---------------------------- 8 ----------------------------

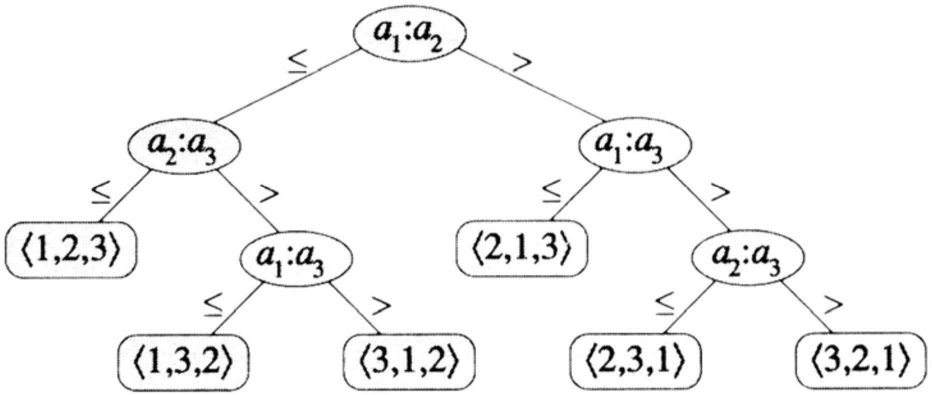
Sorting in Linear Time

8.1 Lower bounds for sorting

***Comparison sorts*:** Determine the sorted order based only on comparisons between the input elements (<, >, =, ≤, ≥). We may not inspect the values of the elements or gain order information about them in any other way.

***The decision-tree model*:** A decision tree is a full binary tree that represents the comparisons performed by a sorting algorithm that operates on an input of a given size. In a decision tree, each internal node is annotated by $a_i : a_j$, and each leaf is annotated by a permutation $\langle \pi(1), \pi(2), ..., \pi(n) \rangle$.

Example: Decision tree of insertion sort with n=3.



Lower bound for the worst case

- Each of the n! permutations on n elements must appear as a leaf.
- Worst case number of comparisons is equal to the height (the longest path from root to a leaf).
- A binary tree of height h contains at most 2^h leaves. We have $n! \le 2^h$, which implies

  $h \ge \lg(n!)$.

  Using Stirling's approximation (3.18):

  $$n! = \sqrt{2\pi n}(n/e)^n(1+\Theta(1/n)),$$

  we have

  $$h \ge \lg(n/e)^n = \Theta(n \lg n)$$

Theorem 8.1 Any decision tree that sorts n elements has height $\Omega(n \lg n)$.

Corollary 8.2 Heapsort and merge sort are asymptotically optimal comparison sorts.

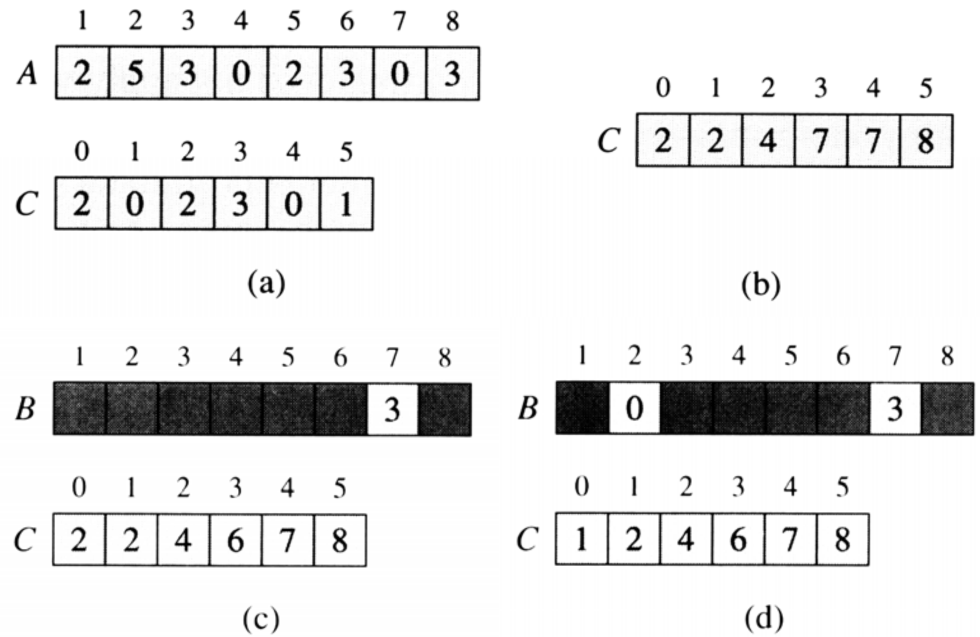
8.2 Counting sort

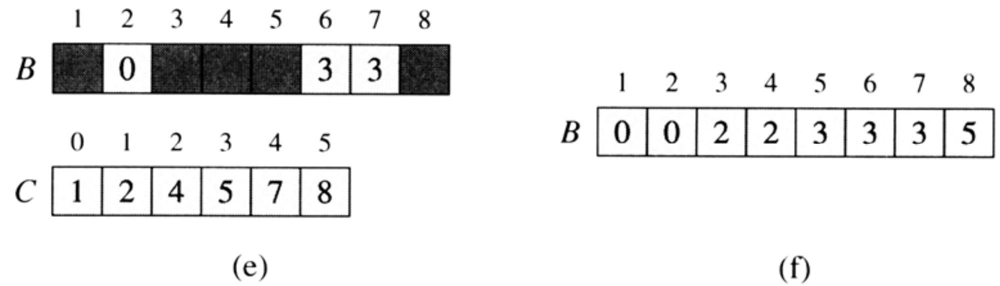
(Assume that each input is an integer in [0..*k*-1].)

A[1..*n*]: input   *C*[0..*k*-1]: counter   *B*[1..*n*]: output

Counting-Sort(*A*, *B*, *k*)
for *i* ←0 **to** *k*-1 **do** *C*[*i*] ← 0   /* Reset counters */
for *i* ←1 **to** *n* **do** *C*[*A*[*i*]] ←*C*[*A*[*i*]]+1   /* counting */
for *i* ←1 **to** *k-1* **do** *C*[*i*] ←*C*[*i*]+*C*[*i*-1] /* prefix sums */
for *i* ← *n* **downto** 1 **do**   /* output */
  B[*C*[*A*[*i*]]] ← *A*[*i*]
  C[*A*[*i*]] ← *C*[*A*[*i*]]-1

Example: *n*=8 and *k*=6.

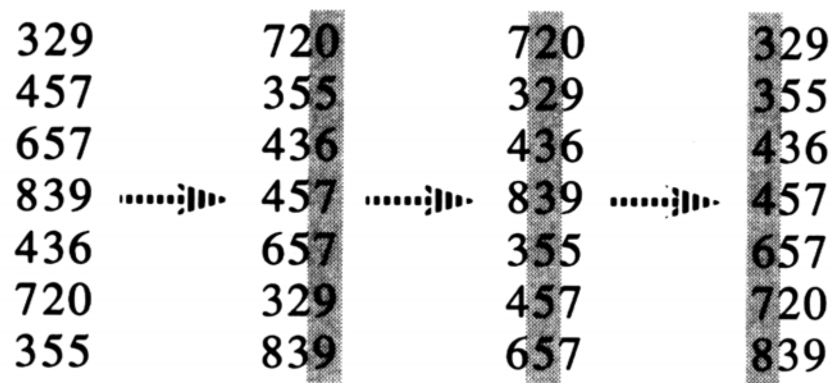




- not a comparison sort.
- **Time:** $T(n)=O(n+k)$ ($=O(n)$ if $k=O(n)$.)
- ***Stable sort*:** numbers of the same value appear in the output array in the same order as they do in the input array.
- Counting sort is stable.

8.3 Radix sort: stable sort on each digit *i* (*i*=1 to *d*)
(Every element consists of *d* digits each of which is an integer in the range [0..*k*-1].)

Example: *n*=7, *d*=3 and *k*=10



- $T(n)=O(d(n+k))$ $(=O(n)$ if $k=O(n)$ & $d=O(1)$.)
- $O(n)$ for sorting n elements in the range $[0..n^d]$, where d is a constant.

8.4 Bucket sort

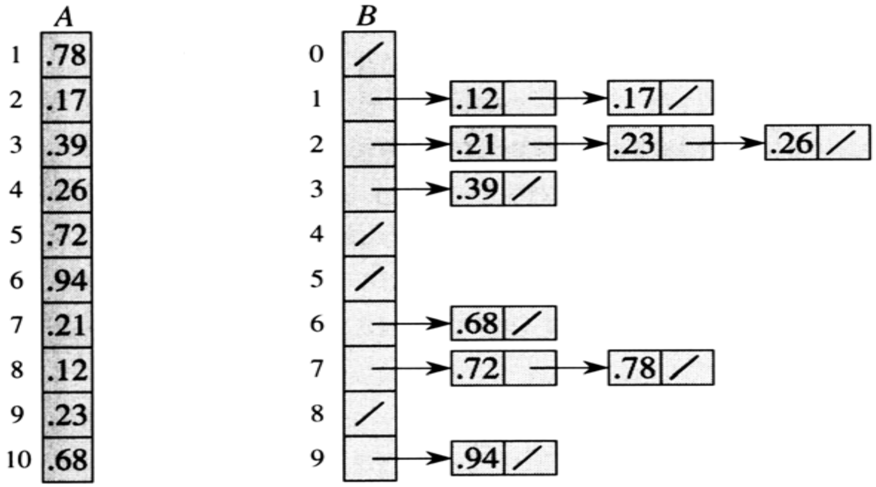
(The input distributes uniformly over the interval [0, 1).)

$A[1..n]$:input $B[0..n-1]$: buckets

Bucket-sort(A)

for $i \leftarrow 1$ **to** n **do** insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$
for $i \leftarrow 0$ **to** n-1 **do** sort list $B[i]$ by insertion sort
concatenate the lists $B[0]$, $B[1]$, ..., $B[n-1]$
convert the list into an array

Example: n=10



- Worst case: $T(n) = O(n) + \sum_{0\le i\le n-1} O(n_i^2) = O(n^2)$.
- Average case: $T(n) = O(n) + \sum_{0\le i\le n-1} O(E[n_i^2]) = O(n) + \sum_{0\le i\le n-1} O(1) = O(n)$

(See the textbook for $E[n_i^2] = \Theta(1)$.)

Homework: Ex. 8.2-4, 8.3-2, 8.4-2, Prob. 8-3, 8-6.