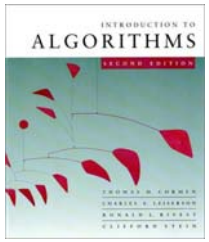
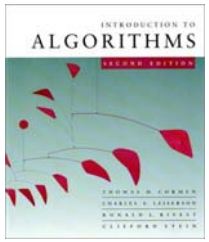
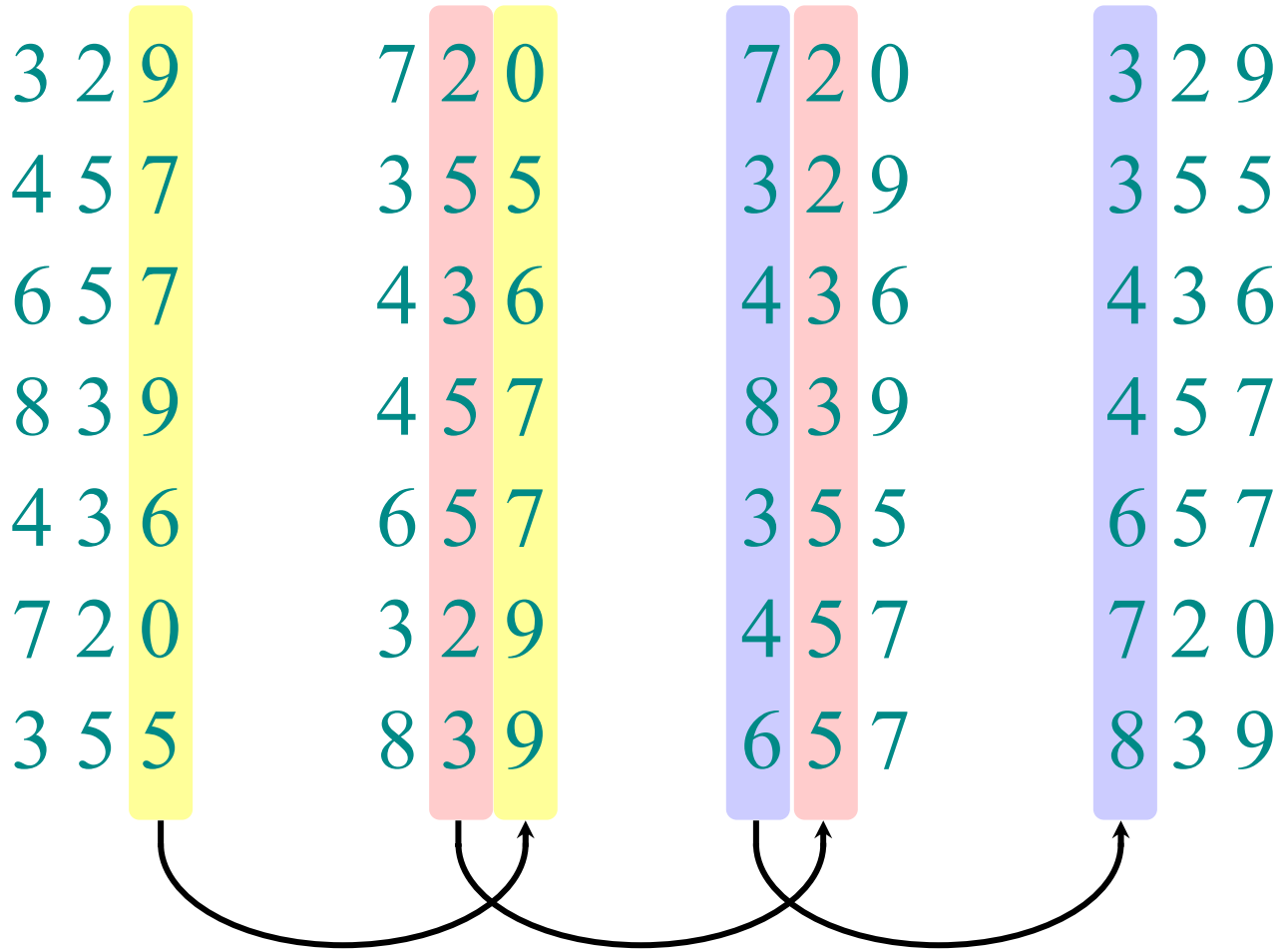


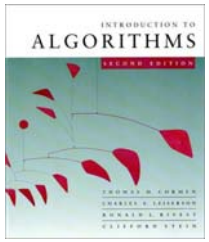
Radix sort

- ***Origin***: Herman Hollerith's card-sorting machine for the 1890 U.S. Census. (See Appendix  .)
- Digit-by-digit sort.
- Hollerith's original (bad) idea: sort on most-significant digit first.
- Good idea: Sort on ***least-significant digit first*** with auxiliary ***stable*** sort.



Operation of radix sort



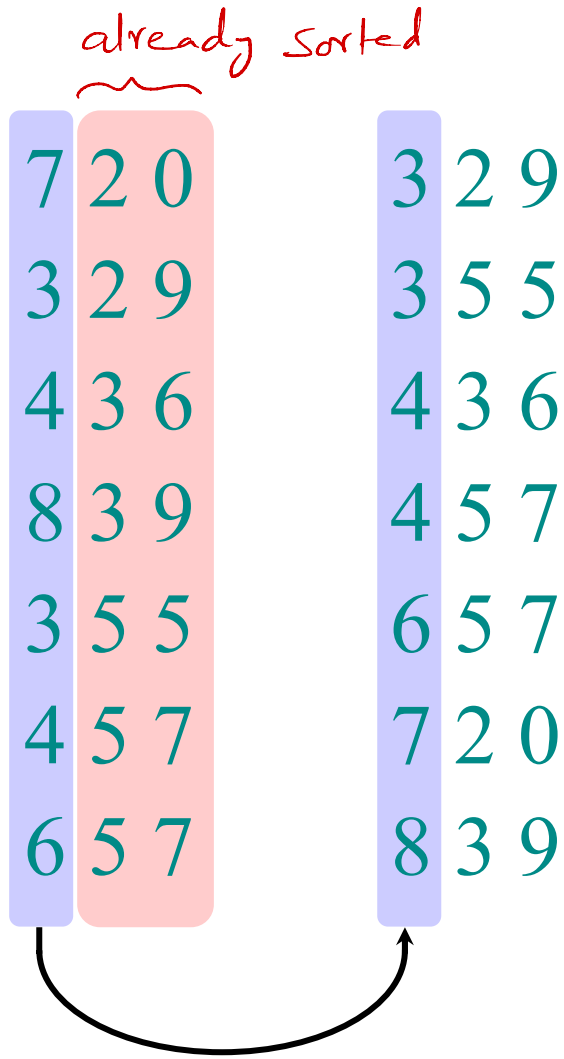


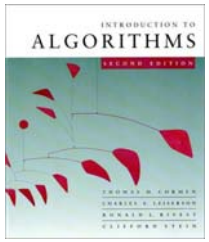
Correctness of radix sort

Induction on digit position

- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t

Given that we use a stable sort.

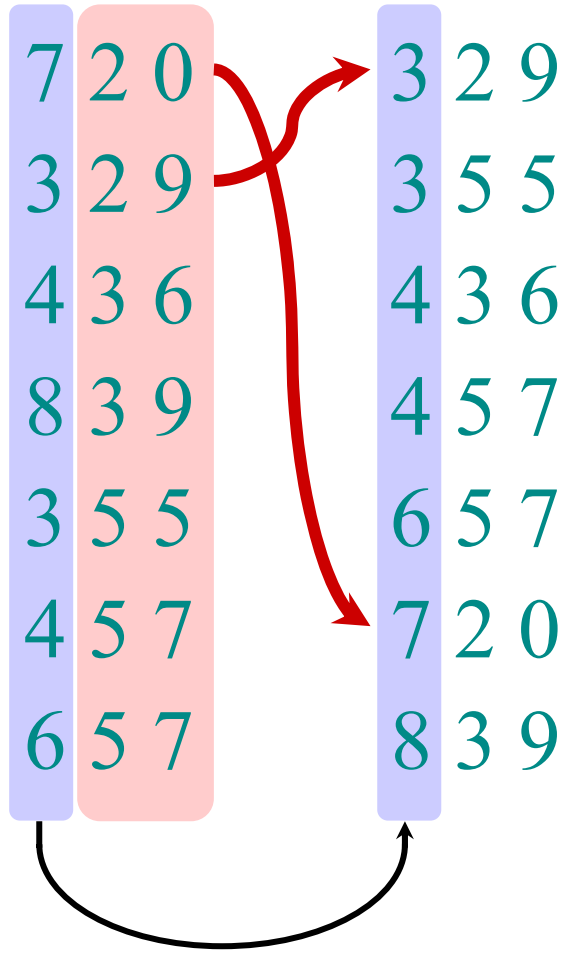


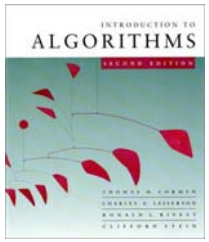


Correctness of radix sort

Induction on digit position

- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t
  - Two numbers that differ in digit t are correctly sorted.

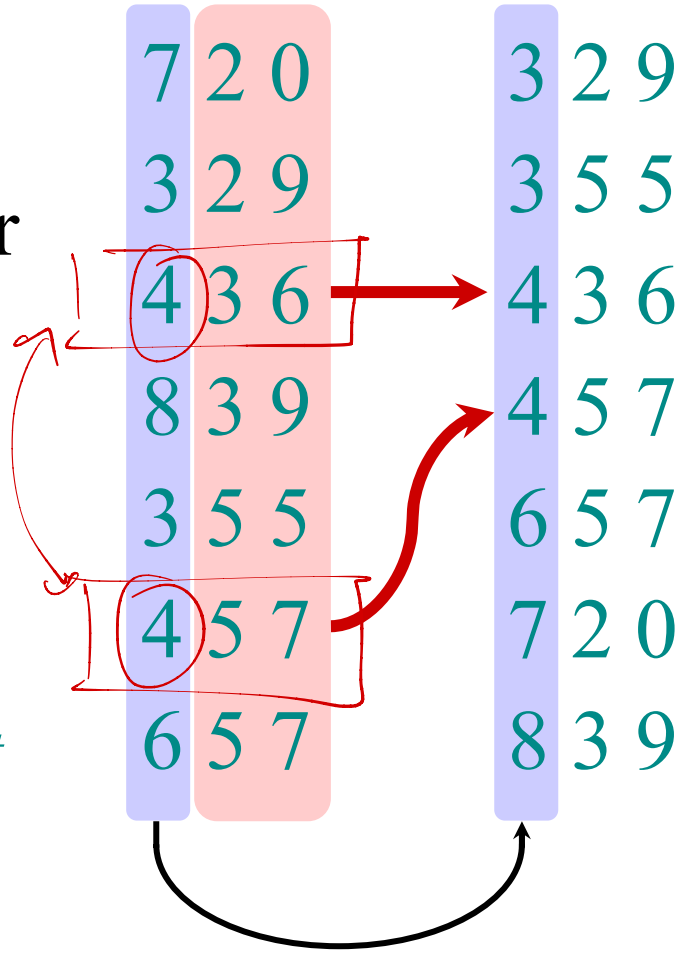




Correctness of radix sort

Induction on digit position

- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t
  - Two numbers that differ in digit t are correctly sorted.
  - Two numbers equal in digit t are put in the same order as the input ⇒ correct order.



→ Since we use stable sorting!

Complexity of radix sort:

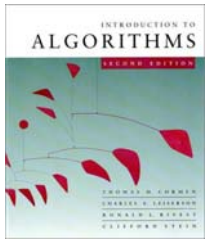
→ Each digit lies between 0 & k-1

(for decimal digits : [0, 9])

→ Each pass over n d-digit numbers

takes $\theta(n+k)$

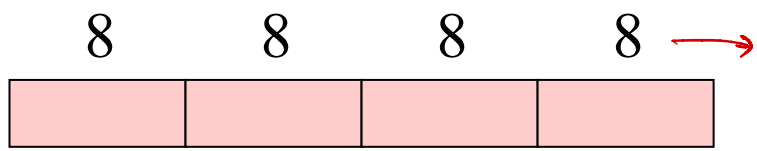
→ for d-passes, total time = $\theta(d(n+k))$



Analysis of radix sort

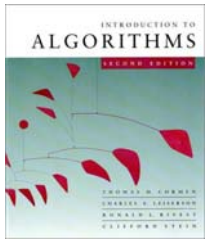
- Assume counting sort is the auxiliary stable sort.
- Sort n computer words of b bits each.
- Each word can be viewed as having b/r base-2^r digits.

Example: 32-bit word



$r = 8 \Rightarrow b/r = 4$ passes of counting sort on base-2^8 digits; or $r = 16 \Rightarrow b/r = 2$ passes of counting sort on base-2^{16} digits.

How many passes should we make?



Analysis (continued)

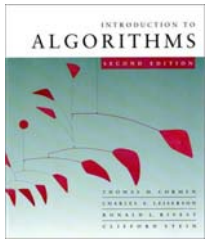
Recall: Counting sort takes $\Theta(n + k)$ time to sort n numbers in the range from 0 to $k - 1$.

If each b-bit word is broken into r-bit pieces, each pass of counting sort takes $\Theta(n + 2^r)$ time. Since there are b/r passes, we have

$$T(n,b) = \Theta\left(\frac{b}{r}\left(n + 2^r\right)\right).$$

Choose r to minimize $T(n,b)$:

- Increasing r means fewer passes, but as $r \gg \lg n$, the time grows exponentially.



Choosing r

$$T(n,b) = \Theta\left(\frac{b}{r}\left(n + 2^r\right)\right)$$

Minimize $T(n,b)$ by differentiating and setting to 0.

Or, just observe that we don't want $2^r \gg n$, and there's no harm asymptotically in choosing r as large as possible subject to this constraint.

Choosing $r = \lg n$ implies $T(n,b) = \Theta(bn/\lg n)$.

- For numbers in the range from 0 to $n^d - 1$, we have $b = d \lg n \Rightarrow$ radix sort runs in $\Theta(dn)$ time.

Given n b-bit numbers, for any $r \leq b$

Complexity of radix sort $= \theta\left(\frac{b}{r}(n + 2^r)\right)$

Case I : If $b < \lfloor \lg n \rfloor$

For $r \leq b$ we get $(n + 2^r) = \theta(n)$

(Because $2^r \leq 2^{\lfloor \lg n \rfloor} \leq n$)

Thus choosing $r = b$ yields a runn time of $\theta\left(\frac{b}{b}(n + 2^b)\right)$

$\theta(n + 2^b) = \theta(n)$

which is asymp. optimal.

Case II : If $b \geq \lfloor \lg n \rfloor$

$r = \lfloor \lg n \rfloor$ gives the best time within a const. factor

Choose, $r = \lfloor \lg n \rfloor \Rightarrow$ running time $\theta\left(\frac{b}{\lg n}(n+n)\right)$

$= \theta(bn/\lg n$

→ if we increase r above $\lfloor \lg n \rfloor$ then it gives a running time of $\Omega(bn/\lg n)$

→ if we decrease r below $\lfloor \lg n \rfloor$ then $\frac{b}{r}$ increases but $(n+2^r)$ remains $\theta(n)$

Bucket Sort

- Assumes input is drawn form a uniform distribution
- Avg running time of $O(n)$
- Input number in range $[0, 1]$
  (input sampled from uniform distribution)

Approach: (1) Divide the interval $[0, 1)$ into n equal sized buckets

(2) Distribute the n input numbers into the buckets

[Note: we do not expect many numbers falling into each bucket due to uniform and independent distribution over $[0, 1)$]

(3) To produce output, we simply sort each bucket and go through each bucket in order, listing elements in each.

→ Use any sorting algo. e.g. Insertion sort $O(n^2)$!

— Given input array $A[1:n]$

where $A[i]$ satisfies $0 \le A[i] < 1$

create an array $B[0:n-1]$ of linked lists (buckets)

BUCKET-SORT(A, n)

Let $B[0:n-1]$ be a new array
for $i = 0$ to $n-1$
    make $B[i]$ an empty linked list
for $i = 1$ to n
    insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$
for $i = 0$ to $n-1$
    sort list $B[i]$ with insertion sort
Concatenate lists $B[0], B[1], \ldots, B[n-1]$ together in order
Return concatenated list

Correctness: Consider two elements $A[i]$ and $A[j]$ such that

$$A[i] \leq A[j]$$

- Since $\lfloor nA[i] \rfloor \leq \lfloor nA[j] \rfloor$, either $A[i]$ goes into the same bucket as $A[j]$, or into a bucket with lower index.
- If both go into the same bucket then insertion sort puts them into correct order.
- If they go into different buckets then the concatenation puts them into correct order.

Running-time complexity: All executions take $O(n)$ except insertion sort

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

, when n_i is the number of element in i^{th} bucket.

To calculate average-case running-time, take expectation.

Let n_i be the random variable denoting the number of elements placed in bucket $B[i]$

$$E(T(n)) = E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \quad \text{—— linearity of expectation}$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \quad \text{—— } E[aX] = aE[X]$$

Claim: $E[n_i^2] = (2 - 1/n)$

- for each $i = 0, 1, 2, \ldots (n-1)$, each bucket i has the same value of $E[n_i^2]$ since each value from A is equally likely to fall into a bucket.

— View each random variable n_i as the number of successes in n Bernoulli trials.

Success = when element goes into bucket B[i]

$$\begin{cases} P = 1/n & \text{for success} \\ q = 1 - 1/n & \text{for failure} \end{cases}$$

$$E[n_i] = np = n\left(\frac{1}{n}\right) = 1$$

$$\text{Var}[n_i] = npq = \left(1 - \frac{1}{n}\right)$$

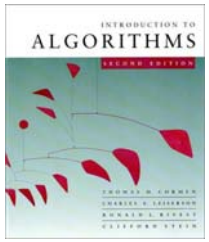
$$E[n_i^2] = \text{Var}[n_i] + E^2[n_i]$$

$$= \left(1 - \frac{1}{n}\right) + 1^2 = (2 - 1/n)$$

$\Rightarrow$ Avg. Case running time of bucket sort is:

$$T(n) = \Theta(n) + n \cdot O(2 - 1/n)$$

$$\Rightarrow \boxed{T(n) = \Theta(n)}$$



Conclusions

In practice, radix sort is fast for large inputs, as well as simple to code and maintain.

Example (32-bit numbers):

- At most 3 passes when sorting ≥ 2000 numbers.
- Merge sort and quicksort do at least $\lceil \lg 2000 \rceil = 11$ passes.

Downside: Unlike quicksort, radix sort displays little locality of reference, and thus a well-tuned quicksort fares better on modern processors, which feature steep memory hierarchies.

