Week 7 Tutorial

CSCI 2100 Teaching Team

Outline

- Counting sort on key-value pairs
  - A linked list version (easy to understand)
  - Another version without using linked lists (fast in practice).

Counting Sort on Key-Value Pairs

- Input:
  - An array containing n key-value pairs, where each key is an integer from [1, U].

    E.g., (93, 1155123456)

- Output:
  - An array storing all the pairs in non-descending order of key.

Counting Sort on Key-Value Pairs

- Input:
  $\{\{9, v_1\}, \{7, v_2\}, \{2, v_3\}, \{6, v_4\}, \{2, v_5\}, \{7, v_6\}, \{1, v_7\}, \{2, v_8\}\}$
- Initially we have the following array

Input Array

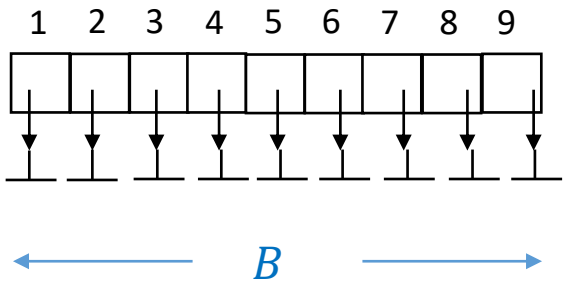
k_1	v_1	k_2	v_2	k_3	v_3	k_4	v_4	k_5	v_5	k_6	v_6	k_7	v_7	k_8	v_8
9	v_1	7	v_2	2	v_3	6	v_4	2	v_5	7	v_6	1	v_7	2	v_8

- Rearrange the elements so that their keys are sorted:

Sorted Array

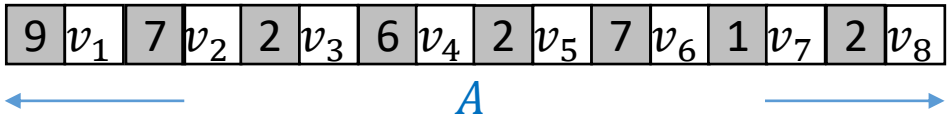
1	v_7	2	v_3	2	v_5	2	v_8	6	v_4	7	v_2	7	v_6	9	v_1

Counting Sort (Linked List Ver.)

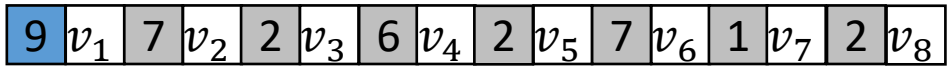


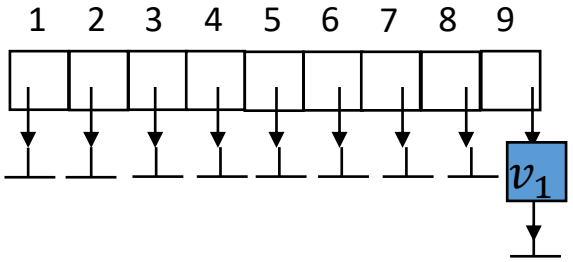
$\perp$: A null pointer

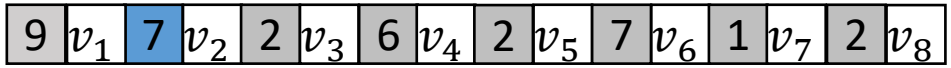
Compute B

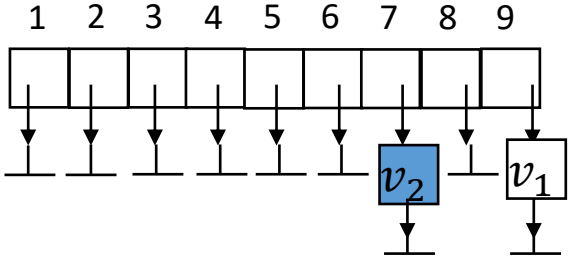


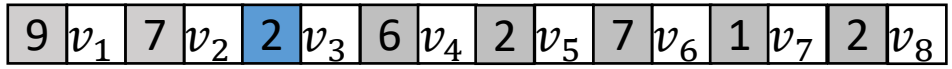
Counting Sort (Linked List Ver.)

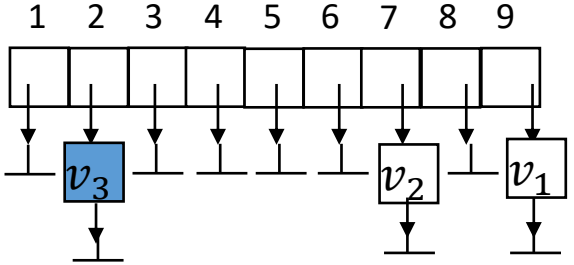




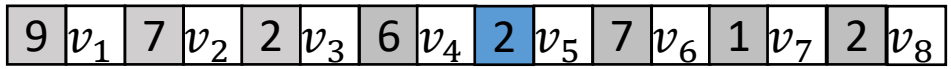


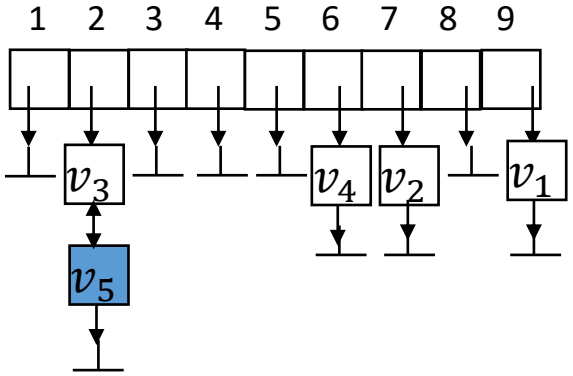


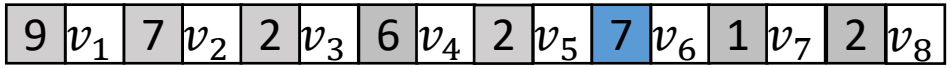


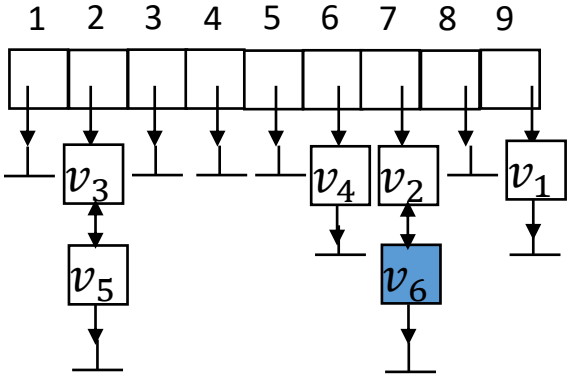


Counting Sort (Linked List Ver.)

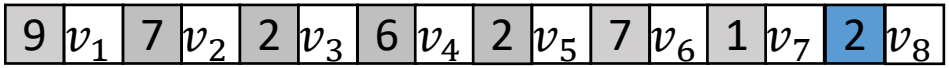


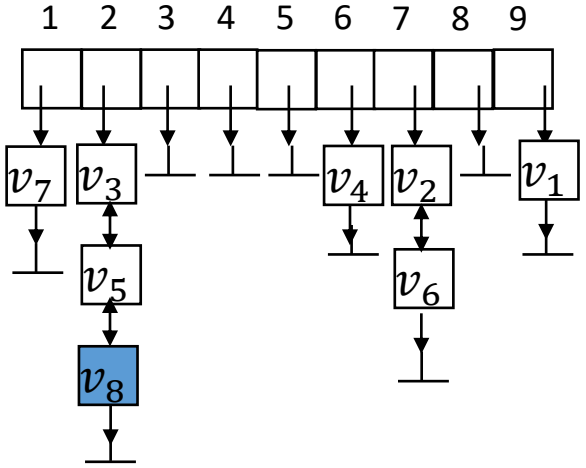




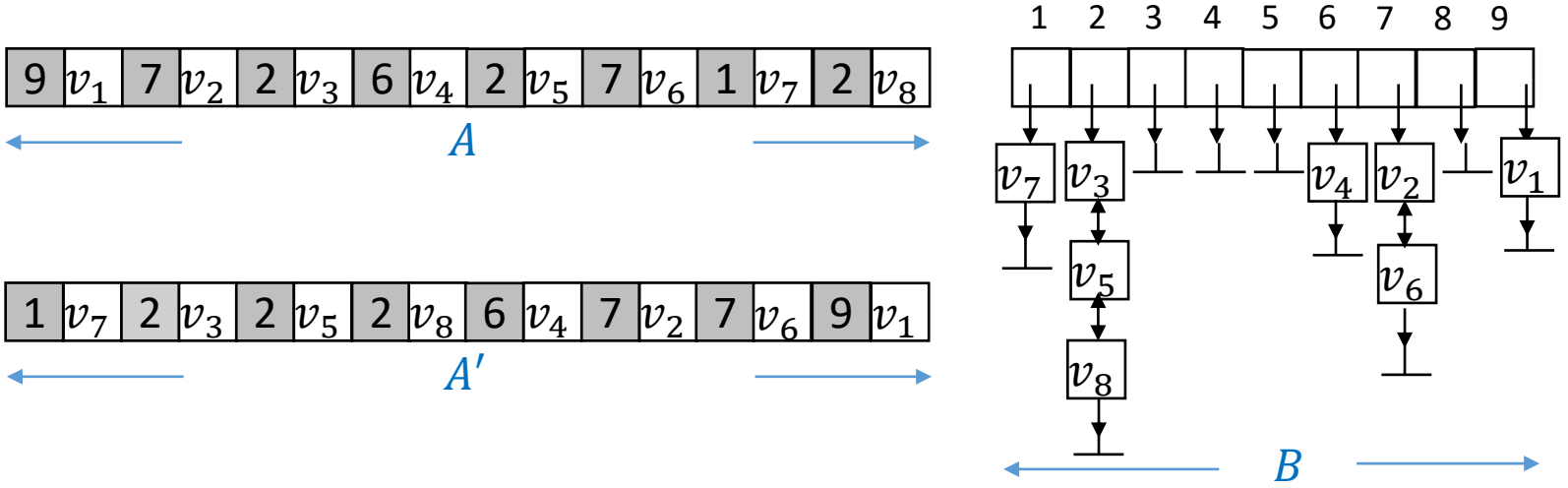


Counting Sort (Linked List Ver.)





Counting Sort (Linked List Ver.)



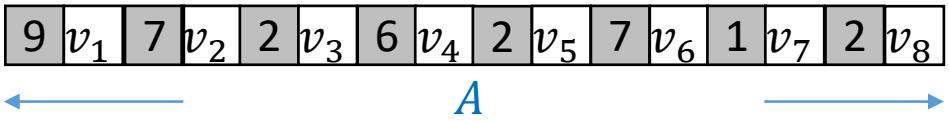
How do we produce the sorted array *A*'?

Scan array *B*. For each cell referencing a non-empty linked list, enumerate all the pairs therein.

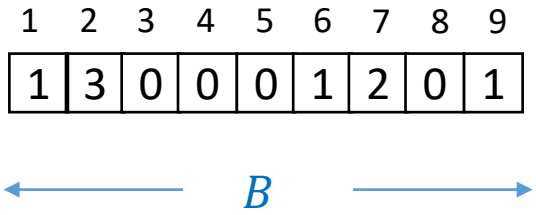
Overall time complexity: $O(n + U)$

Next, we will show how to solve
the problem without using linked lists.

Counting Sort (2nd Ver.)

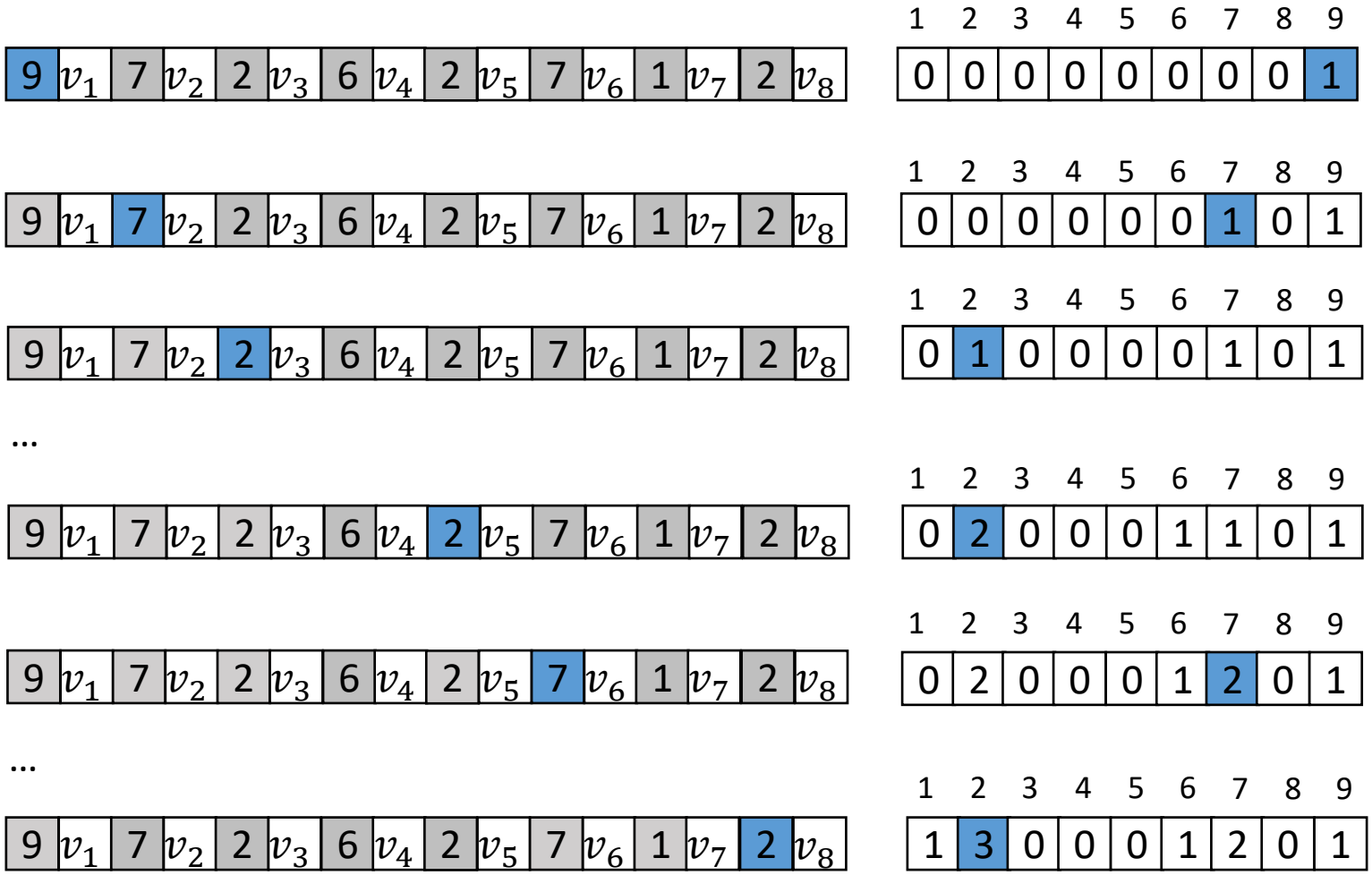


We first compute an array B to store the number of occurrences of each key.

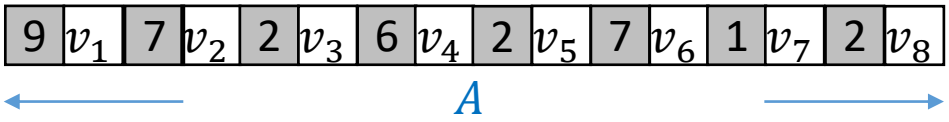


The next slide will explain how to do so.

Counting Sort (2nd Ver.)



Counting Sort (2nd Ver.)



Then we will change *B* from

1	2	3	4	5	6	7	8	9
1	3	0	0	0	1	2	0	1

to

1	2	3	4	5	6	7	8	9
1	4	4	4	4	5	7	7	8

Each $B[k]$ in the new array equals $\sum_{i=1}^{k} B[i]$ in the old array.
The next slide will explain how to do so in *O*(*U*) time.

Counting Sort (2nd Ver.)

For each $i \in [2, U]$, set $B[i] \leftarrow B[i] + B[i-1]$.

$i = 2$

1	2	3	4	5	6	7	8	9
1	3	0	0	0	1	2	0	1

1	2	3	4	5	6	7	8	9
1	4	0	0	0	1	2	0	1

$i = 3$

1	2	3	4	5	6	7	8	9
1	4	0	0	0	1	2	0	1

1	2	3	4	5	6	7	8	9
1	4	4	0	0	1	2	0	1

$i = 4$

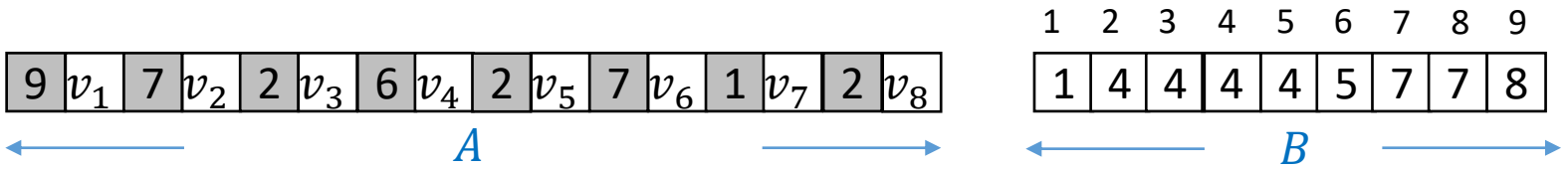
1	2	3	4	5	6	7	8	9
1	4	4	0	0	1	2	0	1

1	2	3	4	5	6	7	8	9
1	4	4	4	0	1	2	0	1

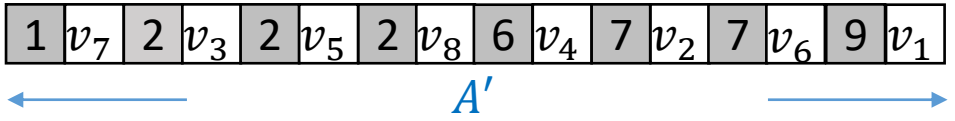
……

1	2	3	4	5	6	7	8	9
1	4	4	4	4	5	7	7	8

Counting Sort (2nd Ver.)



Our goal is to produce

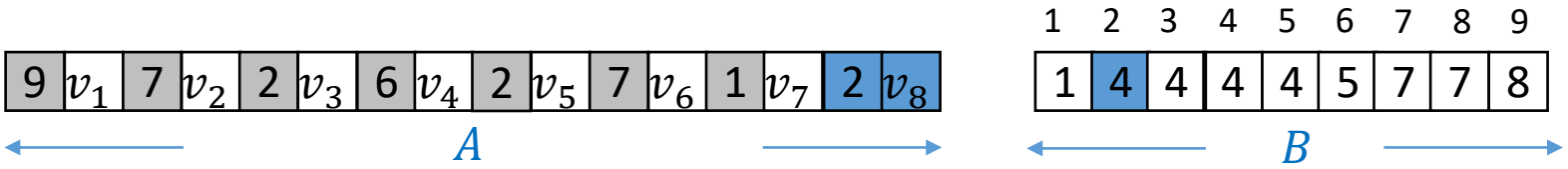


We will scan *A* backward and keep an invariant:

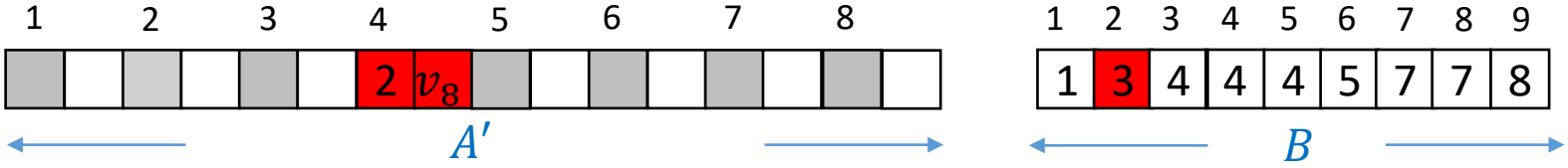
> If (k, v) is the rightmost pair among all the pairs with key *k* in the non-scanned part of *A*, the position of (k, v) in A' is $B[k]$.

Counting Sort (2nd Ver.)

Scan array A from right to left.

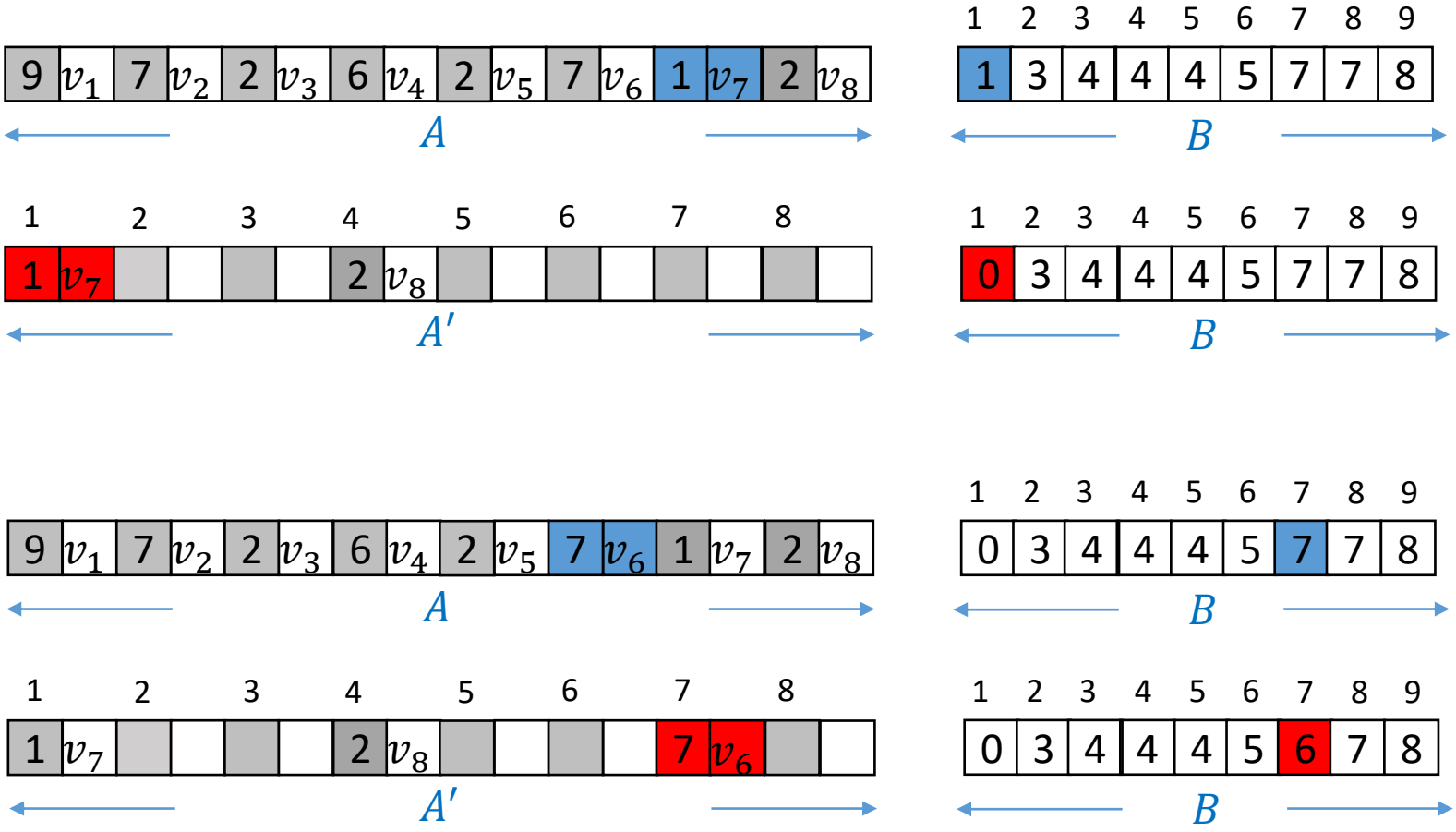


Insert the pair to A' and update B.

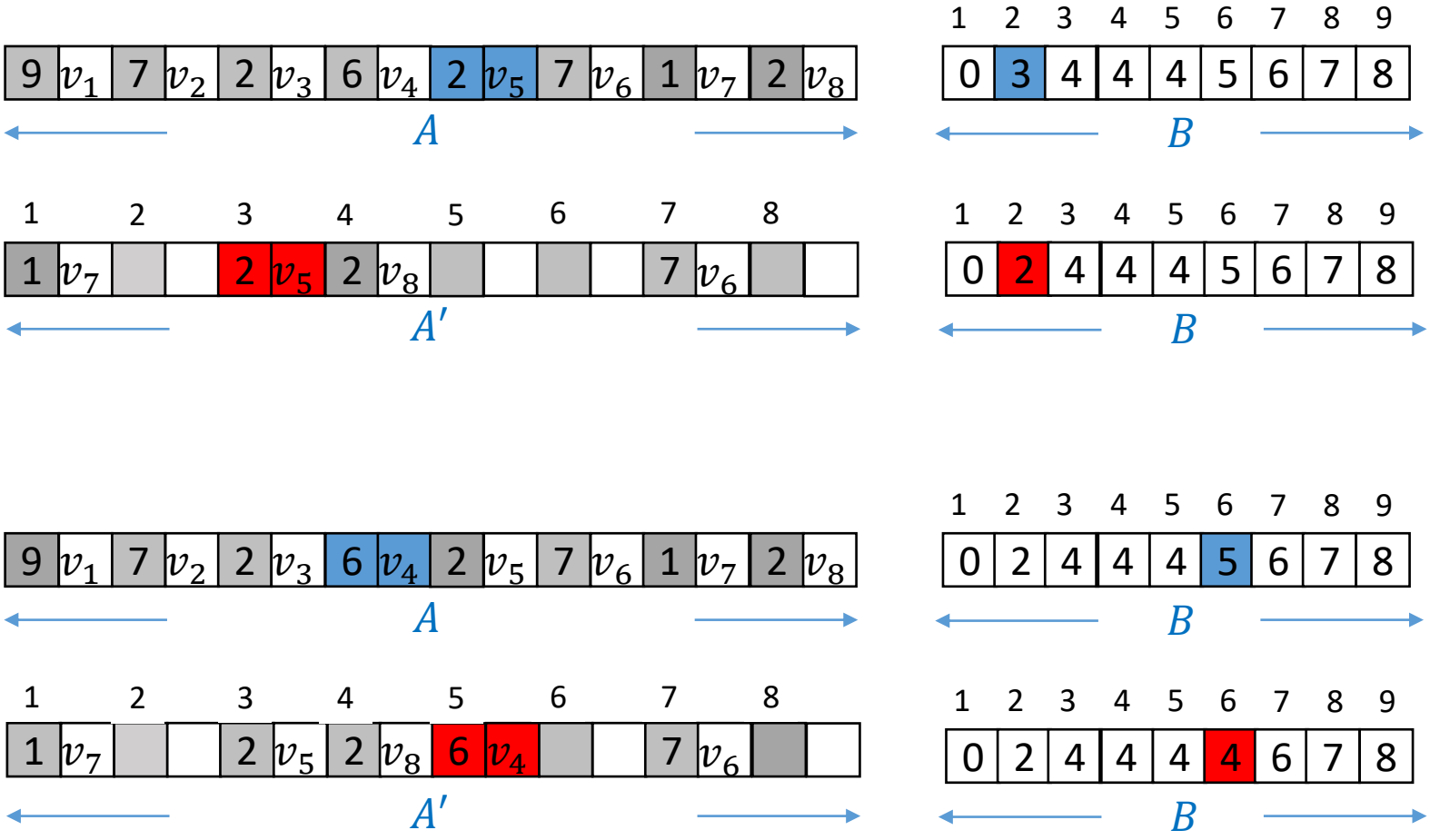


If (k, v) is the rightmost pair among all the pairs with key k in the non-scanned part of A, the position of (k, v) in A' is $B[k]$.

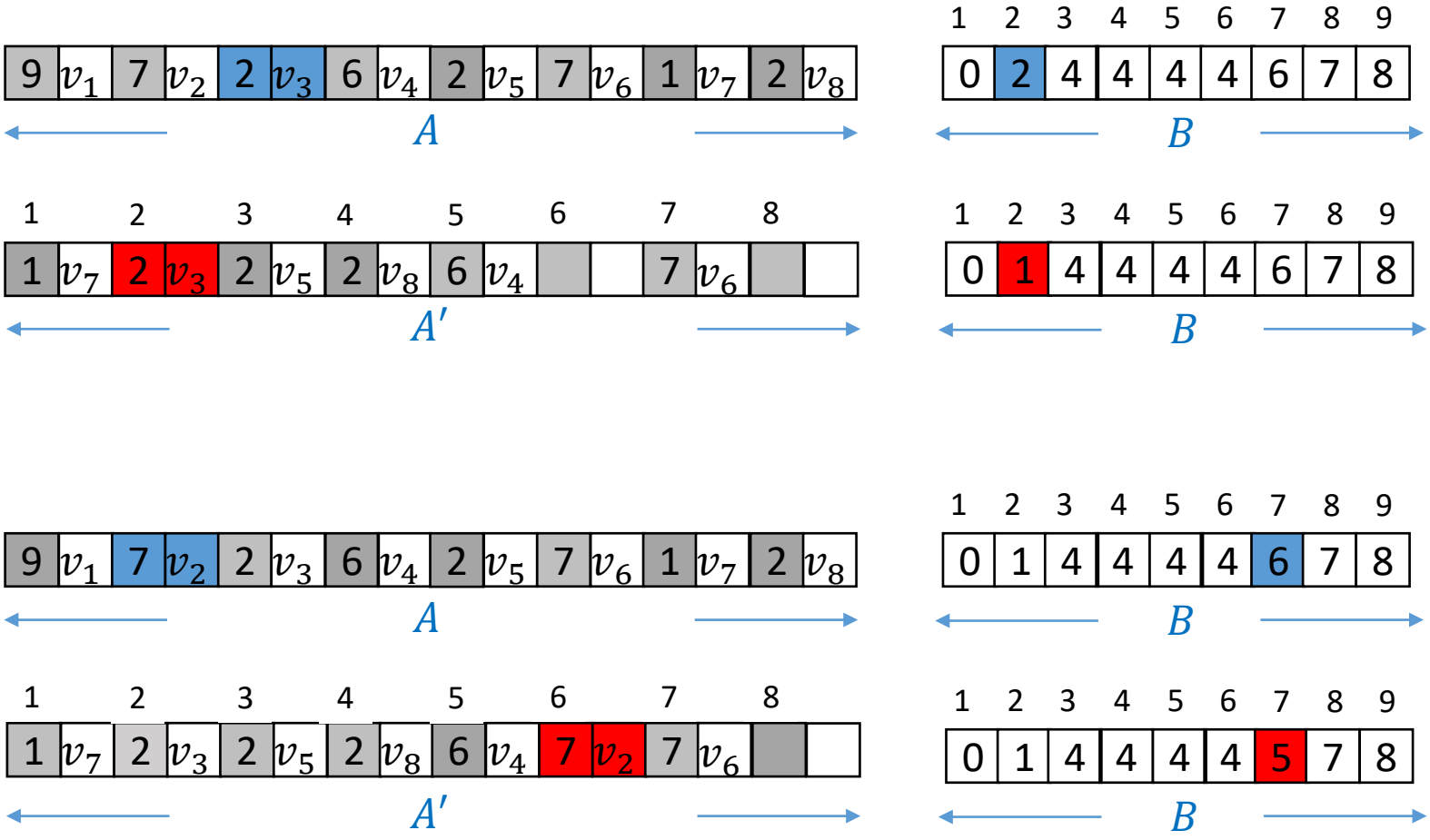
Counting Sort (2nd Ver.)



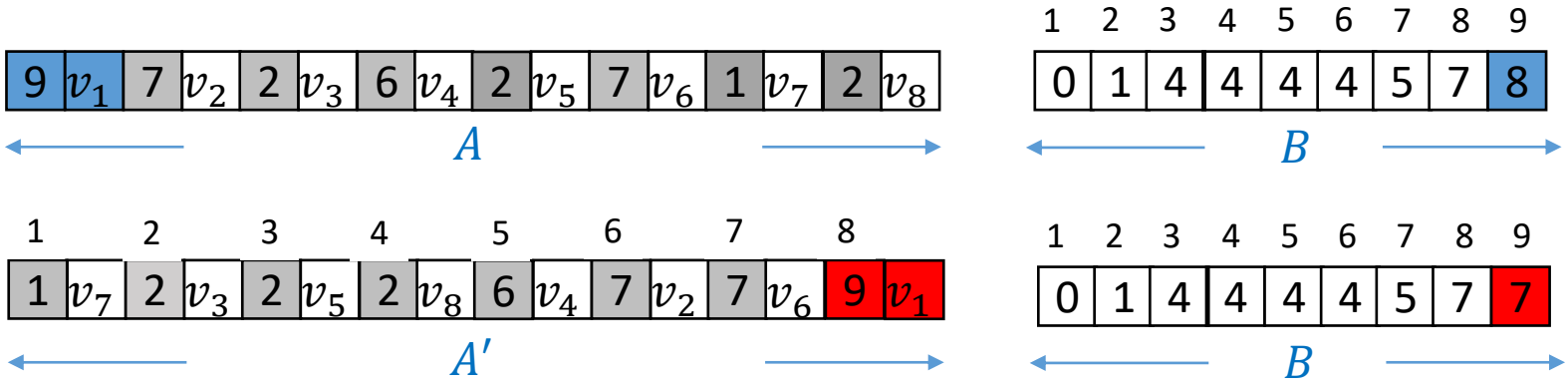
Counting Sort (2nd Ver.)



Counting Sort (2nd Ver.)



Counting Sort (2nd Ver.)



Overall time complexity: $O(n + U)$