Counting Sort

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

We already know that sorting n integers can be done in $O(n \log n)$ time. Today, we will see a **special case** of the sorting problem where the integers come from a small domain.

Sorting in a Small Domain)

Problem Input:

A set S of n integers is given in an array of length n. Every integer is in the range of $[1, U]$ where $U \geq n$.

Goal:

Produce an array that stores the integers of S in ascending order.

Counting Sort

Step 1: Let A be the array storing S. Create an array B of length U. Initialize B by setting all its cells to 0.

Step 2: Carry out the following for every $i \in [1, n]$: set $B[A[i]] = 1$.

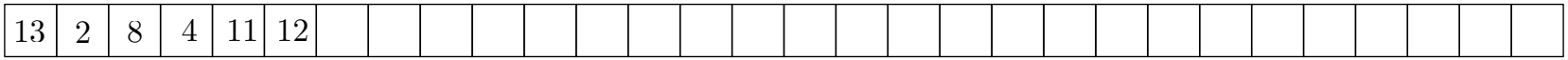
Step 3: Generate the sorted order as follows:

for $x = 1$ to U

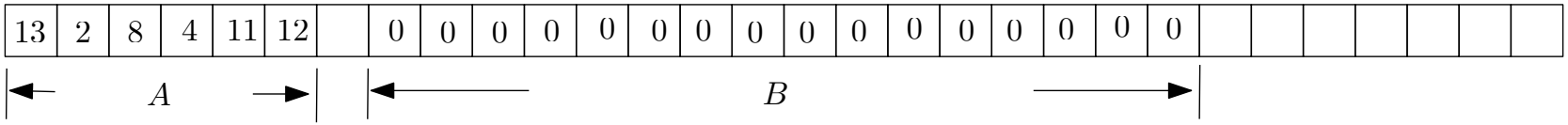
if $B[x] = 1$ **then** append integer x to A.

Example

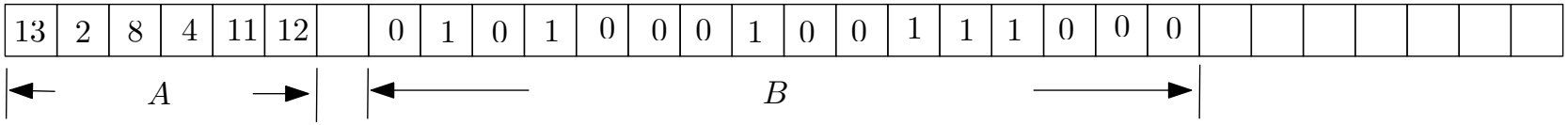
At the beginning



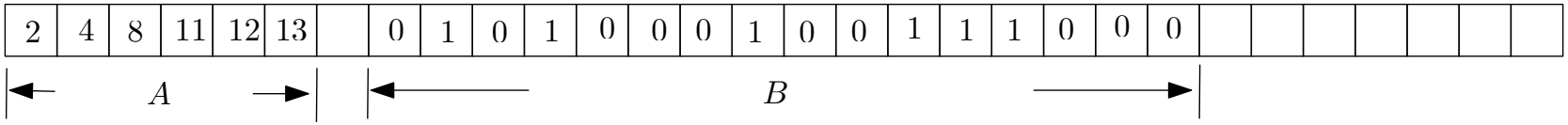
Initialize array B (assuming $U = 16$)



Setting n cells of B to 1



Final sorted list



Analysis of Counting Sort

Steps 1 and 3 take $O(U)$ time.
Step 2 takes $O(n)$ time.

Therefore, the overall running time of counting sort is $O(n + U) = O(U)$.

For small $U = O(n)$ (e.g., $1000n$), the counting sort runs in $O(n)$ time.

It is important to note that counting sort does **not** improve merge sort in general! There are two reasons for this.

- $O(n+U)$ is **incomparable** to $O(n \log n)$. When $U = O(n)$, counting sort is faster, but when $U = \Omega(n^2)$, merge sort is faster.

- Counting sort tackles only a special version of the problem solved by merge sort. The former is designed to sort integers, whereas the latter can be used to sort any items (e.g., strings) on which comparisons are well defined.