COUNTING SORT

SORTING IN LINEAR TIME

- Counting sort
  - No comparisons between elements!
  - ***But…*** depends on assumption about the numbers being sorted
    - We assume numbers are in the range *1.. K*
  - The algorithm:
    - Input: A[1..*N*], where A[j] $\in$ {1, 2, 3, …, *K*}
    - Output: B[1..*N*], sorted
    - Also: Array C[1..*K*] for auxiliary storage

STEP 1

Find the number of times $A[i]$ appears in A

(i.e., frequencies)

input array A:

3	6	4	1	3	4	1	4

Allocate $C[1..r]$ *(r=6)*

allocate C

1	2	3	4	5	6
0	0	0	0	0	0

For $1 \le i \le n, ++C[A[i]];$

i=1, A[1]=3

1	2	3	4	5	6
0	0	1	0	0	0

C[A[1]]=C[3]=1

i=2, A[2]=6

1	2	3	4	5	6
0	0	1	0	0	1

C[A[2]]=C[6]=1

i=3, A[3]=4

1	2	3	4	5	6
0	0	1	1	0	1

C[A[3]]=C[4]=1

.
.
.

i=8, A[8]=4

1	2	3	4	5	6
2	0	2	3	0	1

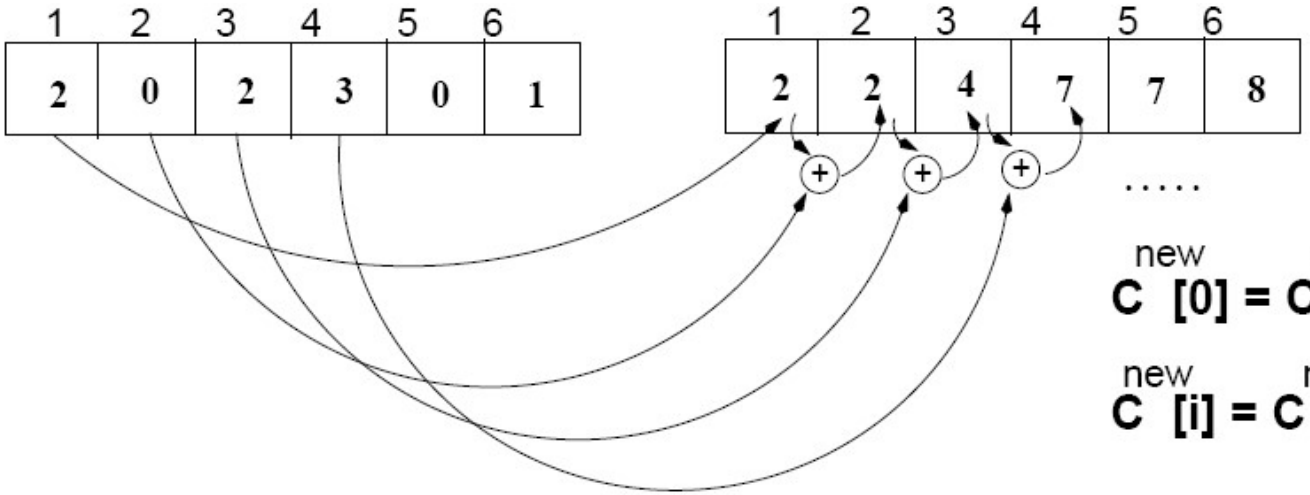
C[A[8]]=C[4]=3

C[i] = number of times element i appears in A

STEP 2

Find the number of elements $\leq A[i]$,

C *(frequencies)* *C^new^* *(cumulative sums)*



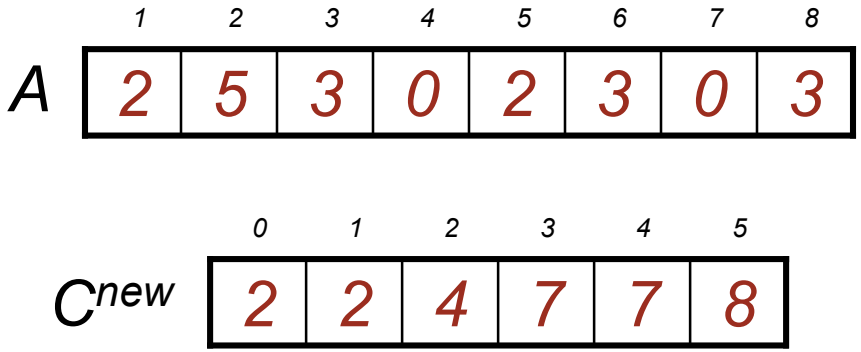
$C^{new}[0] = C^{old}[0]$

$C^{new}[i] = C^{new}[i-1] + C^{old}[i]$

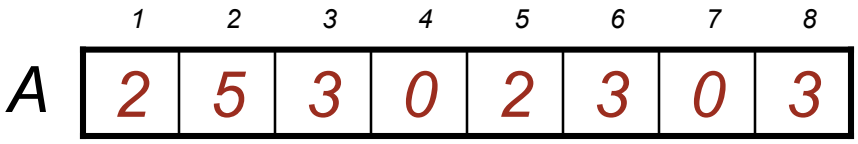
C[i] = # elements <= i

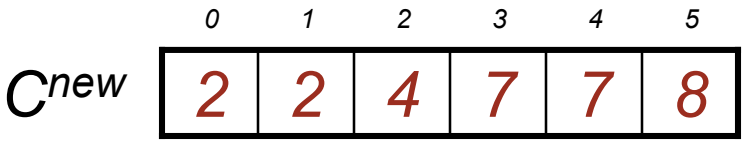
ALGORITHM

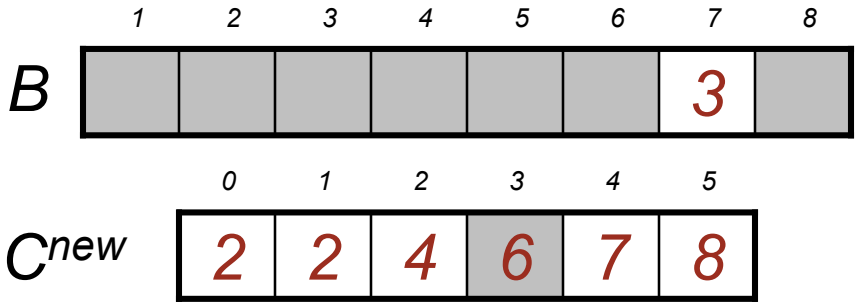
- Start from the last element of A
- Place A[i] at its correct place in the output array
- Decrease C[A[i]] by one

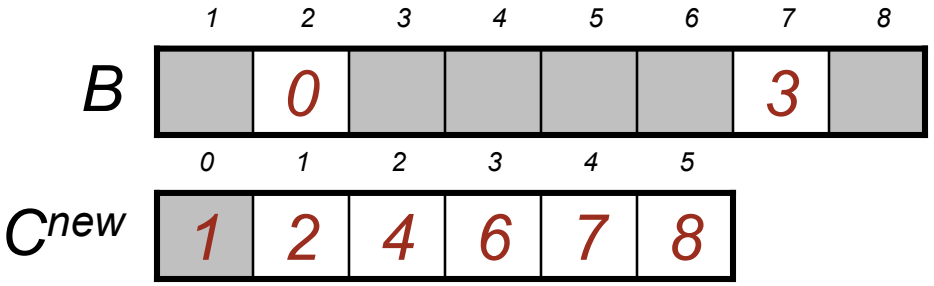


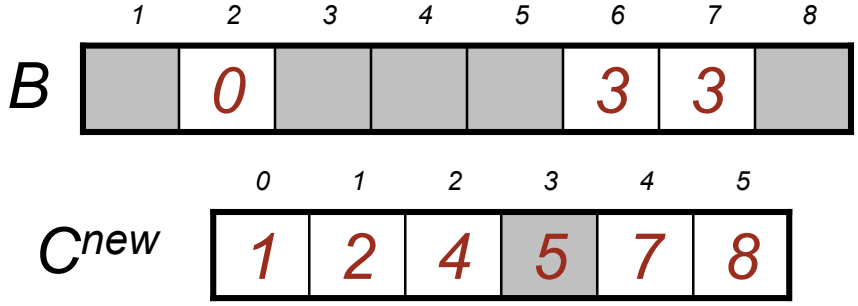
EXAMPLE

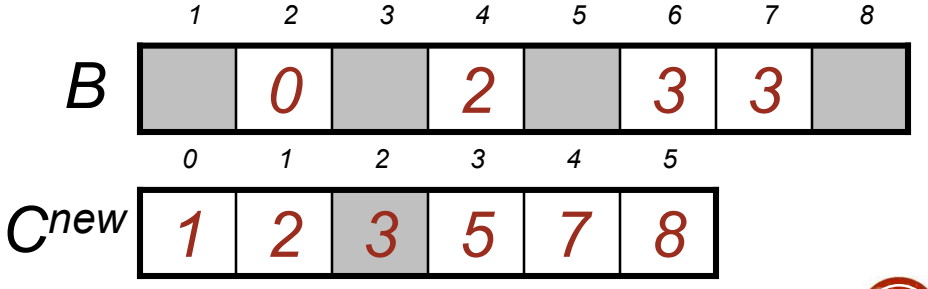

A
1 2 3 4 5 6 7 8
2 5 3 0 2 3 0 3


Cnew
0 1 2 3 4 5
2 2 4 7 7 8

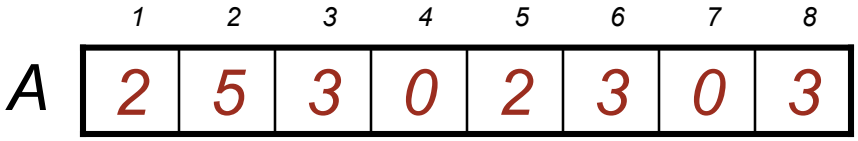

B
1 2 3 4 5 6 7 8
3
Cnew
0 1 2 3 4 5
2 2 4 6 7 8

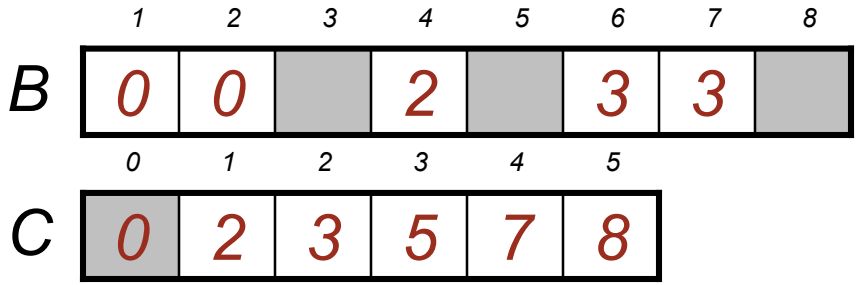

B
1 2 3 4 5 6 7 8
0 3
Cnew
0 1 2 3 4 5
1 2 4 6 7 8

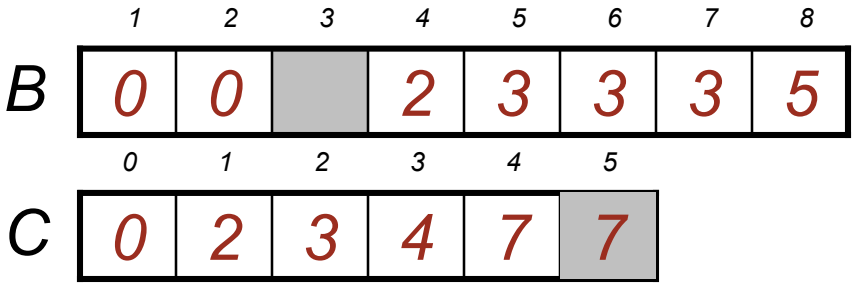

B
1 2 3 4 5 6 7 8
0 3 3
Cnew
0 1 2 3 4 5
1 2 4 5 7 8

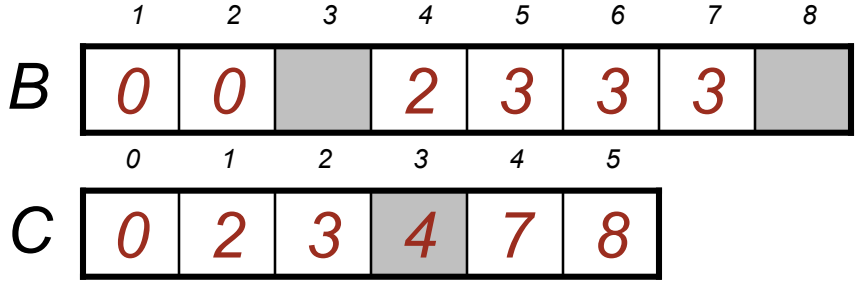

B
1 2 3 4 5 6 7 8
0 2 3 3
Cnew
0 1 2 3 4 5
1 2 3 5 7 8

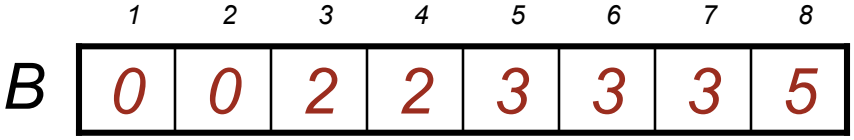
EXAMPLE (CONT.)











COUNTING SORT

```
1      CountingSort(A, B, k)
2            for I=1 to K
3                  C[I]= 0;
4            for J=1 to N
5                  C[A[J]] += 1;
6            for I=2 to K
7                  C[I] = C[I] + C[I-1];
8            for J=N down to 1
9                  B[C[A[J]]] = A[J];
10                       C[A[J]] -= 1;
```

Work through example: A={4 1 3 4 3}, K = 4

Counting sort

```
for i ← 1 to k
    do  C[i] ← 0;
for  j ← 1 to length [A]
    do  C[A[j]] ← C[A[j]] + 1;
for i ← 2  to k
    do  C[i] ← C[i] + C[i − 1];
for  j ← length [A]  downto  1
    do  begin
          B[C[A[j]]] ← A[j];
          C[A[j]] ← C[A[j]] − 1;
     end - for
```

$A: 3, 6, 4, 1, 3, 4, 1, 4$

$C: 2, 0, 2, 3, 0, 1$

$C: 2, 2, 4, 7, 7, 8$

$A: 3, 6, 4, 1, 3, 4, 1, \hat{4}$

$B:$, , , , , , ,4,

$C: 2, 2, 4, 6, 7, 8$

$A: 3, 6, 4, 1, 3, 4, \hat{1}, 4$

$B: \ , 1, \ , \ , \ , \ , 4,$

$C: 1, 2, 4, 6, 7, 8$

$A: 3, 6, 4, 1, 3, \hat{4}, 1, 4$

$B: \ , 1, \ , \ , \ , 4, 4,$

$C: 1, 2, 4, 5, 7, 8$

$A: 3, 6, 4, 1, \hat{3}, 4, 1, 4$

$B: \ , 1, \ , 3, \ , 4, 4,$

$C: 1, 2, 3, 5, 7, 8$

$A: 3, 6, 4, \hat{1}, 3, 4, 1, 4$

$B: 1, 1, \ , 3, \ , 4, 4,$

$C: 0, 2, 3, 5, 7, 8$

$A: 3, 6, \hat{4}, 1, 3, 4, 1, 4$

$B: 1, 1, \ , 3, 4, 4, 4,$

$C: 0, 2, 3, 4, 7, 8$

$A: 3, \hat{6}, 4, 1, 3, 4, 1, 4$

$B: 1, 1, \ , 3, 4, 4, 4, 6$

$C: 0, 2, 3, 4, 7, 7$

$A: \hat{3}, 6, 4, 1, 3, 4, 1, 4$

$B: 1, 1, 3, 3, 4, 4, 4, 6$

$C: 0, 2, 2, 4, 7, 7$

COUNTING SORT

- *Why don't we always use counting sort?*
- Because it depends on range *K* of elements
- *Could we use counting sort to sort 32 bit integers? Why or why not?*
- Answer: no, *K* too large (2^{32} = 4,294,967,296)