More Counting Sort and Sorting-by-Key

Tony Gong

ITEE
University of Queensland

In the lectures last week we looked at the counting sort algorithm for sorting a set of integers that come from some specific domain, i.e. every integer is in some range $[1, U]$.

Today we will look at modifications of counting sort to solve two variations of the problem:

1. What if instead of a set we had a multi-set (where there can be duplicate elements)?
2. What if instead of sorting integers we wanted to sort arbitrary objects based on integer keys?

For the first variation, we will re-define the problem as follows:

The Sorting Problem (in a Small Domain) on a Multi-Set

Problem Input:

A multi-set S of n integers (each in the range $[1, U]$) is given in an array of length n. The values of n and U are inside two registers in the CPU.

Goal:

Store S in an array where the elements are arranged in non-decreasing order.

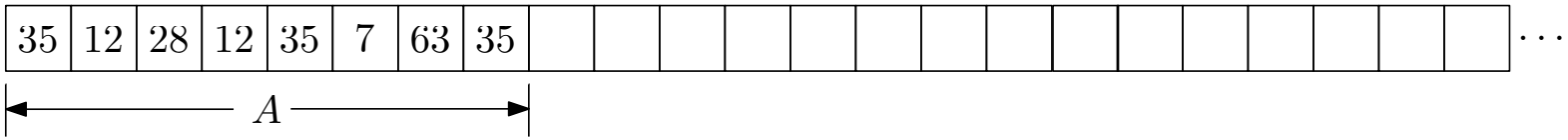
Recall that in the counting sort algorithm on a regular set S, (where there are no duplicate elements), we make use of an array B of length U (initialised to all 0's) and for every $x \in S$ we set $B[x] = 1$ to mean that the element appears in S. At the end we make a linear scan through B and collate the elements marked with a 1.

To deal with duplicate elements, one idea is to think of B now as counters instead of flags. B still gets initialised to 0, but now gets incremented each time we see an element x in the multi-set S. At the end, the value of $B[x]$ will then give the number of times x appears in S.

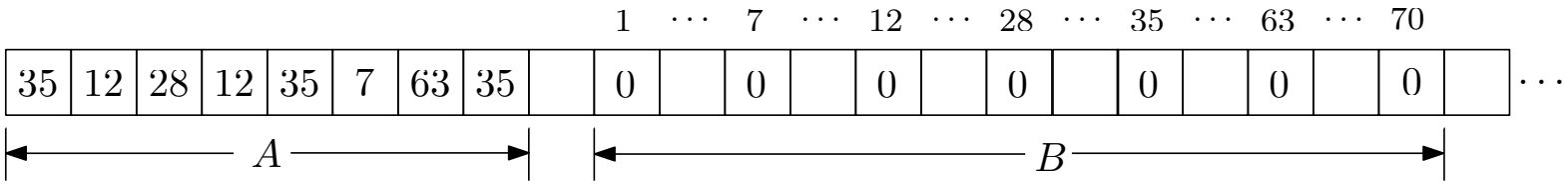
The running time of this algorithm is obviously the same as the one for normal sets, $O(n + U)$.

Example

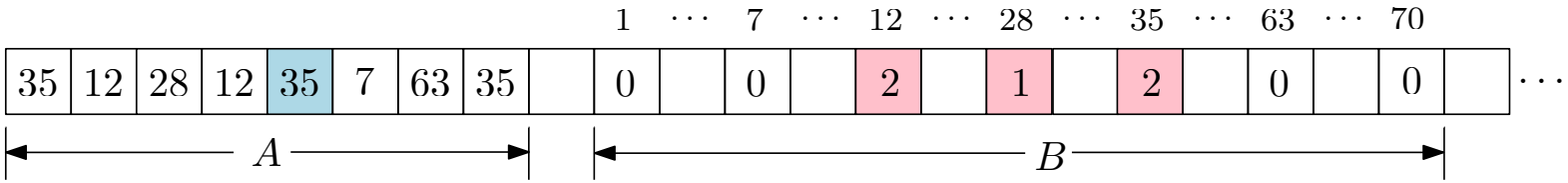
At the beginning



Initialise array B

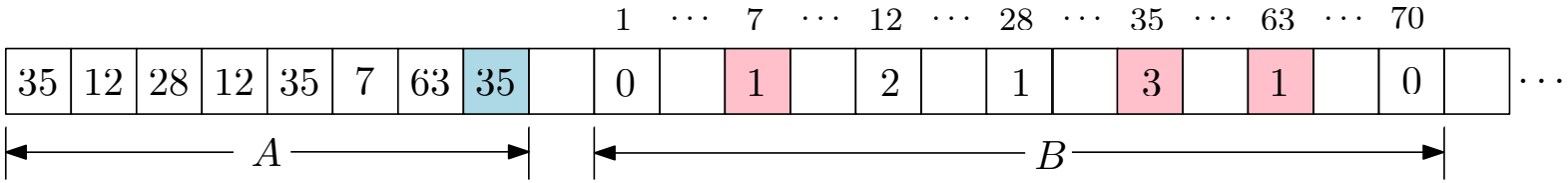


After examining $A[5]$

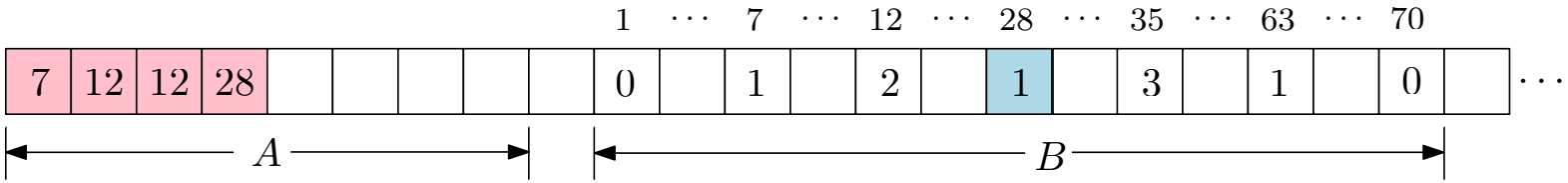


Example

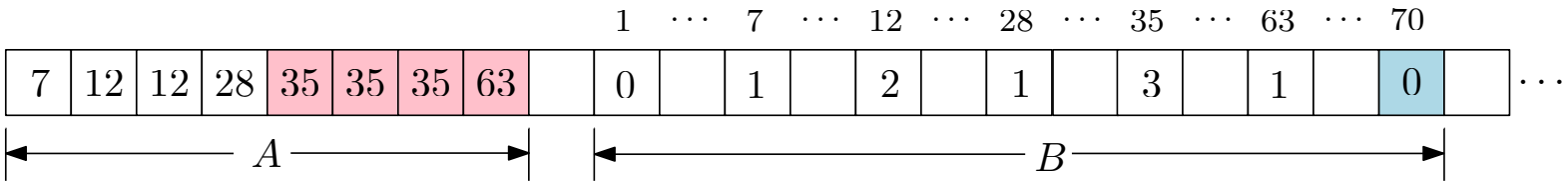
At the end of scan through A



After examining $B[28]$



End result (at the end of scan through B)



For the second variation, we will again extend the problem definition to the following:

The Sort-by-Key Problem (in a Small Domain)

Problem Input:

A multi-set S of n 2-tuples (of integers) is given in an array of length $2n$. We will refer to each 2-tuple as a key-value pair, where the first position gives the key and the second position gives the value. All keys are in the range $[1, U]$. The values of n and U are inside two registers in the CPU.

Goal:

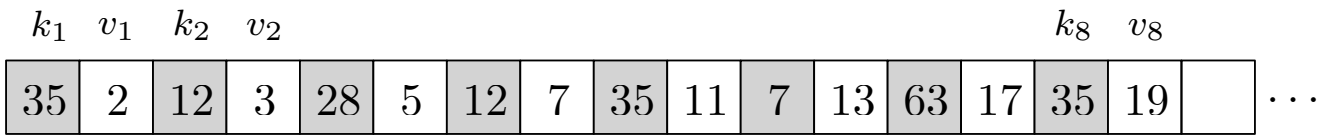
Store S is an array where the elements are arranged in non-decreasing order by key.

Example

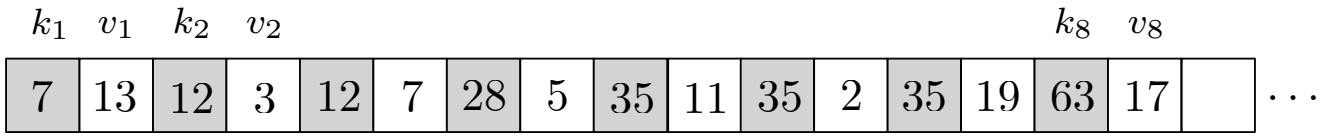
Consider if the multi-set S is given by:

$$S = \{(35, 2), (12, 3), (28, 5), (12, 7), (35, 11), (7, 13), (63, 17), (35, 19)\}$$

Initially we will have the following array:



And we want to rearrange the elements so that the keys are sorted:



The counting sort algorithm on multi-sets proposed will require modification in order to work for this problem because we also need to track information regarding the values.

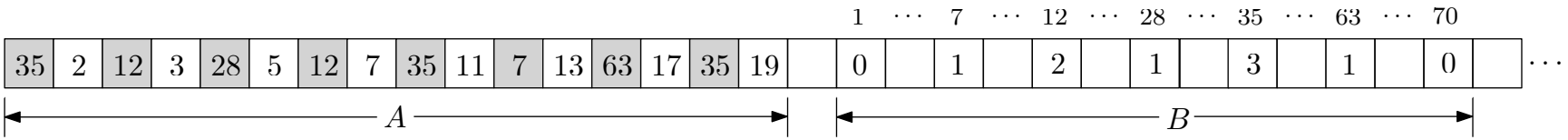
We will look at two strategies for dealing with this:

1. Computing indices into the final sorted array.
2. Using linked lists.

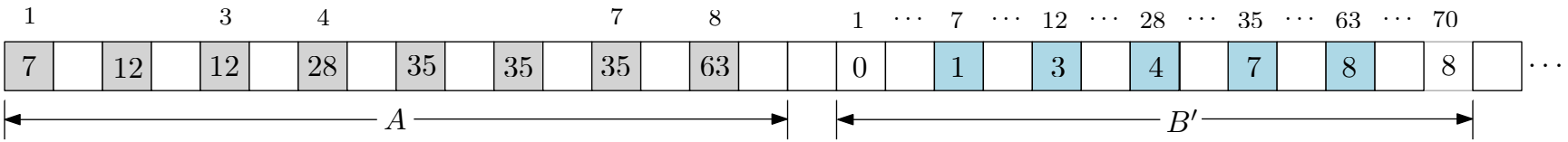
Strategy 1: Computing Indices

Example

Using the same example, after computing B we will reach



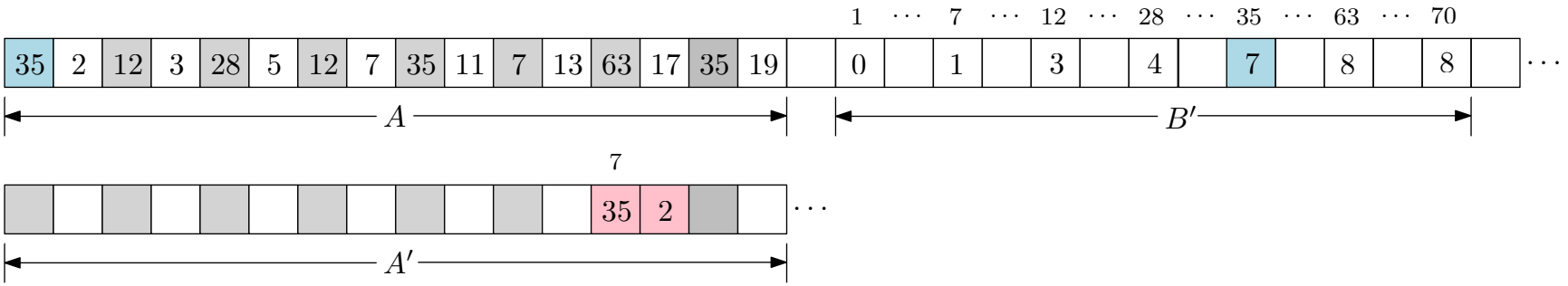
If we look at the cumulative sums of B in conjunction with the keys in sorted order:



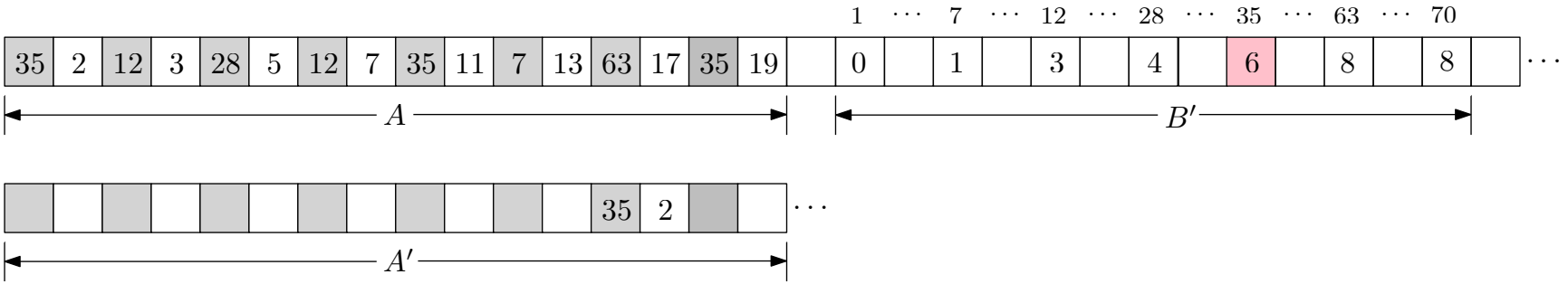
Notice that the cumulative sums encodes the last index a particular key will be found in the final sorted array.

Example

Thus we can build up a new array A' by repeating the following: for a key-value pair (k, v) in A, move it to the $B'[k]$th position in A'

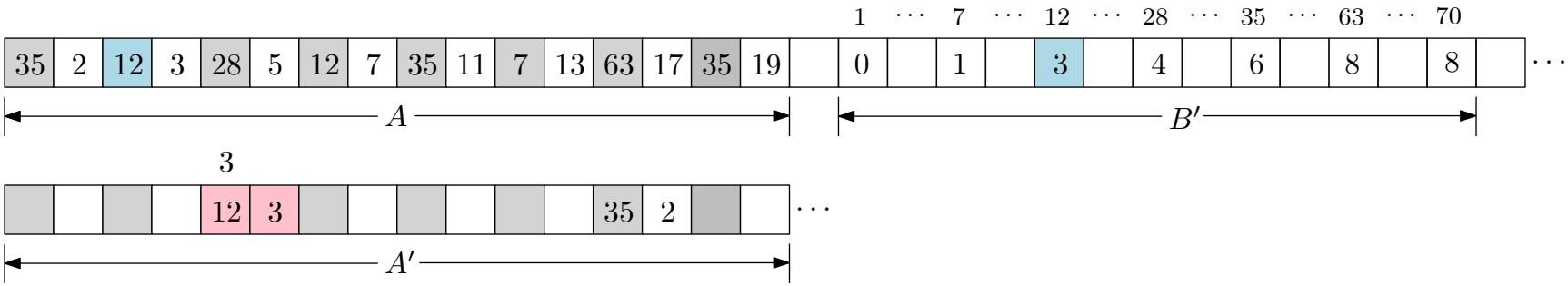


And then decrement the value in B' (to ensure that it always is pointing at a valid, empty position in A')

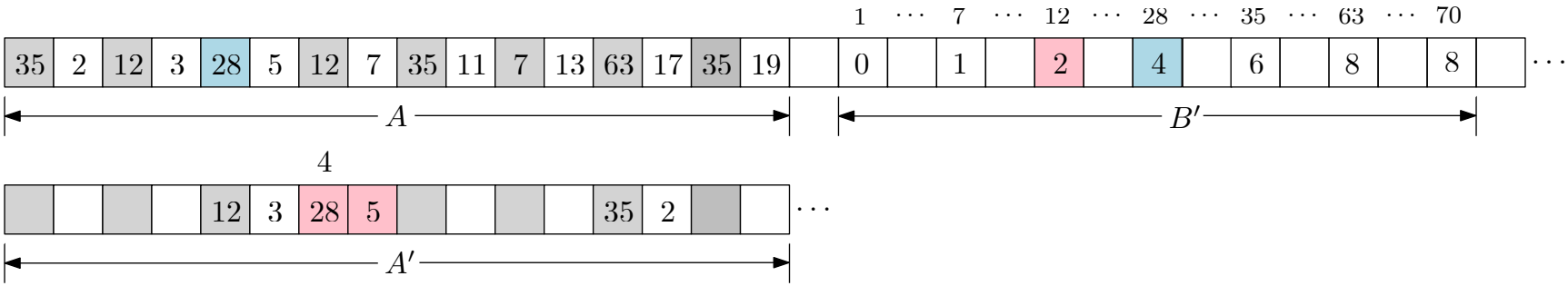


Example

After the second iteration



And the third



And eventually at the end of this process A' will be sorted by key.

Running Time

We can analyse the running time of this algorithm by looking at each step individually:

Step 1: scanning through A to compute B takes $O(n)$ time.

Step 2: computing the cumulative sum B' takes $O(U)$ time. (Think about how you would do this!)

Step 3: scanning through A and using B' to copy elements over into A' takes $O(n)$ time.

Thus overall the running time of the algorithm is $O(n + U)$.

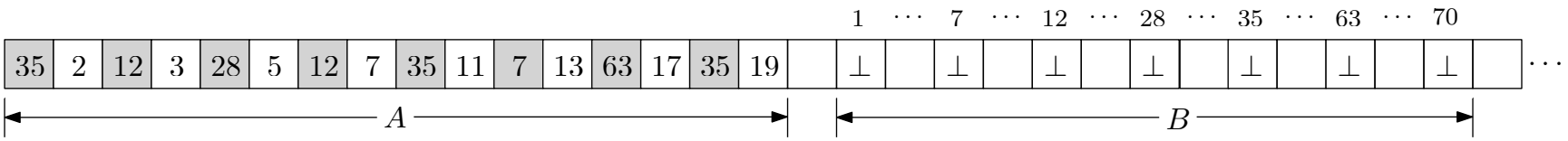
Strategy 2: Linked List

An alternate, more elegant solution is to make use of linked lists. B will still be an array of length U, but now each element will be a linked list. For every key-value pair (k, v) in A, we append the value onto the tail of the list at $B[k]$ (which we can do in $O(1)$ as long as we have a reference to the tail of the list).

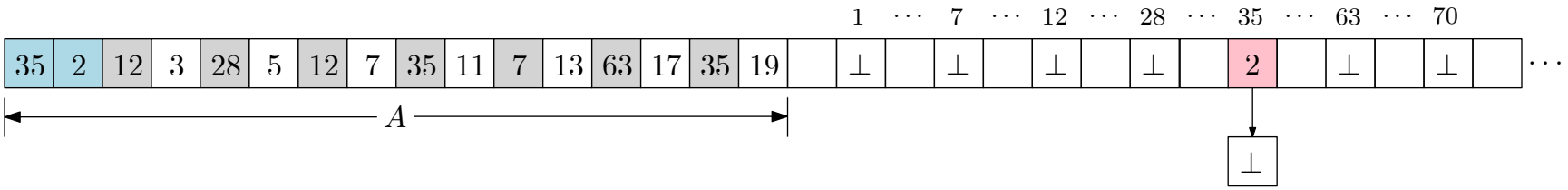
At the end we scan through B and collate all the elements into an array.

Example

Again using our familiar example, we begin with B initialised such that every element is an empty list (to abstract the details of the linked list I have abused notation in the ensuing diagrams)

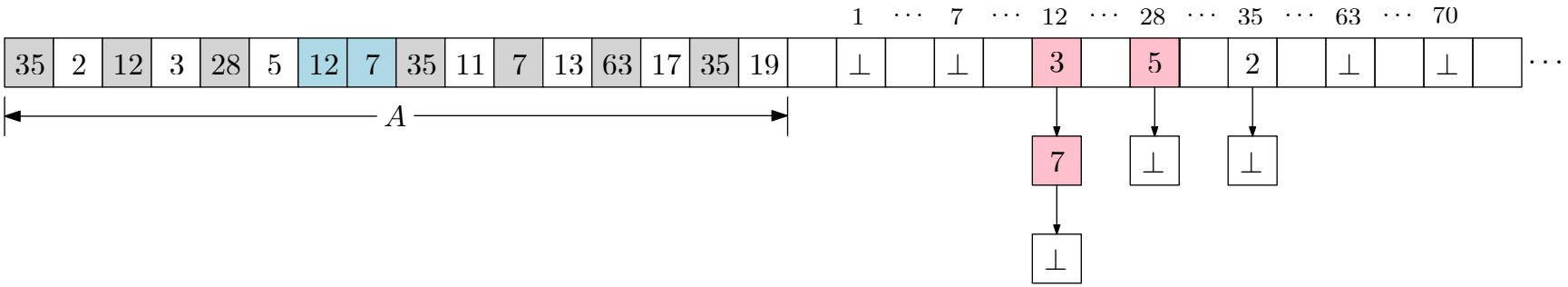


After looking at the first element in A, we append the associated value 2 onto the tail of the linked list at index 35

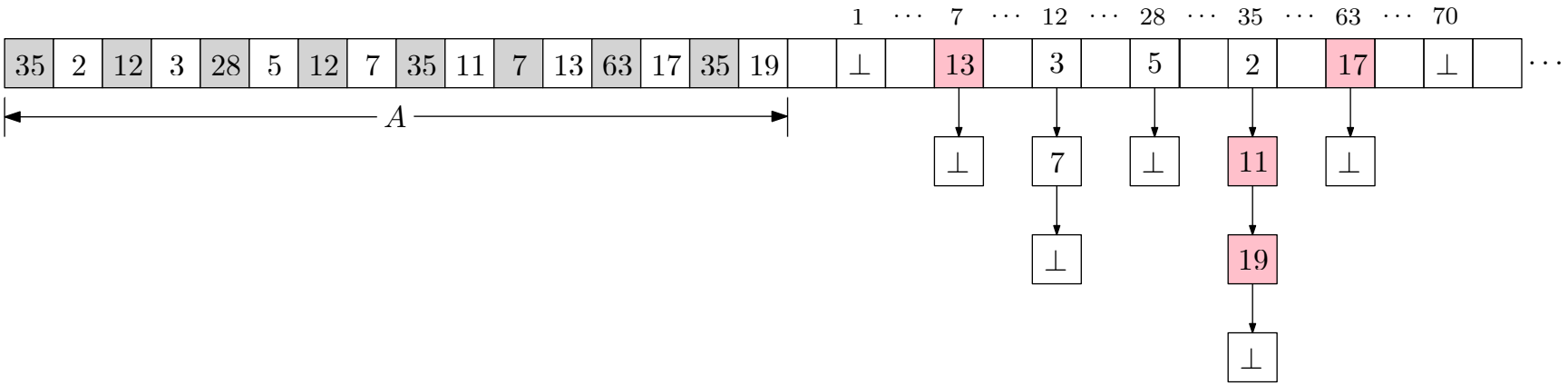


Example

After examining the 4th element in A



After examining all elements in A



And then it is a trivial matter of collating the lists in B into an array.

Running Time

We anaylse the running time in the same way as the first algorithm, by looking at the cost of each step:

Step 1: scanning through A and appending elements to the tail of a linked list in B takes $O(n)$ time.

Step 2: traversing the linked lists in B one by one and copying elements out to an array takes $O(n + U)$ time.

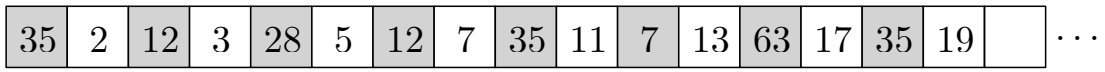
Thus overall the running time is again $O(n + U)$.

Stability of Sorting

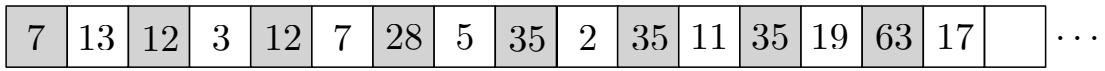
A sorting algorithm is said to be stable if it preserves the ordering of elements with the same key in the sorting process.

Example

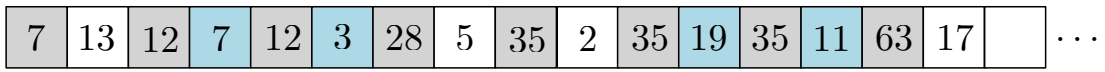
If the following is the array of unsorted elements:



Then a stable sorting algorithm is guaranteed to output:



Whereas a non-stable sorting algorithm may output:



Stability of Sorting

Of the two non-comparison-based algorithms we examined to solve the sort-by-key problem, the first one that computes indices is not a stable sort while the second using linked list is.

As an exercise, modify the first algorithm so that the sort is stable.