Lecture 14: Count Sort and Radix Sort (8.2-8.3)

CS303: Algorithms

Last update: February 9, 2014

1 Review

- Average case efficiency of quick sort $\Theta(nlgn)$
- Comparison-based sorting algorithm efficiency lower bound is $\Theta(nlgn)$

2 Linear time sorting

These algorithms need elements to be integers and elements are integers within a certain range. We will introduce 2 algorithms

2.1 Count sort

Depends on a key assumption: **members to be counted are integers in** $0, 1, ..., k$

Input: $A[1...n]$ where $A[j] \in 0, 1, ...k$ for $j = 1, 2, ..., n$. Array A and values n and k are given as parameters.

Output: $B[1...n]$ sorted. B is assumed to be already allocated and is given as a parameter.

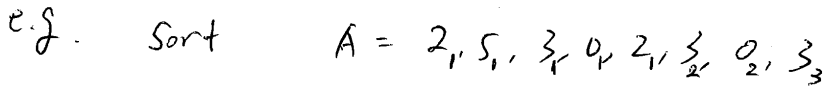
Auxiliary storage: $C[0...k]$

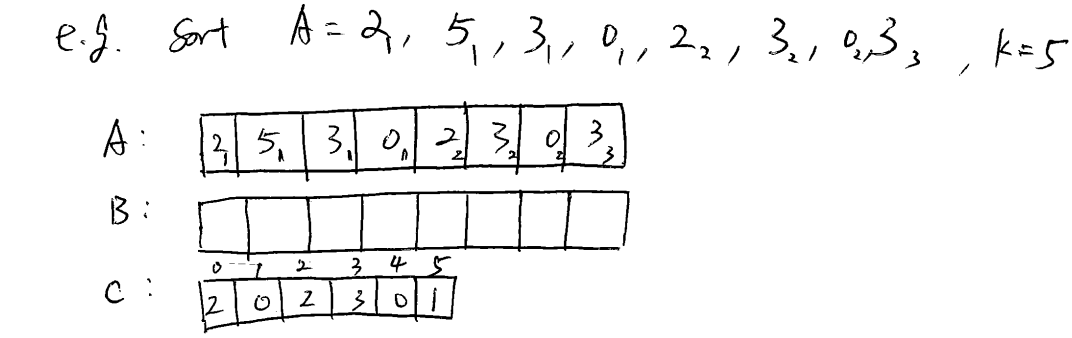
2.1.1 Algorithm

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

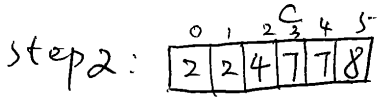
What this algorithm does is to first summarize the input array and then distribute the elements in the array into the right places in the output array.

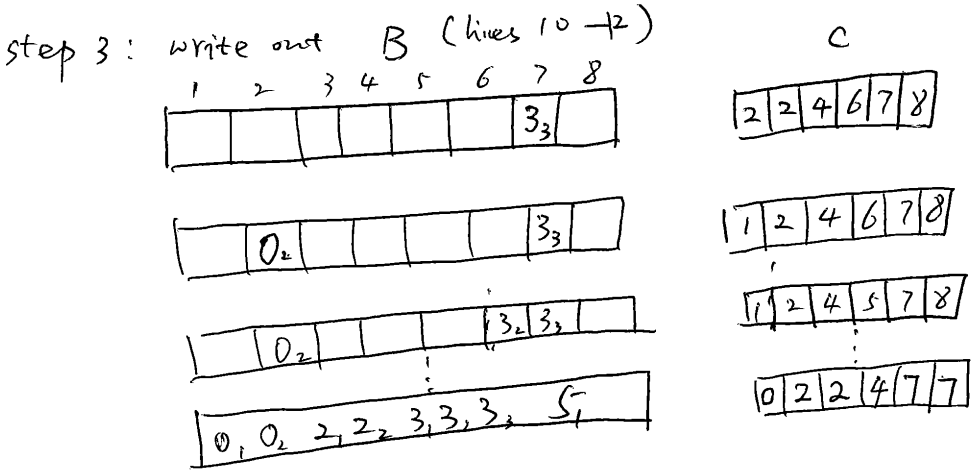
2.1.2 Example





step 1 (line 4-5) Summarise A into C (shown above)





Q:/ What's the worst case scenario?

A:/ there is really no worst case since there is no if. First and third for loops take k and the other two for loops take n. Thus the running time is $O(k+n)$.

if $k \leq n$, then it is a good algorithm. Or else, if k is like n^2 or n^3, then it isn't such a good algorithm.

e.g. if you have 8 bit integers, then $k = 256$. Thus for a regular n, it is a good algorithm. But if you have 32 bit integers, then you need $n \geq 2^{32}$ which is about 4 billion. Thus our C array is like 16G bytes (4*4.2billion) to be initialized.

One important property of counting sort is stability (Stable sort preserves the relative order of equal elements.) - some of the sorting algorithms we learned are stable, some are not. Merge, insertion, and bubble are stable sorting algorithms.

2.2 Radix sort

To handle large range of integers, radix sort uses count sort as a subroutine.

Radix sort is one of the oldest sorting algorithms (around 1900). There are punch cards with grids of 80 columns and 12 rows.

This is how IBM made its money. Punch card readers for census tabulation in early 1900's. It usually took about 10 years to analyze the data from the census. :)

Card sorters, worked on one column at a time. It's the algorithm for using the machine that extends the technique to multi-column sorting. The human operator was part of the algorithm!

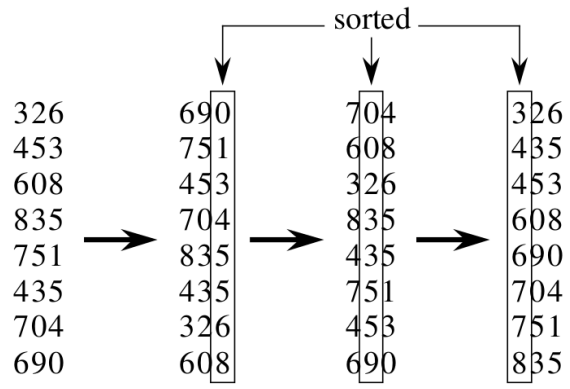
To summerize the problem, we have n numbers of d digits each.

Key idea: sort the least significant digit first.

2.2.1 Algorithm

RADIX-SORT(A, d)
for $i = 1$ **to** d
use a stable sort to sort array A on digit i

2.2.2 Example



2.2.3 Analysis

Correctness: can be proved via induction

- Assume digits $1, 2, ..., i-1$ are sorted
- Show that a stable sort on digit i are different ordering by position i is correct, and positions $1, ...i-1$ are irrelevant.
  - If 2 digits in position i are different, ordering by position i is correct, and positions $1, ..., i-1$ are irrelevant.
  - If 2 digits in position i are eual, numbers are already in the right order (by inductive hypothesis). The stable sort on digit i leaves them in the right order.

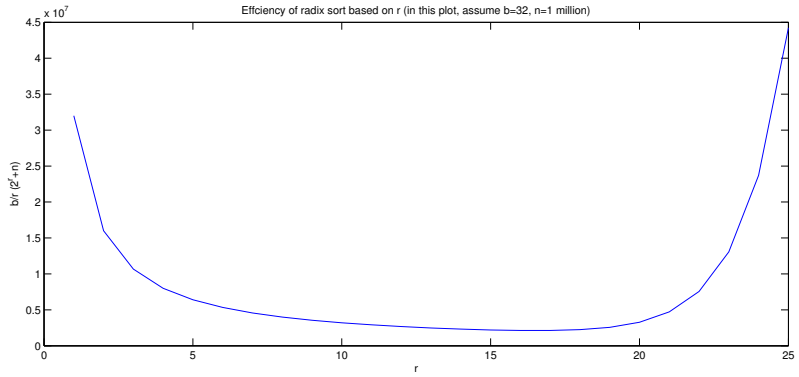
Efficiency: for count sort, it's efficiency is $O(k+n)$. Thus the running time of radix sort is $O(d(k+n))$

But we only use the range in count sort, not the digits like in radix sort

Q:/ How to handle this?
A:/ If we have n integers each is b bits long, the range is 0 to $2^b - 1$. We split each integer into b/r digits each digit is r bits long. Thus $d = b/r$ and $k = 2^r - 1$. Thus for radix sort, it's efficiency $O(b/r * (n+k)) = O(b/r(n+2^r))$. The efficiency has a min of $b/lgn \times n$ when $r = lgn$.

- If we choose $r < lgn$, then $b/r > b/lgn > b/lgn$ and $n + 2^r$ doesn't improve.
- If we choose $r > lgn$, then $n + 2^r$ term gets big.

The following figure shows the efficiency of radix sort based on $n = 1,000,000, b = 32$. As can be seen, when r is approximately $lg1000000 = 19$, the efficiency is the best.



So to sort 2^{16} 32 bit integers, use $r = lg2^{16} = 16$ bits, $\lfloor b/r \rfloor = 2$ passes.

2.2.4 Comparison between radix sort and comparison based sort

If we use 1 million 32 bit integers, radix sort need 2 passes while merge or quicksort need $lgn = 20$ passes. However, each radix sort pass is actually 2 passes, one to make cencus, and one to move data
Q:/ How can we achieve linear time sorting in radix sort? A:/

- Using count sort allows us to gain information about keys by means other than directly comparing 2 keys
- Uses keys as indexes of arrays because we only sort integers using radix sort.