Week 7 Tutorial

CSCI 2100 Teaching Team, 2021 Fall

Outline

- Counting sort – a linked list version
- Counting inversions (Ex List 5 Problem 4)

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 nondescending 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

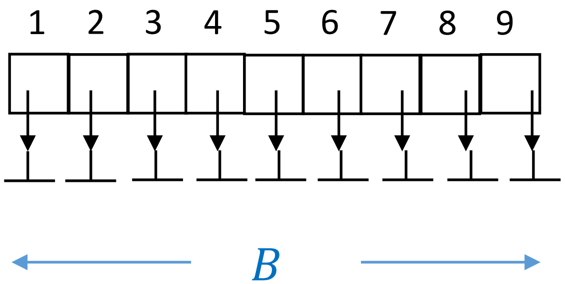
k_1	v_1	k_2	v_2	k_3	v_3	k_4	v_4	k_5	v_5	k_6	v_6	k_7	v_7	k_8	v_8
9	v_1	7	v_2	2	v_3	6	v_4	2	v_5	7	v_6	1	v_7	2	v_8

- Rearrange the elements so that their keys are sorted:

Sorted Array

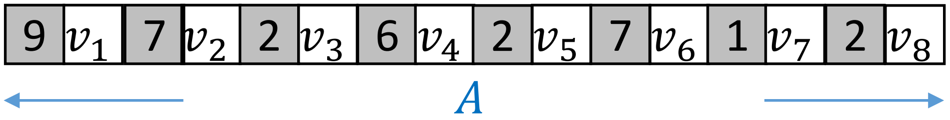
1	v_7	2	v_3	2	v_5	2	v_8	6	v_4	7	v_2	7	v_6	9	v_1

Counting Sort (Linked List Ver.)

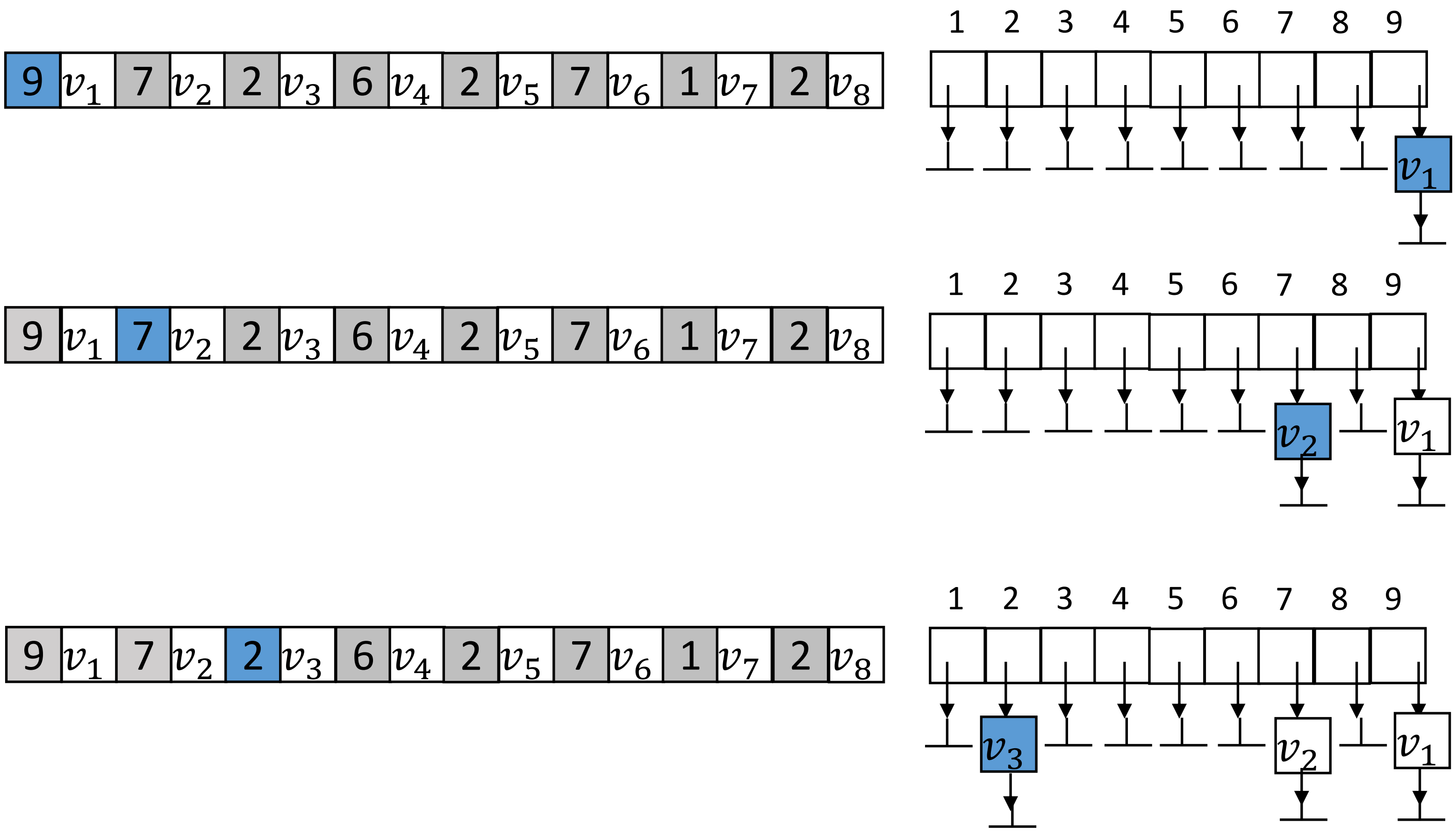


$\perp$: A null pointer

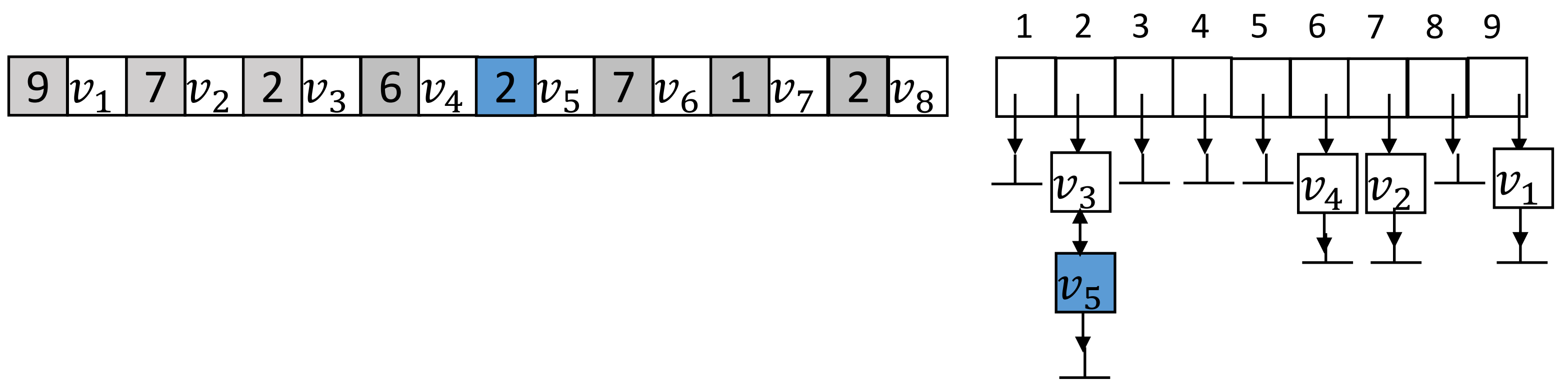
Compute B

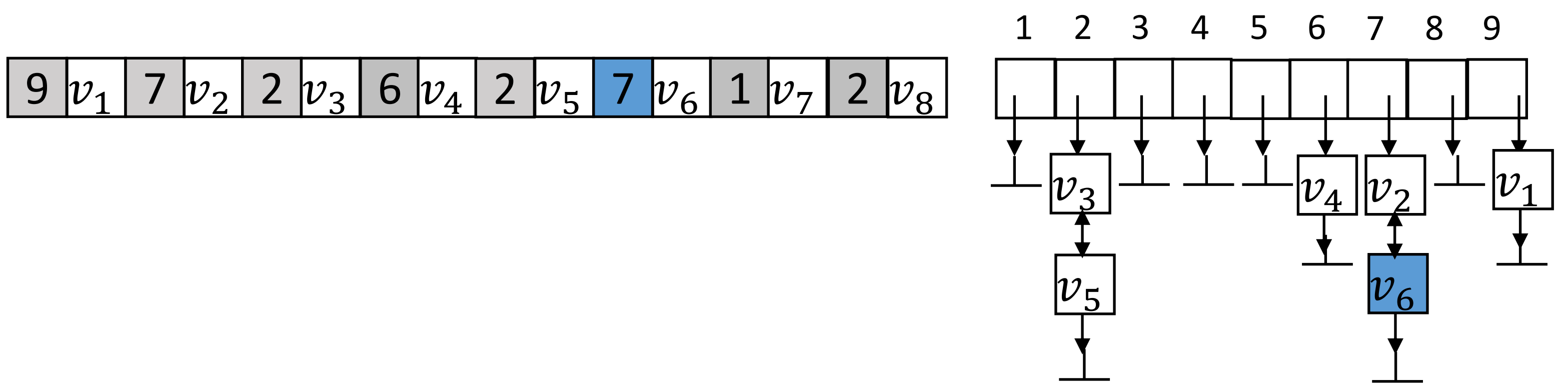


Counting Sort (Linked List Ver.)

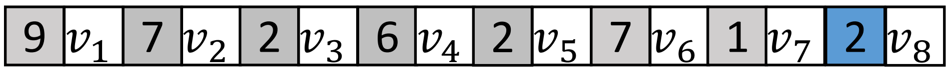


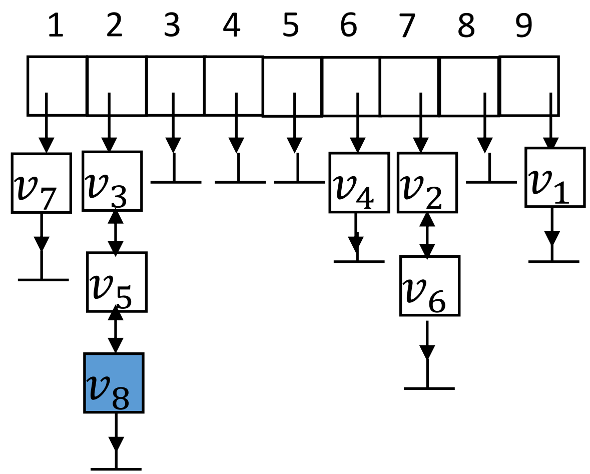
Counting Sort (Linked List Ver.)



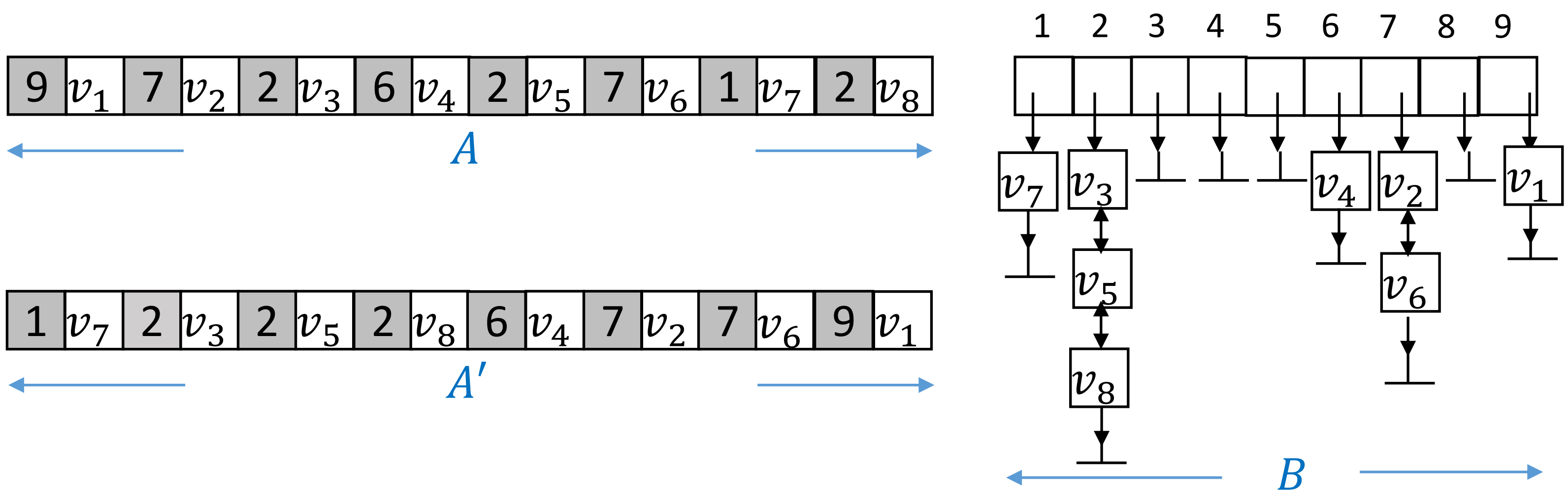


Counting Sort (Linked List Ver.)





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: $O(n + U)$

Our next problem will demonstrate a somewhat unusual way to apply recursion. To solve the problem, we will need to manually add new output requirements.

Counting Inversions

- Input:
  - Let A be an array of n integers (not necessarily sorted). We call (i, j) an inversion if $i < j$ but $A[i] > A[j]$.

- Goal: design an algorithm to count the number of inversions.

10	15	7	12

Inversions: (1, 3), (2, 3) and (2, 4)
Output: 3

Counting Inversions

- Input:
  - Let A be an array of n integers (not necessarily sorted).

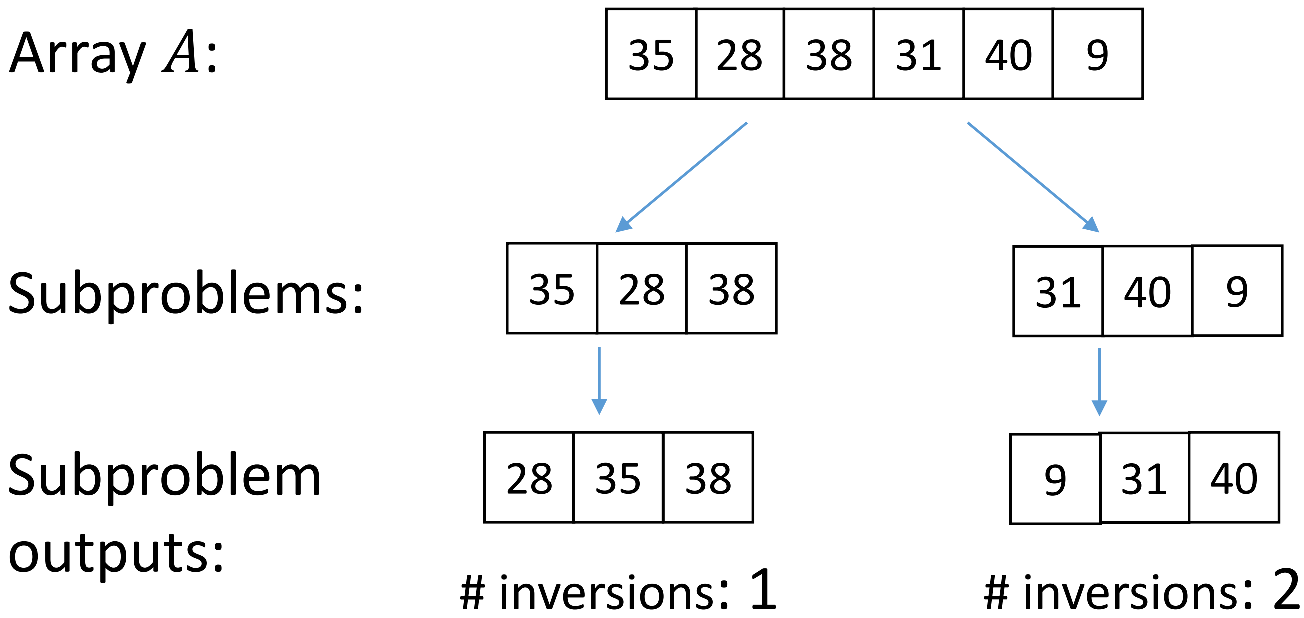
- Goal: design an algorithm to
  - count the number of inversions, and
  - sort A in ascending order.

10	15	7	12

Output: 3 inversions, and

7	10	12	15

Counting Inversions



It remains to count inversions (i, j) such that i comes from the first half and j from the second.

Counting Inversions

Crossing inversion: an (i, j) where i comes from the first half and j from the second.

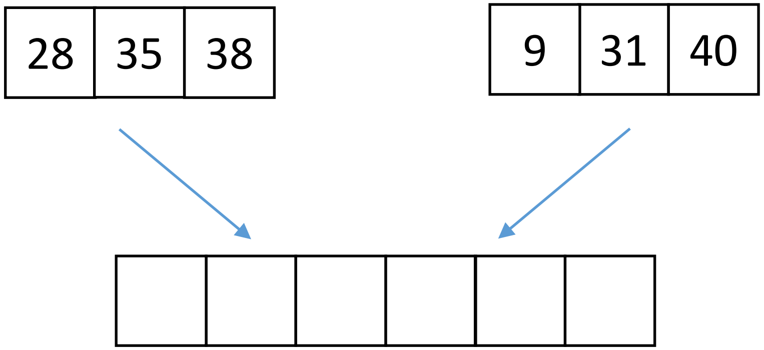
28	35	38

9	31	40

5 crossing inversions: (1,4), (2,4), (3,4), (2,5), (3,5)

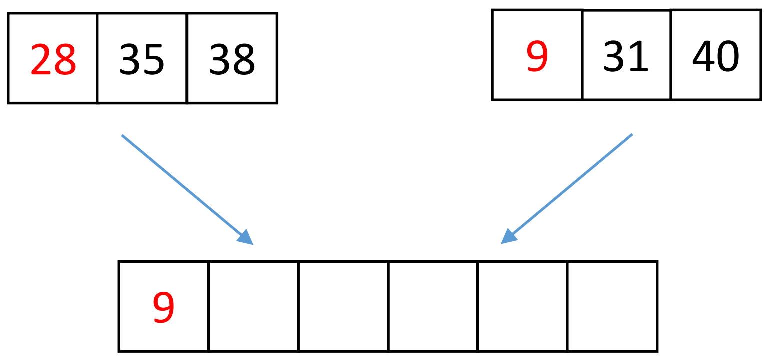
Counting Inversions

We can count the crossing inversions as we merge the two halves into a sorted array.



crossing
Inversions: 0

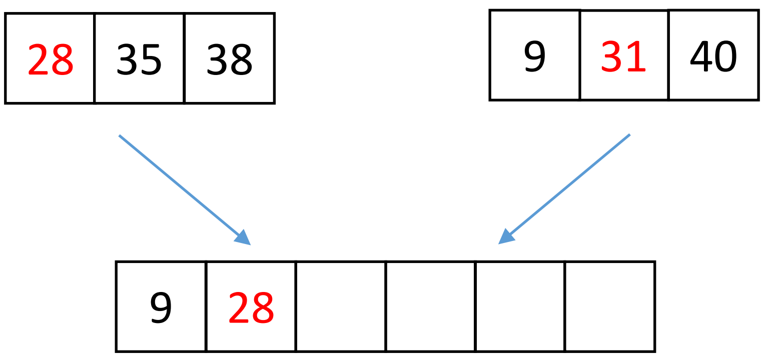
Counting Inversions



crossing
Inversions: 3

Think: why

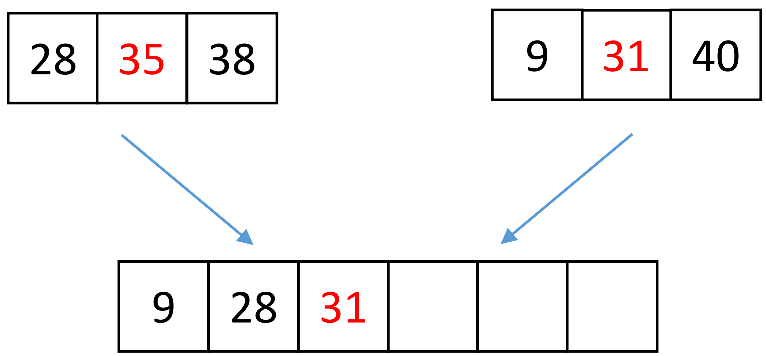
Counting Inversions



crossing
Inversions: 3

Think: why no change?

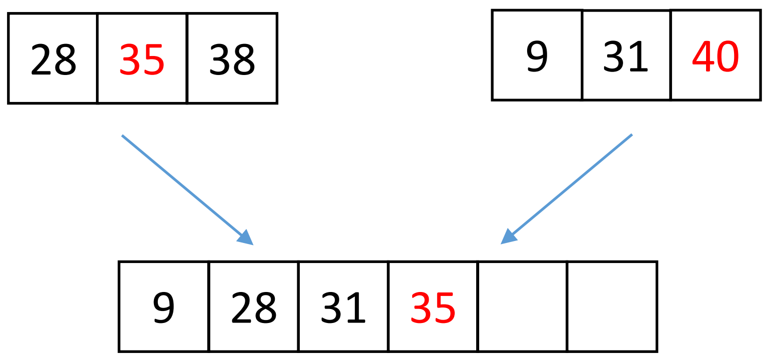
Counting Inversions



crossing
Inversions: 5

In general, every time we move an element from the second half to the merged array, we increase the counter by the number of remaining elements in the first half.

Counting Inversions



\# crossing
Inversions: 5

no change if an element from the first half enters the merged array

Counting Inversions

- $T(1) = O(1)$
- $T(n) \leq 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)$
- Solving the recurrence gives $T(n) = O(n\ log\ n)$