Lecture 10a: Linear Sorting Algorithms

Knowledge Has Organizing Power

1

1

Wholeness Statement

We proved that the lower bound on sorting by key comparisons in the best and worst cases is O(n log n). However, we can do better, i.e. linear time, but only if we have knowledge of the structure and distribution of keys. *Science of Consciousness:* Knowledge has organizing power; pure knowledge has infinite organizing power.

2

2

Outline

- Linear Time Sorting Algorithms (§4.5)
  - Bucket Sort
    - Text version of bucket sort
  - Lexicographic Sort
  - Radix Sort
  - Generic Bucket Sort

3

3

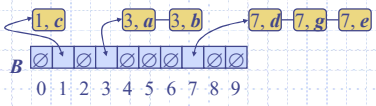
Linear Time Sorting Algorithms

Pure Knowledge Has Infinite Organizing Power

4

4

Bucket-Sort and Radix-Sort

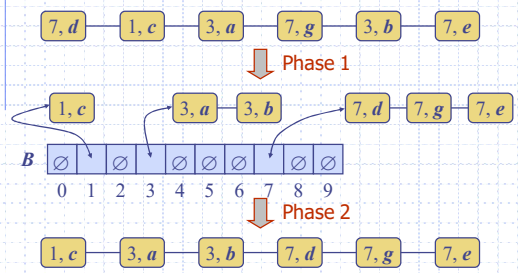


5

5

Example

- Key range [0, 9]



6

6

Bucket-Sort (§ 4.5.1)

- Let S be a list containing n (key,element) items with keys in the range $[0, N-1]$
- Bucket-sort uses the keys as indices into an auxiliary array B of lists (buckets)
  - Phase 1: Empty list L by moving each item (k, o) into its bucket $B[k]$
  - Phase 2: For $i = 0, \ldots, N-1$, move the items of bucket $B[i]$ to the end of list L
- Analysis:
  - Phase 1 takes $O(n)$ time
  - Phase 2 takes $O(n+N)$ time

  Bucket-sort takes $O(n+N)$ time

```
Algorithm bucketSort(L, N)
  Input List L of (key, element) items
    with keys in the range [0, N − 1]
  Output List L sorted by increasing
keys

  B ← array of N empty lists
  while ¬L.isEmpty() do
    (k, o) ← L.remove(L.first())
    B[k].insertLast((k, o))
  for i ← 0 to N − 1 do
    while ¬B[i].isEmpty() do
      f ← B[i].first()
      (k, o) ← B[i].remove(f)
      L.insertLast((k, o))
```


Properties and Extensions

- Key-type Property
  - The keys are used as indices into an array and cannot be arbitrary objects
  - No external comparator

Extensions

- Integer keys in the range $[a, b]$
  - Put item (k, o) into bucket $B[k-a]$
- String keys from a set D of possible strings, where D has constant size (e.g., names of the 50 U.S. states)
  - Sort D and compute the rank $r(k)$ of each string k of D in the sorted sequence
  - Put item (k, o) into bucket $B[r(k)]$

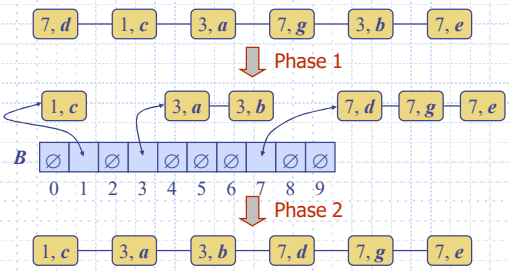
Lexicographic Sort

Stable Sorting

- **Stable** Sort Property
  - The relative order of any two items with the same key is preserved after the execution of the algorithm
- Not all sorting algorithms preserve this property

Example

- Key range $[0, 9]$



Lexicographic Order

- A d-tuple is a sequence of d keys $(k_1, k_2, \ldots, k_d)$, where key k_i is said to be the i-th dimension of the tuple
- Example:
  - The Cartesian coordinates of a point in space are a 3-tuple
- The lexicographic order of two d-tuples is recursively defined as follows

$$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d)$$
$$\Leftrightarrow$$
$$x_1 < y_1 \vee (x_1 = y_1 \wedge (x_2, \ldots, x_d) < (y_2, \ldots, y_d))$$

I.e., the tuples are compared by the first dimension, then by the second dimension, etc.

Lexicographic-Sort

- Let C_i be the comparator that compares two tuples by their i-th dimension
- Let *stableSort*(S, C) be a stable sorting algorithm that uses comparator C
- Lexicographic-sort sorts a sequence of d-tuples in lexicographic order by executing d times algorithm *stableSort*, one per dimension
- Lexicographic-sort runs in $O(dT(n))$ time, where $T(n)$ is the running time of *stableSort*

Algorithm *lexicographicSort*(S)
Input sequence S of d-tuples
Output sequence S sorted in lexicographic order

for $i \leftarrow d$ **downto** 1
  stableSort(S, C_i)

Example:

(7,4,6) (5,1,5) (2,4,6) (2,1,4) (3,2,4)

(2,1,4) (3,2,4) (5,1,5) (7,4,6) (2,4,6)

(2,1,4) (5,1,5) (3,2,4) (7,4,6) (2,4,6)

(2,1,4) (2,4,6) (3,2,4) (5,1,5) (7,4,6)

13

13

- This kind of ordering is sometimes used when records are keyed by multiple fields
- Question:
  - What is the meaning of "radix"?

14

14

Radix

- The base of a number system,
  - e.g., base 10 in decimal numbers or base 2 in binary numbers
  - aka radix 10 or radix 2

15

15

Radix Sort

16

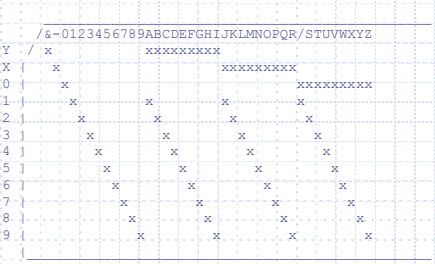
16

Radix Sort

- The algorithm used by card sorting machines (now found only in museums)
  - Cards were organized into 80 columns such that a hole could be punched in 12 possible slots per column
  - The sorter was mechanically "programmed" to examine a given column of each card and distribute the card into one of 12 bins
- What if we need to sort more than one column?
  - Radix sort solves the problem
  - Requires a stable sorter (defined below)

17

17



18

18

Punch Cards

From a Fortran program: Z(1) = Y + W(1)

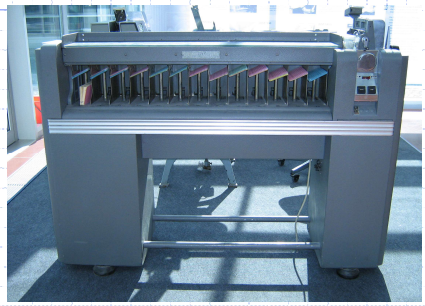
Binary punch card

19

19

20

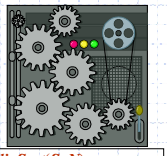
20



21

21

Radix-Sort (§ 4.5.2)



- Radix-sort is a specialization of lexicographic-sort that uses bucket-sort as the stable sorting algorithm in each dimension
- Radix-sort is applicable to tuples where the keys in each dimension i are integers in the range $[0, N-1]$
- Radix-sort runs in time $O(d(n+N))$

Algorithm *radixSort(S, N)*
Input sequence S of d-tuples such that $(0, \ldots, 0) \le (x_1, \ldots, x_d)$ and $(x_1, \ldots, x_d) \le (N-1, \ldots, N-1)$ for each tuple $(x_1, \ldots, x_d)$ in S
Output sequence S sorted in lexicographic order

for $i \leftarrow d$ **downto** 1
  replace the key k of each item (k, x) of S with dimension x_i of x
  bucketSort(S, N)

22

22

Radix-Sort for Binary Numbers

- Consider a sequence of n b-bit integers
  $x = x_{b-1} \ldots x_1 x_0$
- We represent each element as a b-tuple of integers in the range $[0, 1]$ and apply radix-sort with $N = 2$
- This application of the radix-sort algorithm runs in $O(bn)$ time
- For example, we can sort a sequence of 32-bit integers in linear time

Algorithm *binaryRadixSort(S)*
Input sequence S of b-bit integers
Output sequence S sorted
  replace each element x of S with the item $(0, x)$
for $i \leftarrow 0$ **to** $b-1$
  replace the key k of each item (k, x) of S with bit x_i of x
  bucketSort$(S, 2)$

23

23

Example

- Sorting a sequence of 4-bit integers

1001	0010	1001	1001	0001
0010	1110	1101	0001	0010
1101	1001	0001	0010	1001
0001	1101	0010	1101	1101
1110	0001	1110	1110	1110

24

24

Main Point

1. A radix-sort does successive bucket sorts, one for each "digit" in the key beginning with the least significant digit going up to the most significant; it has linear running time.
   The nature of life is to grow and progress; Natural Law unfolds in perfectly orderly sequence that gives rise to the universe, all of manifest creation.

25

25

Summary of Sorting Algorithms (§4.6)

Algorithm	Time	Notes (pros & cons)
insertion-sort		
merge-sort		
quick-sort		
heap-sort		
bucket-sort radix-sort		

26

26

Summary of Sorting Algorithms (§4.6)

Algorithm	Time	Notes (pros and cons)
insertion-sort	$O(n^2)$ or $O(n+k)$	◆ excellent for small inputs ◆ fast for 'almost' sorted inputs
merge-sort	$O(n \log n)$	◆ excels in sequential access ◆ for huge data sets
quick-sort	$O(n \log n)$ expected	◆ in-place, randomized ◆ locality of reference
heap-sort	$O(n \log n)$	◆ in-place ◆ fewest key comparisons
bucket-sort radix-sort	$O(n+N)$ $O(d(n+N))$	◆ if integer keys & keys known ◆ faster than quick-sort

27

27

Generic Bucket Sort

What was the problem with the Bucket Sort that we saw earlier?

28

28

Generic Bucket Sort

- Three phases
  1. Distribution into buckets
  2. Sorting the buckets
  3. Combining the buckets

29

29

1. Distribution

- Each key is examined once
  - a particular field of bits is examined or
  - some work is done to determine in which bucket it belongs
    - e.g., the key is compared to at most k preset values
- The item is then inserted into the proper bucket
- The work done in the distribution phase must be $\Theta(n)$

30

30

2. Sorting the Buckets

- Most of the work is done here
- O(m log m) operations are done for each bucket
  - m is the bucket size (m=n/N)

3. Combining the Buckets

- The sorted sequences are concatenated
- Takes Θ(n) time

31

31

Analysis of Generic Bucket Sort

- If the keys are evenly distributed among the buckets and there are *N* buckets
  - Then the size of the buckets m = n/N
  - Thus the work (key comparisons) done would be
    c m log m for each of the N buckets
  - That is, the total work would be
    N c (n/N) log (n/N) = c n log (n/N)
- If the number of buckets N = n/20, then the size of each bucket (n/N) is equal to 20, so the number of key comparisons would be
  c n log 20
- Thus bucket sort would be linear when the input comes from a uniform distribution
- Note also that the larger the bucket size, the larger the constant (log m)

32

32

Summary of Sorting Algorithms (§4.6)

Algorithm	Time	Notes (pros and cons)
insertion-sort		
merge-sort		
quick-sort		
heap-sort		
Generic bucket-sort		

33

33

Summary of Sorting Algorithms (§4.6)

Algorithm	Time	Notes (pros and cons)
insertion-sort	$O(n^2)$ or $O(n+k)$	excellent for small inputs fast for 'almost' sorted inputs
merge-sort	$O(n \log n)$	excels in sequential access for huge data sets
quick-sort	$O(n \log n)$ expected	in-place, randomized excellent generalized sort
heap-sort	$O(n \log n)$	in-place fastest for in-memory
Generic bucket-sort	$O(n \log(n/k))$	if keys can be distributed evenly and relatively small bucket sizes

34

34

Main Point

2. In Bucket-sort, knowledge of the structure of keys allows them to be distributed into k distinct buckets that are sorted separately and recombined. The running time is O(n log(n/k)). Knowledge has organizing power. Pure knowledge has infinite organizing power for optimum efficiency in fulfilling one's desires.

35

35

Connecting the Parts of Knowledge with the Wholeness of Knowledge

1. Using comparison of keys only, the best sorting algorithm can only achieve a running time of O(n log n) on the average.
2. Through further knowledge of the structure and distribution of keys, a bucket sort and a radix sort can achieve O(n) running time.

36

36

3. **Transcendental Consciousness**, when directly experienced, is the basis for fully understanding the unified field located by Physics.
4. **Impulses within Transcendental Consciousness**: The dynamic natural laws within this field create and maintain the order and balance in creation. We verify this through regular practice and finding the nourishing influence of the Absolute in all areas of our life.
5. **Wholeness moving within itself:** In Unity Consciousness, knowledge is on the move; the fullness of pure consciousness is flowing onto the outer fullness of relative experience. Here there is nothing but knowledge; the knowledge is self-validating.

37

37