CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 7:
Randomized Quick Sort, Counting Sort, Hashing

Randomized Algorithms: Randomized Quick Sort

Quick Sort

- **Quick-Sort**(A[1:n]):

1. If n=0 or 1, return A.
2. Pick p to be any **arbitrary** number in $\{1,2,\cdots,n\}$, say p =1.
3. Run **Partition**(A,p) and let q be returned position of pivot.
4. Recursively run **Quick-Sort**(A[1:q-1]) and **Quick-Sort**(A[q+1:n]).

Randomized Quick Sort

- **Randomized-Quick-Sort**(A[1:n]):

1. If n=0 or 1, return A.
2. Pick p to be a **uniformly at random** number in $\{1, 2, \cdots, n\}$
3. Run **Partition**(A,p) and let q be returned position of pivot.
4. Recursively run **Randomized-Quick-Sort**(A[1:q-1]) and **Randomized-Quick-Sort**(A[q+1:n]).

Randomized Algorithm

- An algorithm that uses randomization (!)
- Two main types of randomized algorithms:
  - Monte Carlo: An algorithm that uses randomization to “help” its correctness:
    - It outputs a correct answer on any input with a large probability, say, 99%, but may sometimes output a wrong number
  - Las Vegas: An algorithm that uses randomization to “help” its running time:
    - It is always correct on every input but its runtime depends on the random bits

Randomized Algorithm

- An algorithm that uses randomization (!)
- Two main types of randomized algorithms:
  - Monte Carlo: An algorithm that uses randomization to “help” its correctness:
    - It outputs a correct answer on any input with a large probability, say, 99%, but may sometimes output a wrong number
  - **Las Vegas: An algorithm that uses randomization to “help” its running time:**
    - **It is always correct on every input but its runtime depends on the random bits**

Counting Sort and Simple Searching

Counting Sort

- A very simple sorting algorithm for sorting n numbers in $\{1,2,\ldots,M\}$ in $O(n+M)$ time.

- Also a very helpful idea for a simple searching algorithm

Counting Sort

- **Counting-Sort**(A[1:n],M)

  1. Create an array C[1:M] initialized to be 0
  2. For i=1 to n: increase C[A[i]] by one
  3. Let p = 1. For j = 1 to M:

     A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

Counting Sort: Example

n=8 M=6

A

4	2	2	5	3	5	1	6

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

A	4	2	2	5	3	5	1	6

C	0	0	0	0	0	0

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0

2. For i=1 to n: increase C[A[i]] by one

3. Let p = 1. For j = 1 to M:

   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

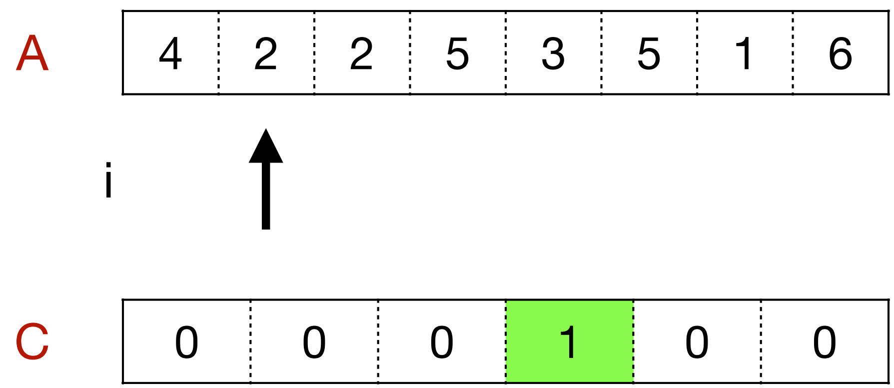
A	4	2	2	5	3	5	1	6

i

C	0	0	0	0	0	0

Counting Sort: Example

n=8 M=6



- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

Counting Sort: Example

n=8 M=6

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0

2. For i=1 to n: increase C[A[i]] by one

3. Let p = 1. For j = 1 to M:

   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

A

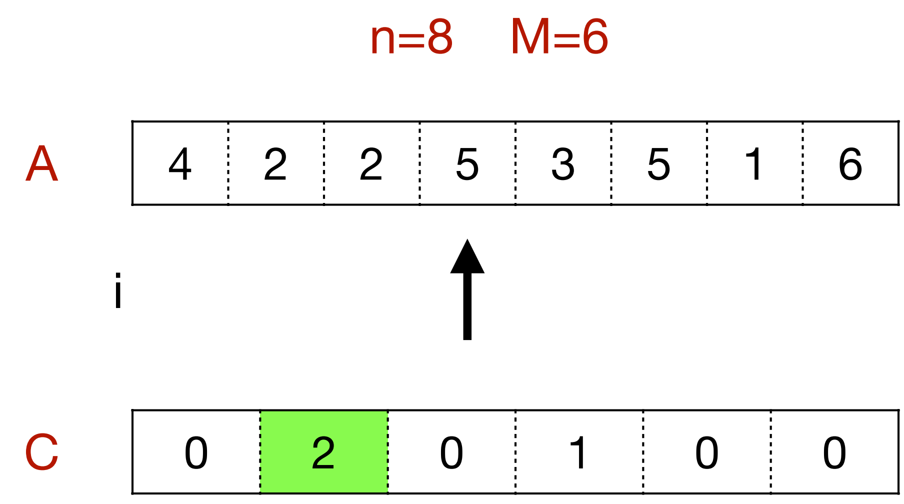
4	2	2	5	3	5	1	6

i

C

0	1	0	1	0	0

Counting Sort: Example

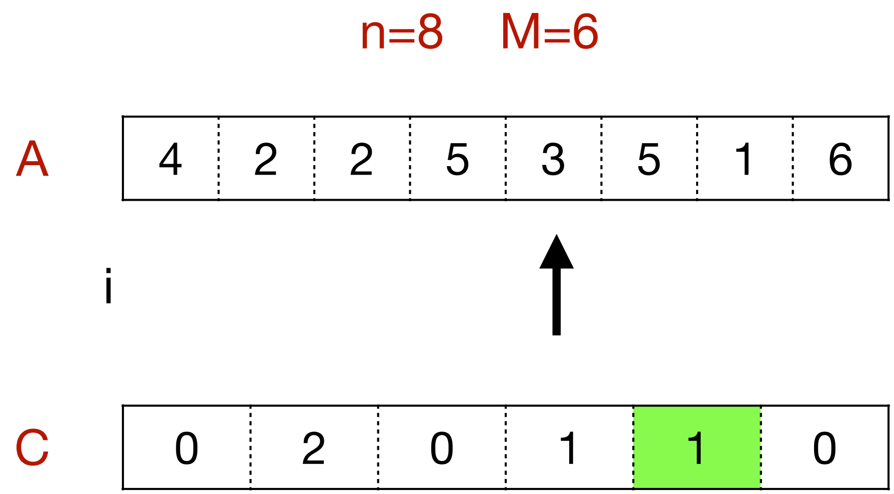


- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:

   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

Counting Sort: Example



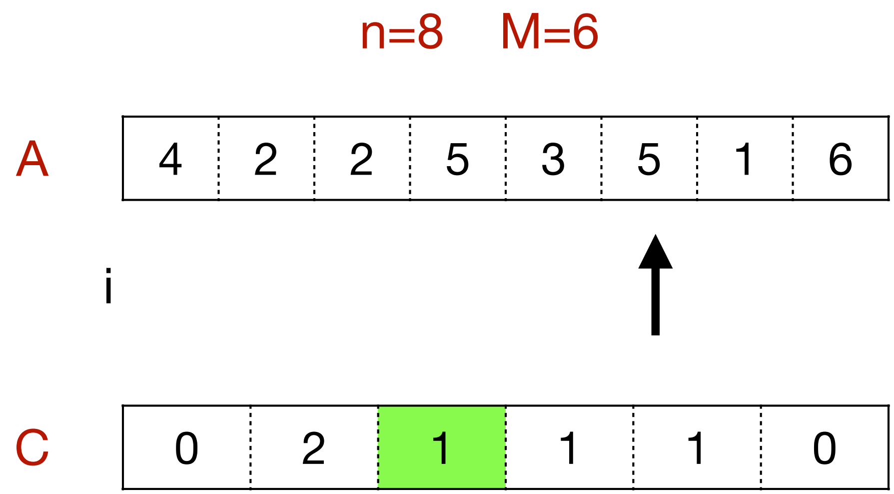
- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one



Counting Sort: Example

n=8 M=6

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

A	4	2	2	5	3	5	1	6

i

C	0	2	1	1	2	0

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

A

4	2	2	5	3	5	1	6

i

C

1	2	1	1	2	0

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:

   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

A	4	2	2	5	3	5	1	6

i

C	1	2	1	1	2	1

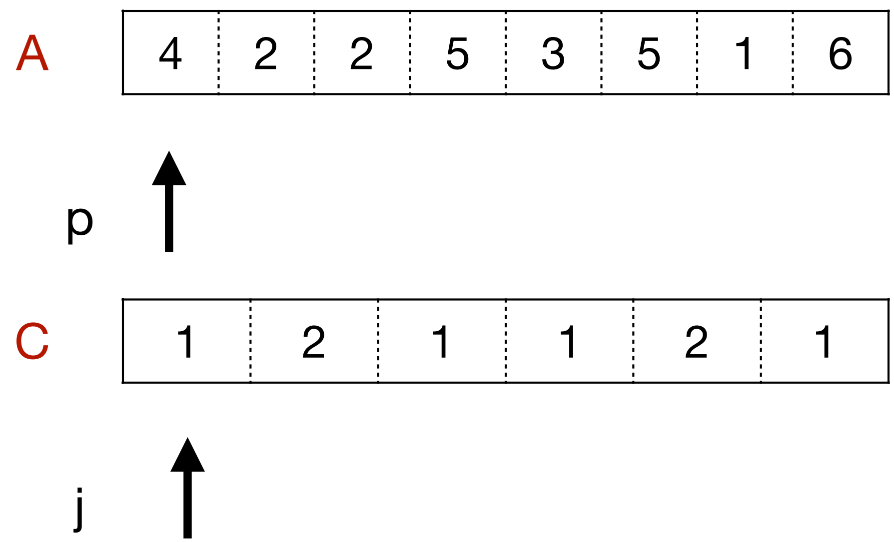
Counting Sort: Example

n=8 M=6

- **Counting-Sort**(A[1:n],M)

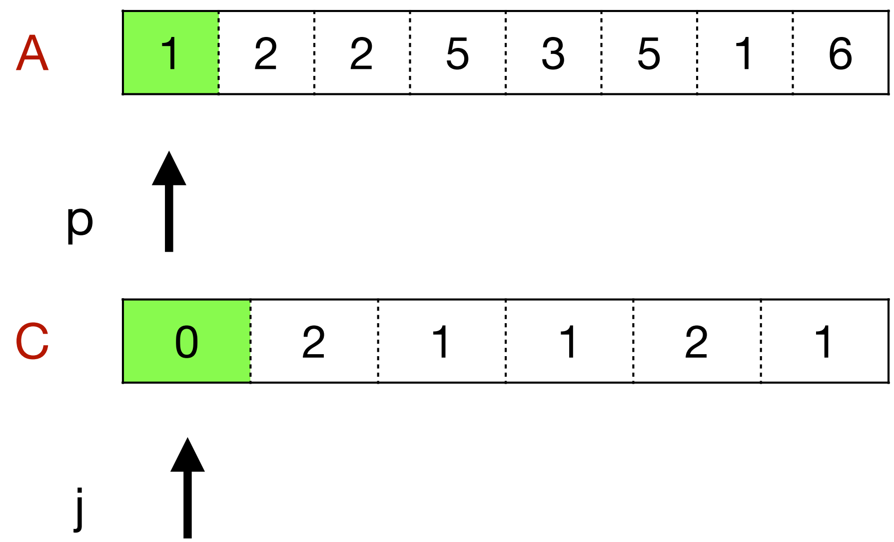
1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:

   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one



Counting Sort: Example

n=8 M=6



- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

A								
	1	2	2	5	3	5	1	6

p

C						
	0	2	1	1	2	1

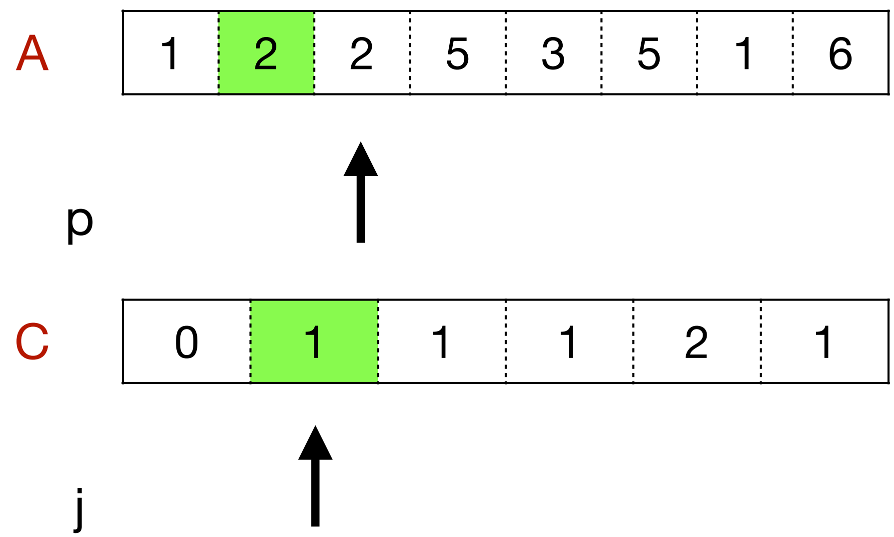
j

Counting Sort: Example

n=8 M=6

- **Counting-Sort**(A[1:n],M)

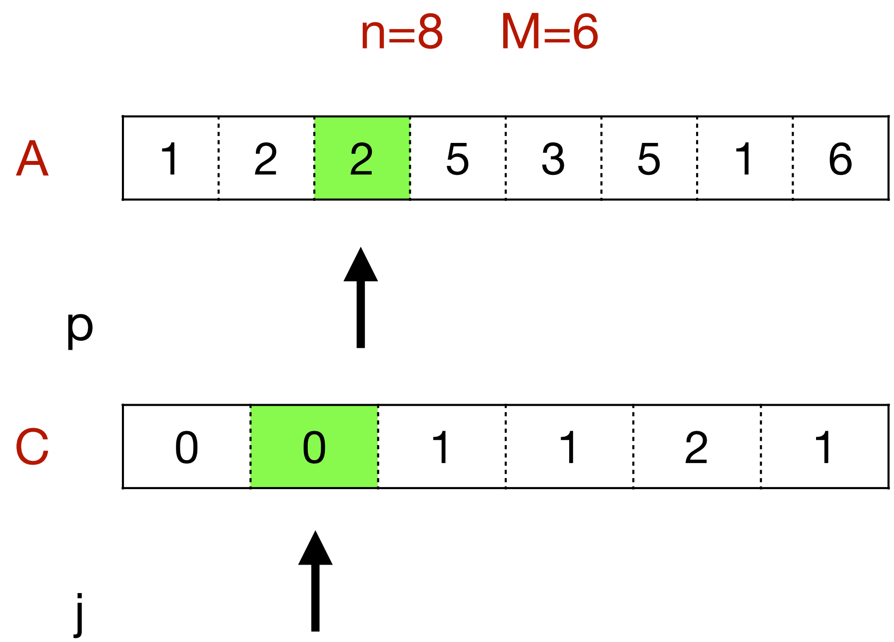
1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one



Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one



Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

A

1	2	2	5	3	5	1	6

p

C

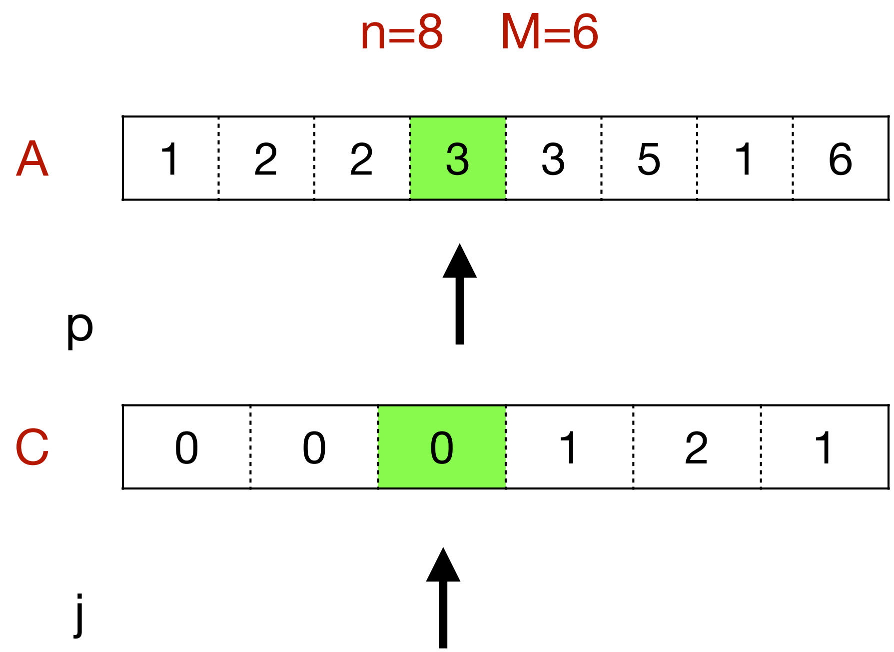
0	0	1	1	2	1

j

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one



Counting Sort: Example

n=8 M=6

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

A

1	2	2	3	3	5	1	6

p

C

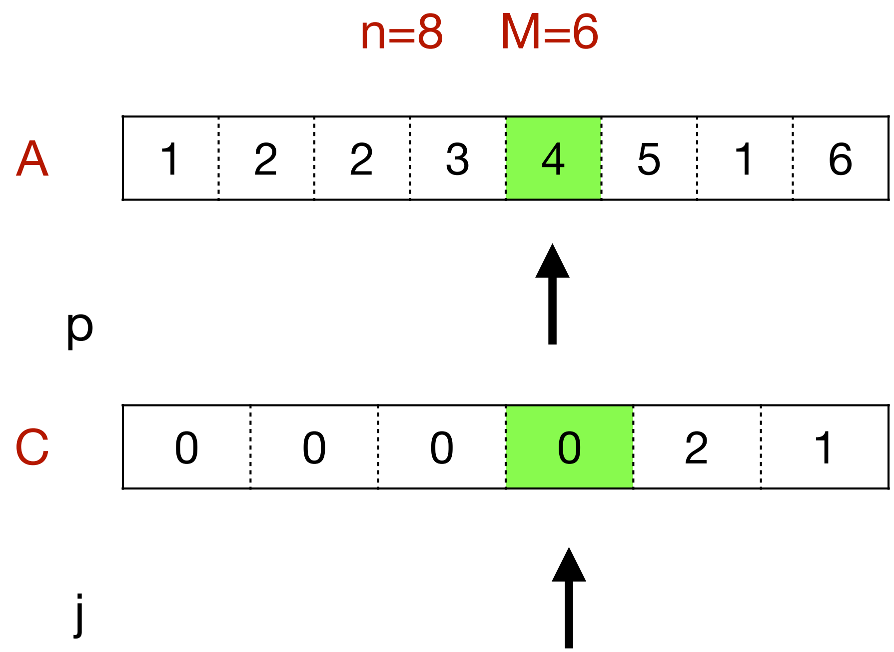
0	0	0	1	2	1

j

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one



Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

A

1	2	2	3	4	5	1	6

p

C

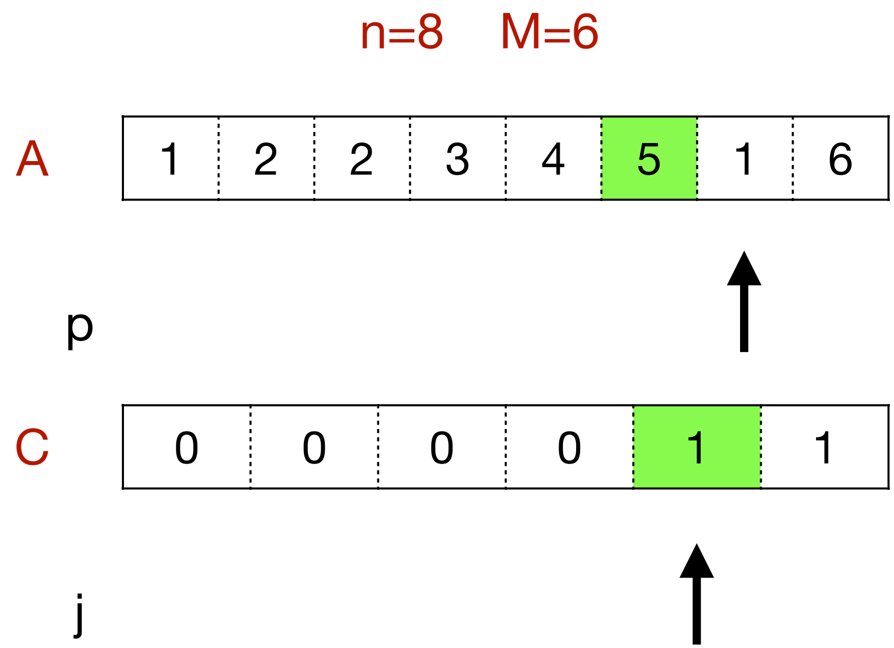
0	0	0	0	2	1

j

Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

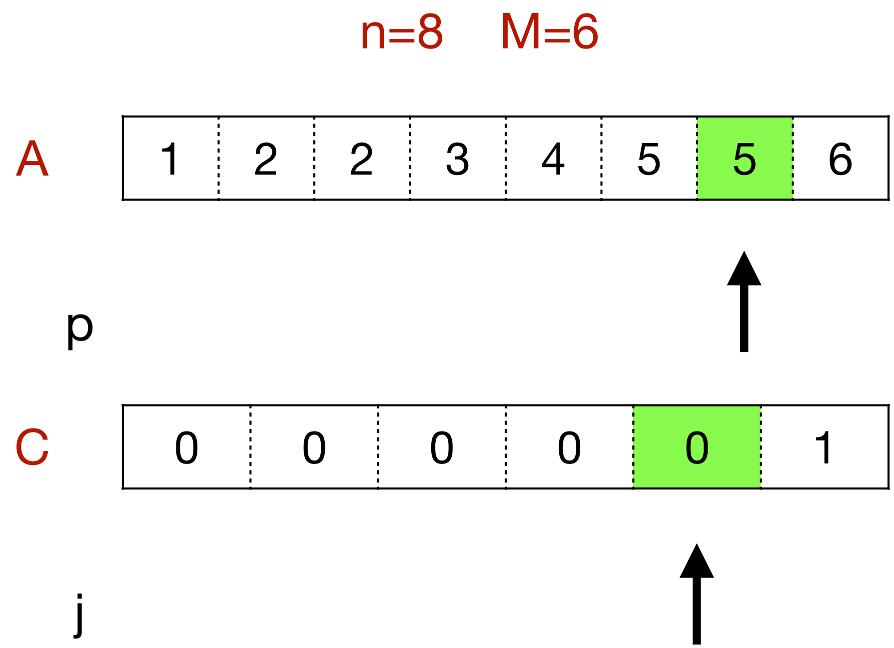
1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one



Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

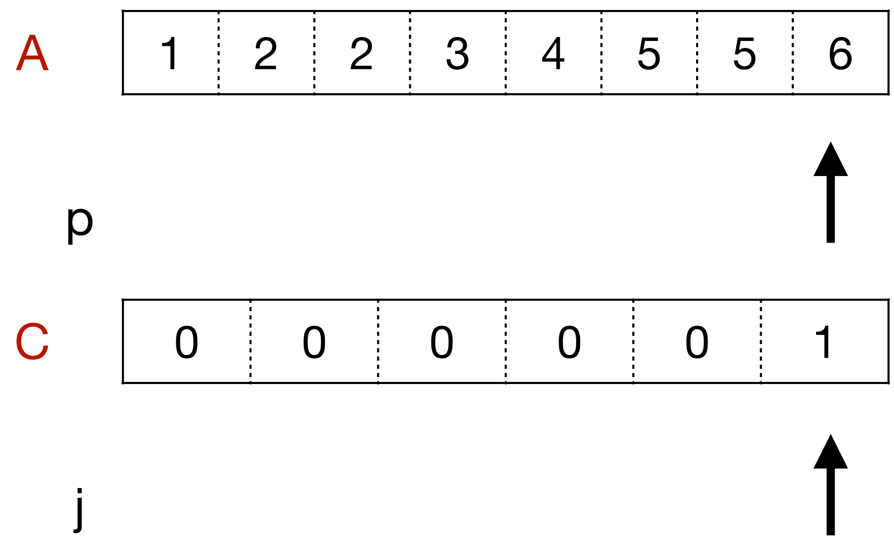


Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6

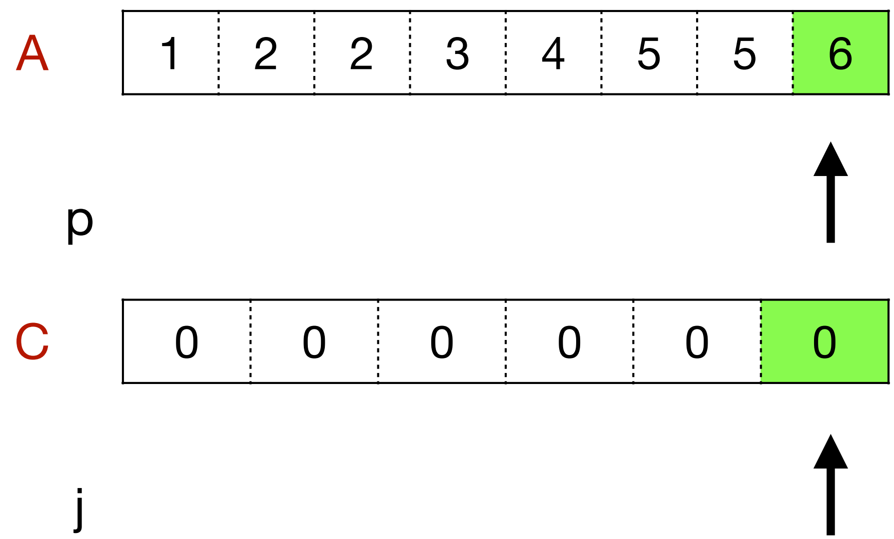


Counting Sort: Example

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

n=8 M=6



Counting Sort: Example

$n=8 \quad M=6$

A

1	2	2	3	4	5	5	6

C

0	0	0	0	0	0

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

Counting Sort: Proof of Correctness

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

- **Observation**: After line 2, for every $1 \leq j \leq M$, C[j] is equal to # of times j appears in A.
- For any $1 \leq j \leq M$, define p_j as the value of pointer p after iteration j.
- **Statement:** For $1 \leq j \leq M$, after iteration j, array A[1:p_j-1] contains all numbers $\leq j$ originally in A in the sorted order

Counting Sort: Runtime Analysis

- **Counting-Sort**(A[1:n],M)

1. Create an array C[1:M] initialized to be 0
2. For i=1 to n: increase C[A[i]] by one
3. Let p = 1. For j = 1 to M:
   A. While C[j] > 0, let A[p] = j, increase p by one and decrease C[j] by one

- Line 1 takes O(M) time
- Line 2 takes O(n) time
- Line 3 takes O(M) iterations and total while-loops can take another O(n) time
- So total runtime is O(n+M)