-1)n \le cn\log(n)$ `// for which values of c?`

$n\log(n) \le n\log(n)$ if $c=1$ 

$\frac{1}{2}n\log(n)+\frac{1}{2}n \le \frac{1}{2}n\log(n)$ if $c=\frac{1}{2}$ 

Substitution Method (cont.)

Idea:
1. Propose (guess) an order of time complexity for *T*(*n*).
2. Find constants (by induction) that support your proposed solution.

Example. Prove that $T(n)=2T(\frac{n}{2})+n$ is $O(n\log(n))$

(1). $T(n) \le cn\log(n)$ $\exists c>0$

(2). $T(m) \le cm\log(n)$ $\forall m<n$

Let $m=\frac{n}{2}$

(3). $T(\frac{n}{2}) \le c\frac{n}{2}\log(\frac{n}{2})$

(4). $T(n) \le \cancel{2}c\frac{n}{\cancel{2}}\log(\frac{n}{2}) + n = cn\log(\frac{n}{2}) + n = cn(\log(n)-\log(\cancel{2}^{1})) + n$

$T(n) \le cn\log(n)-cn + n$

$T(n) \le cn\log(n)-(c-1)n$

(5). $cn\log(n)-(c-1)n \le cn\log(n)$

Solution: $T(n) \le cn\log(n)$ $\forall c \ge 1$

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

	0	1	2	3	4	5	6	7
a:	2	5	3	0	2	3	0	3

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

	0	1	2	3	4	5	6	7
a:	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
c:	0	0	0	0	0	0

c[i] : number of elements equal to i

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

	0	1	2	3	4	5	6	7
a:	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
c:	2	0	2	3	0	1

c[i] : number of elements equal to i

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

	0	1	2	3	4	5	6	7
a:	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
c:	2	0	2	3	0	1

```
Update array c
c[i] will be the number of elements equal or less than i
c[i] = c[i] + c[i - 1]
```

	0	1	2	3	4	5
c:	2	?	?	?	?	?

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

	0	1	2	3	4	5	6	7
a:	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
c:	2	0	2	3	0	1

```
Update array c
c[i] will be the number of elements equal or less than i
c[i] = c[i] + c[i - 1]
```

	0	1	2	3	4	5
c:	2	2	4	7	7	8

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

	0	1	2	3	4	5	6	7
a:	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
c:	2	2	4	7	7	8

c[i] : number of elements equal or less than i

	0	1	2	3	4	5	6	7
b:	0	0	0	0	0	0	0	0

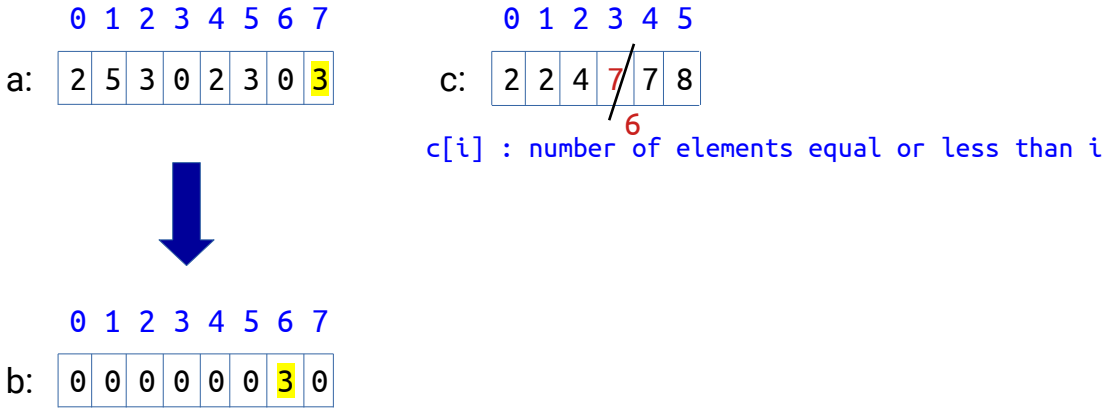
sorted array

```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

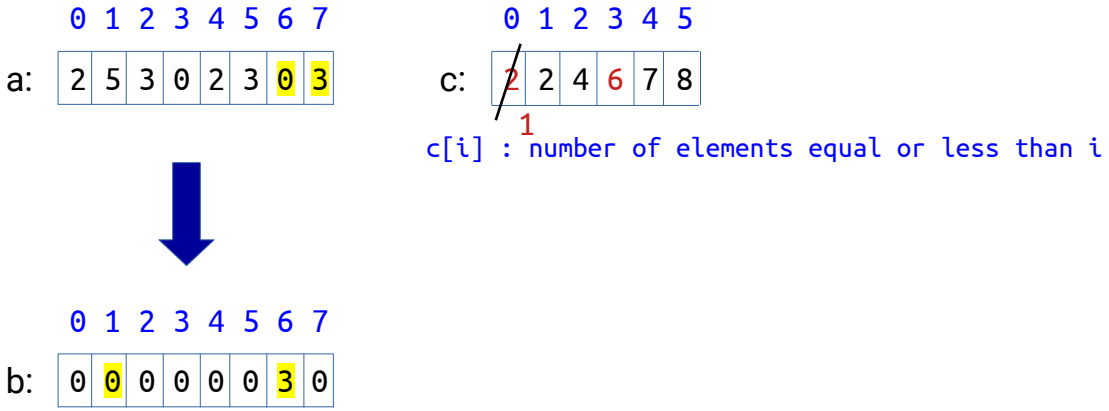



```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

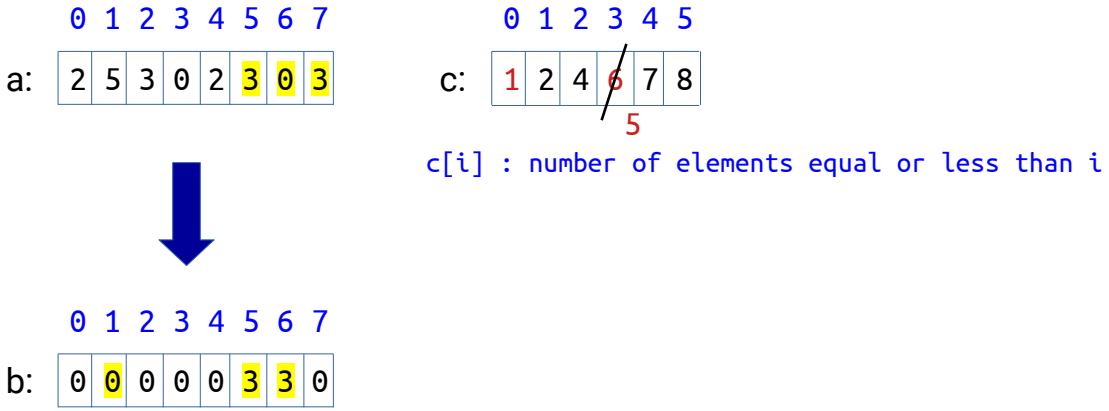



```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

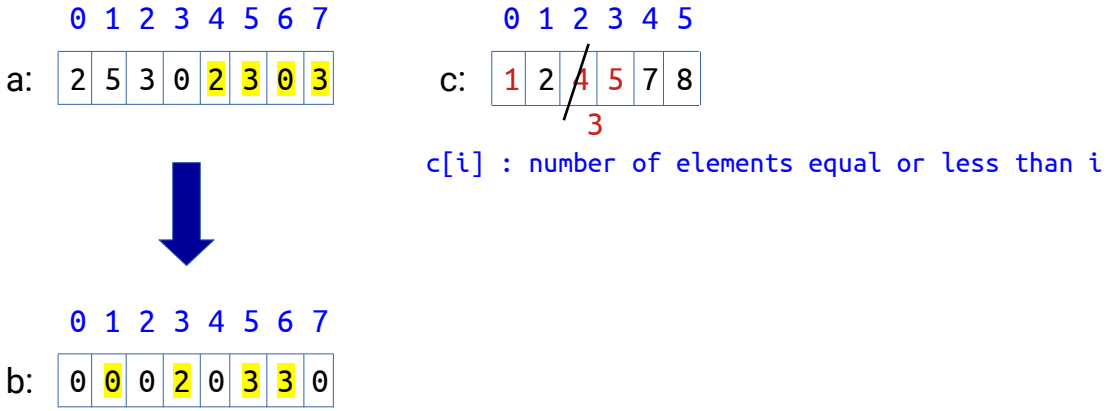



```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

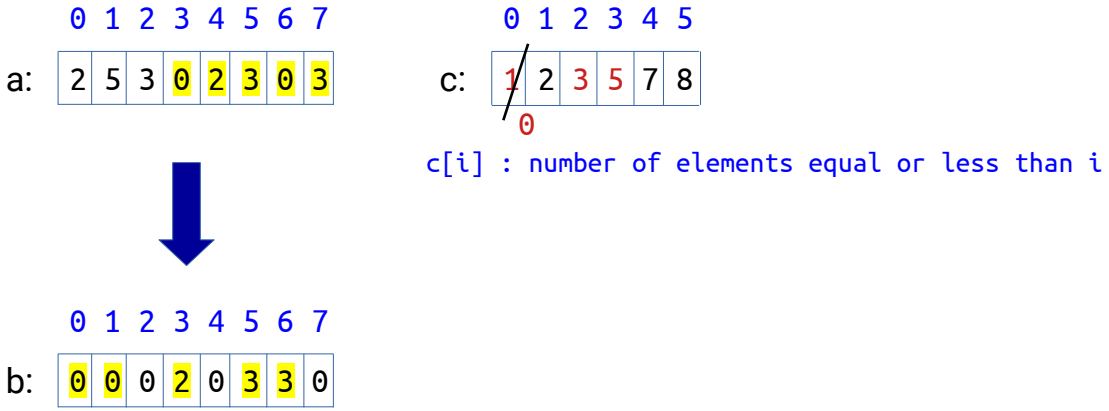



```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

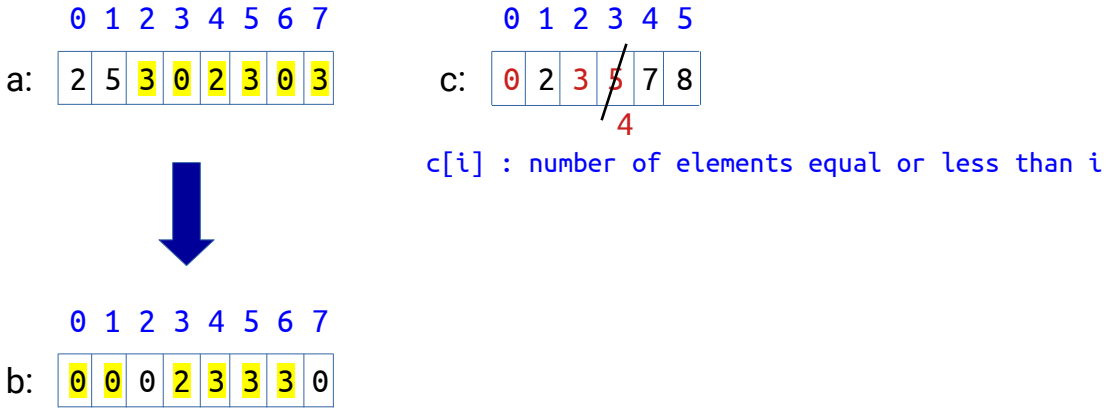



```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

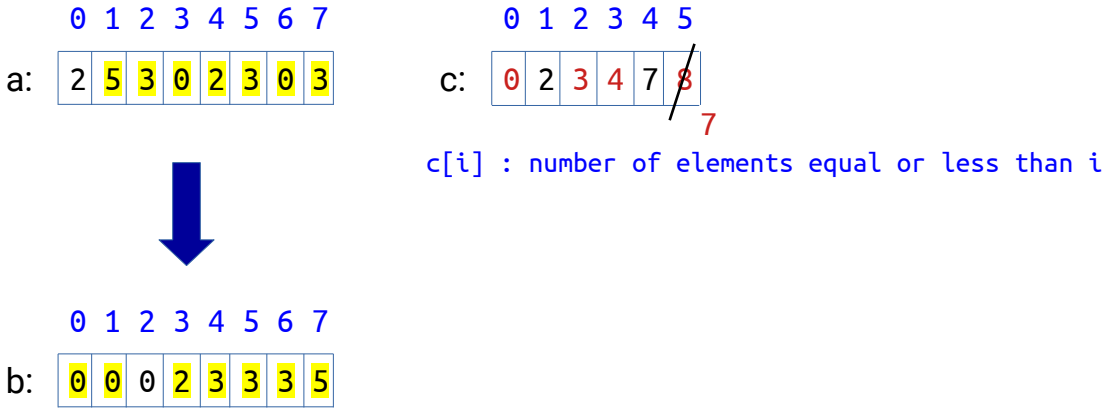



```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

n = 8 k = 5

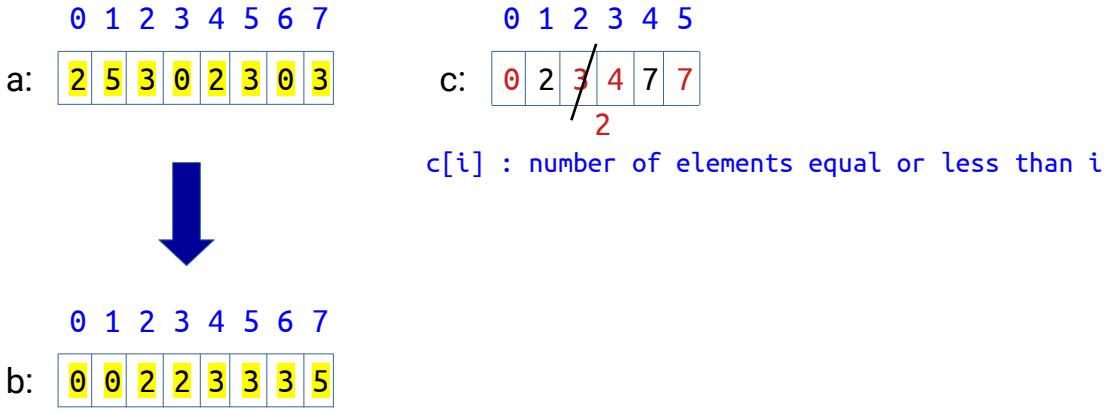



```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

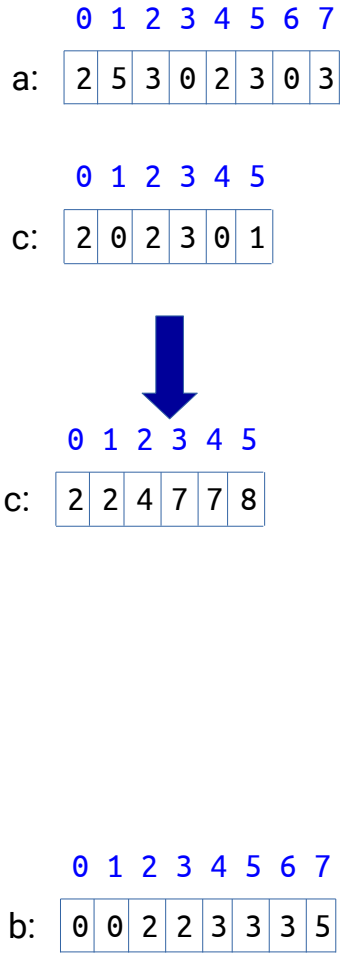
n = 8 k = 5




```
for(j = n - 1; j >= 0; j--)
  insert a[j] in position c[a[j]] - 1 of array b;
  c[a[j]] ← c[a[j]] - 1;
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

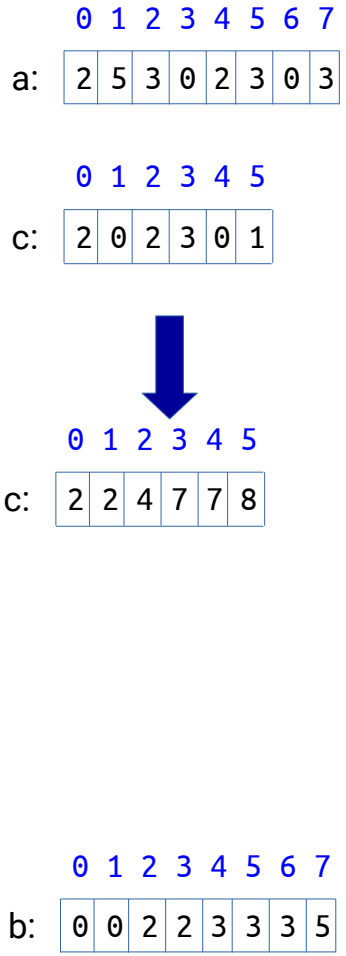



```
1  void countingSort(int a[], int b[], int n, int k)
2  {
3      int c[k + 1];
4
5      for(int i = 0; i <= k; i++)
6          c[i] = 0;
7
8      for(int j = 0; j < n; j++)
9          c[a[j]]++;
10
11     // c[i]: number of elements equal to i
12
13     for(int i = 1; i <= k; i++)
14         c[i] = c[i] + c[i-1];
15
16     // c[i]: number of elements less or equal to i
17
18     for(int j = n - 1; j >= 0; j--)
19     {
20         b[c[a[j]] - 1] = a[j];
21         c[a[j]]--;
22     }
23 }
```

Counting Sort

Sort an array of n integers, where each integer is in the range [0…k]

O(n) provided that k is less or equal than n




```
1  void countingSort(int a[], int b[], int n, int k)
2  {
3      int c[k + 1];
4
5      for(int i = 0; i <= k; i++)
6          c[i] = 0;
7
8      for(int j = 0; j < n; j++)
9          c[a[j]]++;
10
11     // c[i]: number of elements equal to i
12
13     for(int i = 1; i <= k; i++)
14         c[i] = c[i] + c[i-1];
15
16     // c[i]: number of elements less or equal to i
17
18     for(int j = n - 1; j >= 0; j--)
19     {
20         b[c[a[j]] - 1] = a[j];
21         c[a[j]]--;
22     }
23 }
```

Radix Sort

Idea:

- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1.* Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins

d = 3 **a:**

0	329
1	457
2	657
3	839
4	436
5	720
6	355

bins:

0

1

2

3

4

5

6

7

8

9

Radix Sort

Idea:
- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1.* Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins

d = 3 *j = 2*

a:

0	329
1	457
2	657
3	839
4	436
5	720
6	355

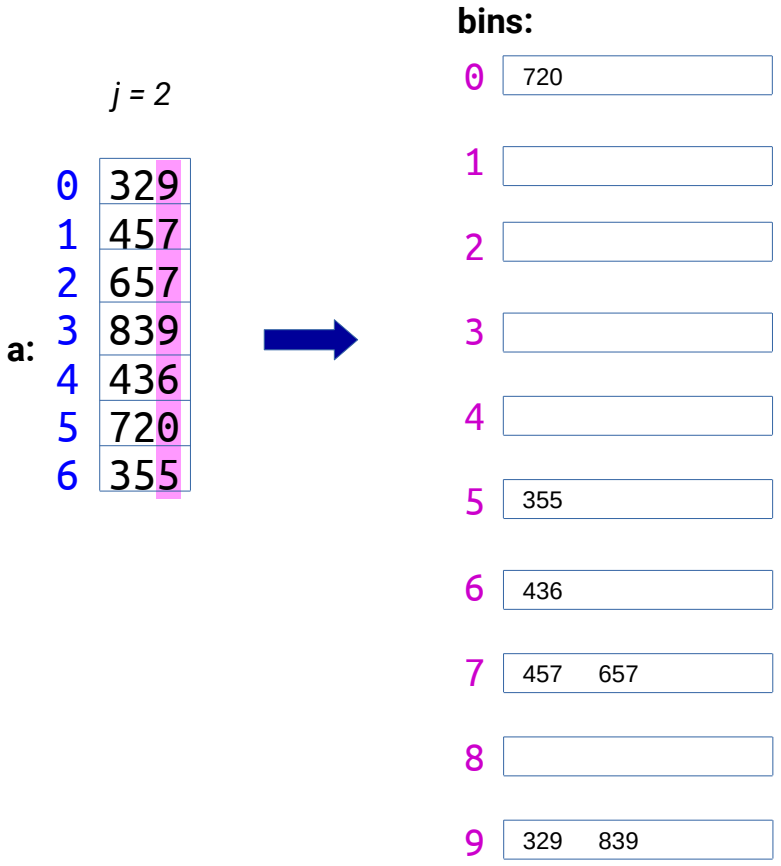
bins:

0
1
2
3
4
5
6
7
8
9

Radix Sort

Idea:

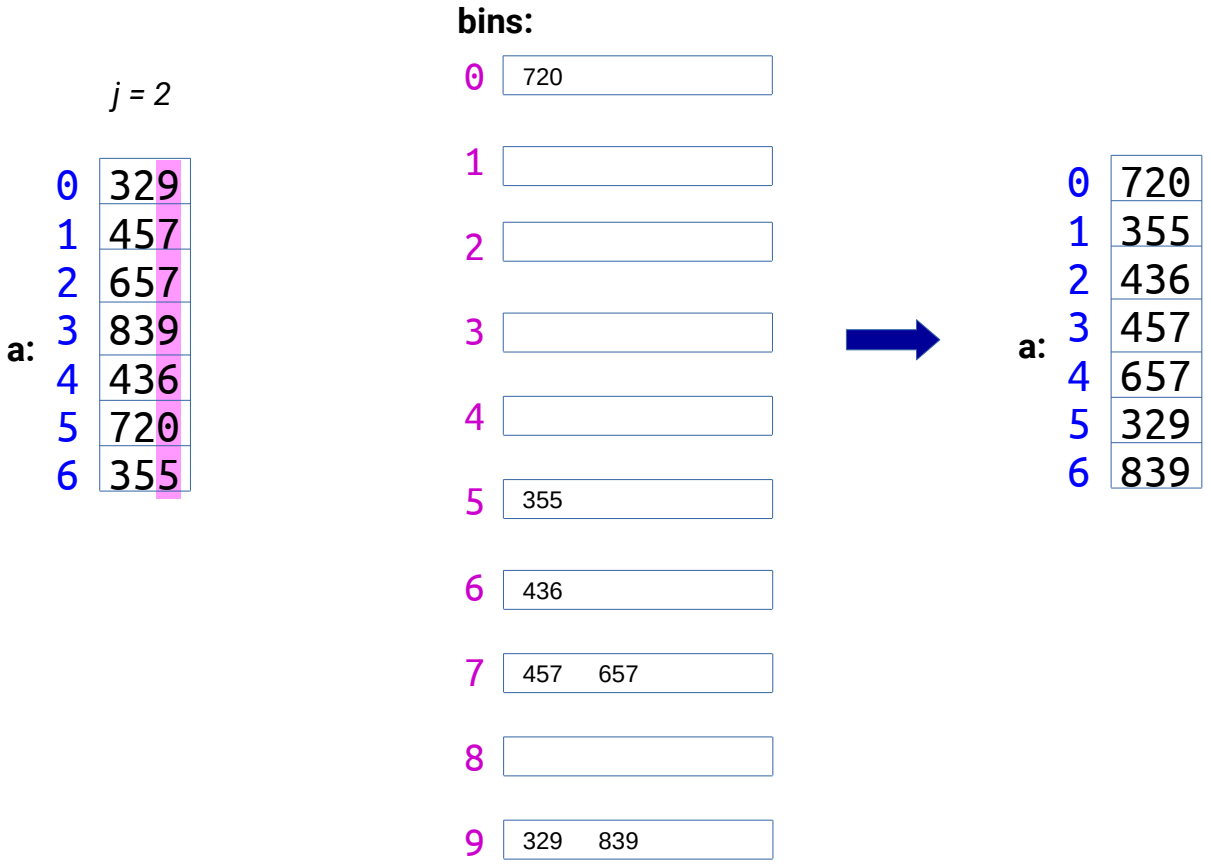
- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1.* Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins



Radix Sort

Idea:

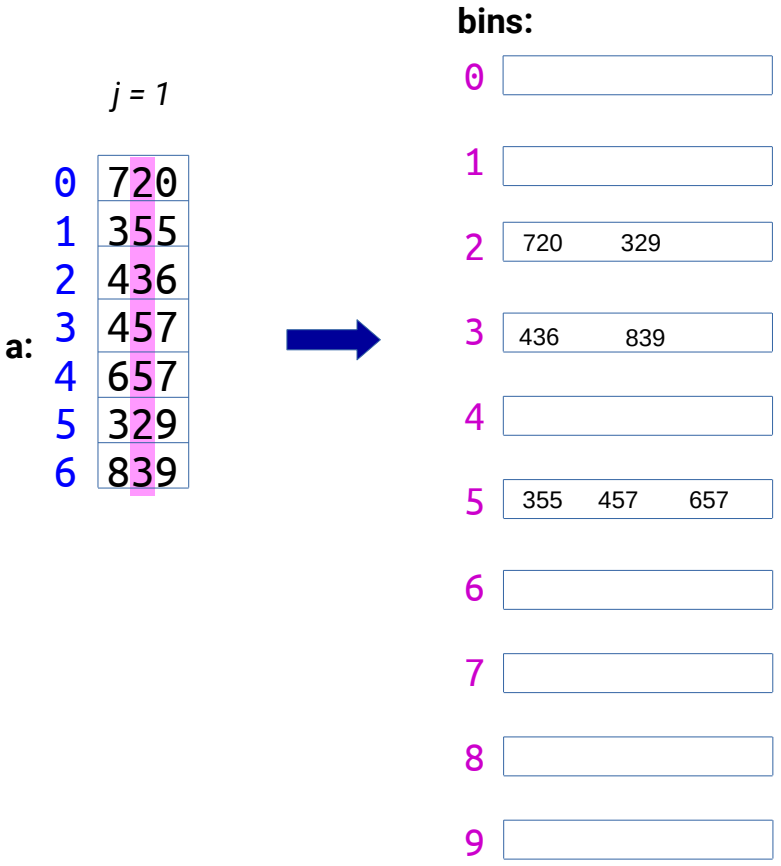
- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1*. Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins



Radix Sort

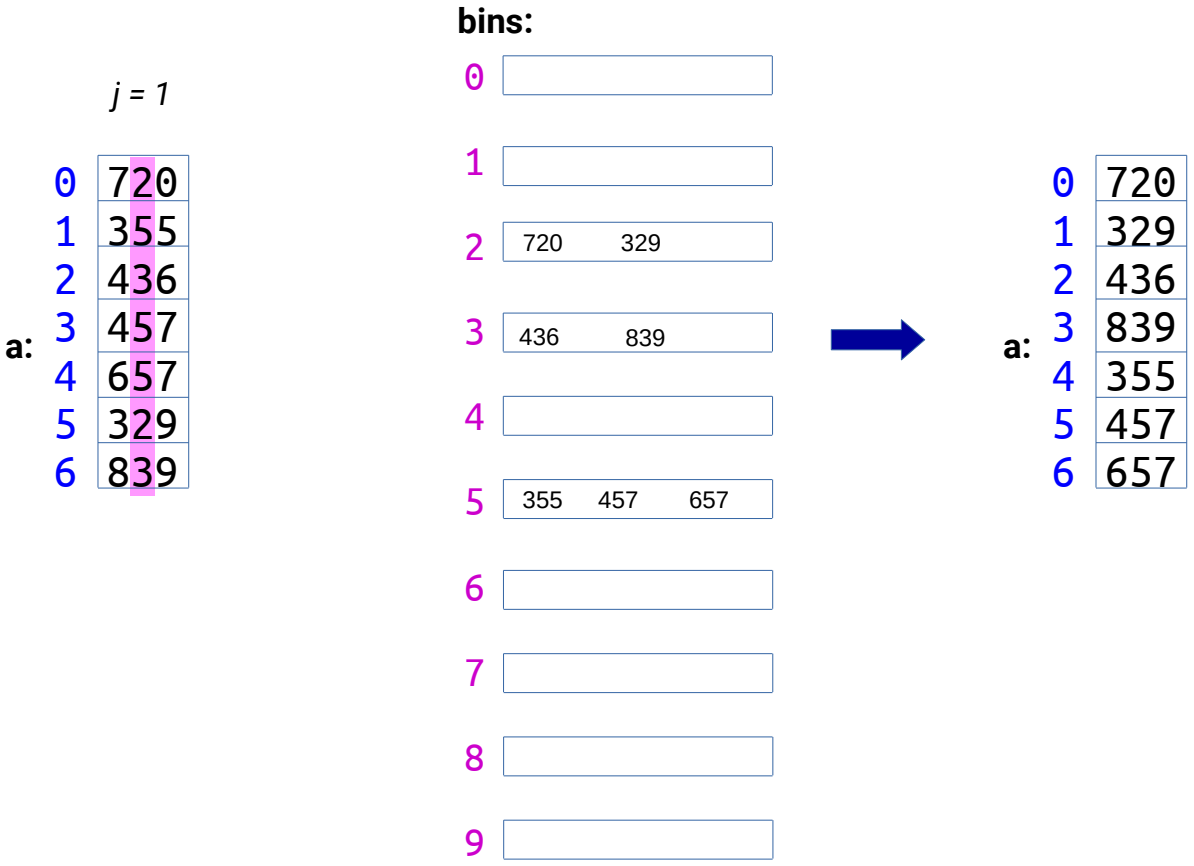
Idea:

- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1*. Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins



Radix Sort

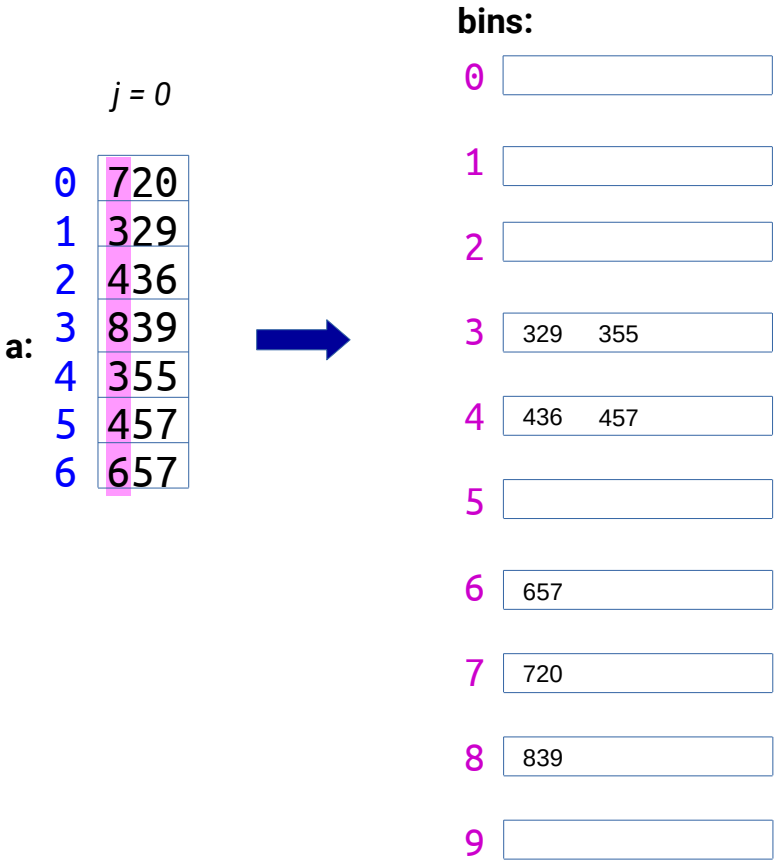
Idea:
- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1*. Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins



Radix Sort

Idea:

- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1.* Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins

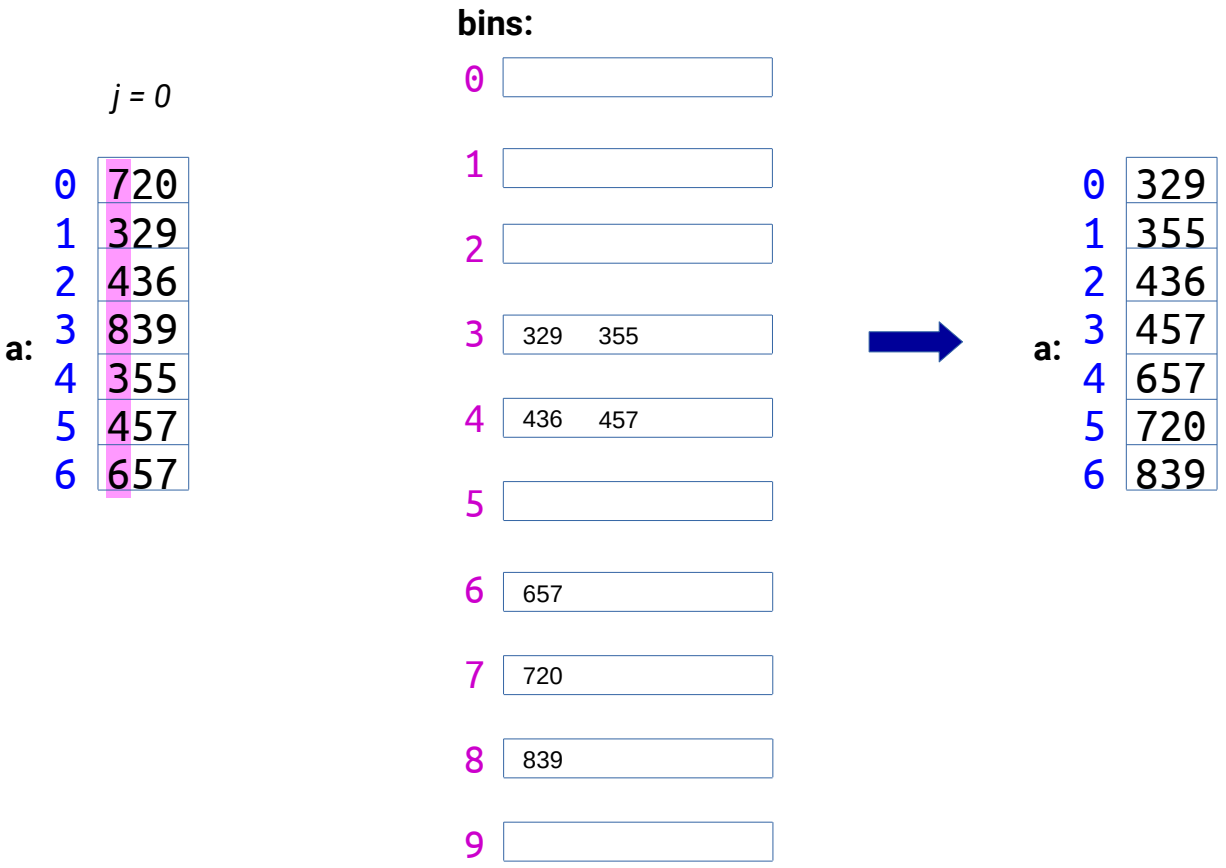


Radix Sort

O(n*d) for n numbers of d digits

Idea:

- Sort an array “a” of “n” integers, where each integer has “d” digits.
- 10 “bins” (vectors) corresponding to digits 0,…,9
- Loop, starting from the rightmost digit (“least significant”). *j* = d – 1, d – 2, …, 0
  - For i = *0,...n-1*. Insert number a[i] at the end of bin number digit(j, a[i])
  - Update array “a” by combining bins 0,…,9 (in order)
  - Clear the bins



Exercise

Implement the RadixSort algorithm to sort a vector of n strings.

input:

```
std::vector<std::string> v = {"COW", "DOG", "SEA", "RUG", "ROW", "MOB", "BOX", "TAB", "BAR", "EAR", "TAR", "DIG", "BIG", "TEA", "NOW", "FOX"};
```

output:

BAR BIG BOX COW DIG DOG EAR FOX MOB NOW ROW RUG SEA TAB TAR TEA

Notes/Hints:

- You are free to use C++ or Python! :-)
- You can assume all strings have the same number of chars!
- To ease the work, consider only uppercase letters (or only lowercase letters)
- In C++, you could create the bins as written below.

```
using Bins = std::map<char,std::vector<std::string>>;
Bins bins;
```

- How many pairs (char,std::vector) in the map (how many bins) should be initialized in the map before starting to sort the vector?
- Remember to clean the bins in your map after each iteration.
  - For C++ users, you can use bins[c].clear() to remove all elements from the bin of letter c.