Linear Time Sorting in a Polynomial Domain

[Notes for ESTR2102]

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Recall that counting sort is able to sort n integers in the range from 1 to U in $O(n + U)$ time. The running time is expensive for large U. We will significantly improve this by describing how to sort in $O(n)$ time for any $U \leq n^c$, where c is a constant (e.g., 10).

The new algorithm is called radix sort.

Without loss of generality, we will consider that n is a power of 2 (why no generality is lost?). Hence, every integer can be represented by $c \log_2 n$ bits (in binary form), which we denote as $b_{c \log_2 n} b_{c \log_2 n-1} \ldots b_2 b_1$, where b_1 is the least significant bit.

For every integer $b_{c \log_2 n} b_{c \log_2 n-1} \ldots b_2 b_1$, we divide the bits into c disjoint chunks, each of which contains $\log_2 n$ bits:

- The first chunk contains the right most $\log_2 n$ bits, namely, $b_{\log_2 n} b_{\log_2 n-1} \ldots b_1$.
- The second chunk contains the next $\log_2 n$ bits, namely, $b_{2 \log_2 n} b_{2 \log_2 n-1} \ldots b_{\log_2 n+1}$.
- ...
- The last chunk contains the left most $\log_2 n$ bits, namely, $b_{c \log_2 n} b_{c \log_2 n-1} \ldots b_{(c-1) \log_2 n+1}$

For any integer $x = b_{c\log_2 n} b_{c\log_2 n-1} \ldots b_2 b_1$, and any $i \in [1, c]$, we can obtain the i-th chunk of x as follows:

- Calculate $y = x \mod n^i$. The binary form of y corresponds to the rightmost $i \cdot \log_2 n$ bits of x. If $i = 1$, then return y. Otherwise, proceed to the next step.
- Return y/n^{i-1} (integer division).

We can prepare $n, n^2, n^3, \ldots, n^c$ in advance to ensure that y can be calculated in $O(1)$ time. The values of $n, n^2, n^3, \ldots, n^c$ can be calculated in $O(c) = O(1)$ total time.

Example

Suppose that $c = 4$, $n = 16$, and $x = 011011000010$ (i.e., 1730 in decimal). To get its 2nd chunk, we do:

- $y = x \mod n^2 = 1730 \mod 256 = 194$
- We return $y/n = 194/16 = 12$.

This is correct because 12 is 1100 in binary, namely, the 2nd chunk of x.

Stable sorting: The input is a set S of n key-value pairs of the form (k, v), where k is the **key** and v is the **value**. These pairs are given in an array A. Every key is in the range from 1 to n.

The goal is to produce an array B that stores all the pairs in non-descending key order. Furthermore, the sorting must be **stable** in the following sense. For any two pairs (k_1, v_1) and (k_2, v_2) such that $k_1 = k_2$, if (k_1, v_1) is positioned earlier than (k_2, v_2) in A, this must also be true in B.

We can adapt counting sort easily to solve the above problem in $O(n)$ time (details left to you).

Radix Sort

We now return to our problem. Let A be the input array of n integers. We sort them by executing the stable counting sort algorithm of the previous slide c times:

- Stable-sort A according to their 1st chunks. Replace A with the array output.
- Stable-sort A according to their 2nd chunks. Replace A with the array output.
- ...
- Stable-sort A according to their c-th chunks. Replace A with the array output.

Return the final A.

Analysis

Correctness guaranteed by stability.

Running time clearly $c \cdot O(n) = O(n)$.