Linear Sorting

- Topics Covered:
  - Lower Bounds for Sorting
  - Counting Sort
  - Radix Sort
  - Bucket Sort

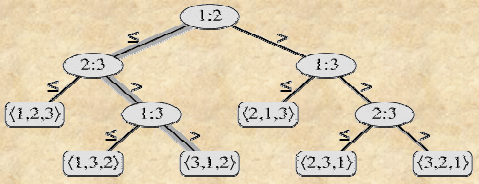
Lower Bounds for Sorting

- Comparison vs. non-Comparison sorting
- Decision tree model
- Worst case lower bound

Comparison Sorting

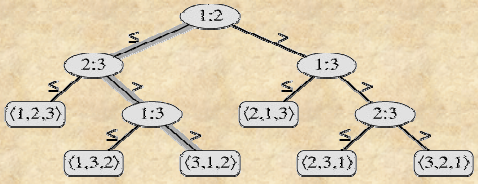
- Relies solely on relative ordering of elements
- Generalization: only ≤ operation allowed between any two elements
- Examples of comparison sorts: quicksort, heapsort, bubblesort, mergesort, selection sort, etc.

Decision Tree Model p.1



- Internal node = comparison between two elements
- Leaf node = correct ordering (sort complete)

Decision Tree Model p.2



- Path = comparisons made to arrive at an ordering
- Every possible ordering must appear as a leaf

Worst Case Lower Bound

- The length of longest path from root to leaf is number of comparisons in worst case
- Problem reduces to: what is min height of decision tree in which every possible permutation is a leaf?
- Let h = height of tree, n = size of set. Then answer = min{ h: $n! \le 2^h$ } or min { h: $\log(n!) \le h$ }, that is

$$h = \log(n!) = \Theta(\log(n^n)) = \Theta(n \log n) = \Omega(n \log n)$$

Proof of Lower Bound

- Stirling's Approximation for n!

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+O\left(\frac{1}{n}\right)\right)$$

$$\log(n!) = \frac{1}{2}\log(2\pi n)+\log\left(n^n\right)-n\log(e)+\log\left(1+O\left(1/n\right)\right)$$

$$= \log\left(n^n\right)+O(n) = \Theta(n\log n)$$

Counting Sort

- Tallies each number's occurrence within the array
- The algorithm assumes each element in the array is an integer, in the range 0 to k
- Needs a temporary array for working space with length k
- Also needs another array to hold the output

Counting Sort Algorithm

orig ← Original Array
out ← Output Array
temp ← Temporary array
for *i* ← 0 **to** *k*
  do temp[*i*] ← 0
for *j* ← 1 **to** *length*[*orig*]
  do temp[orig[*j*]] ← *temp*[orig [j]] + 1

Counting Sort

orig

5	3	2	4	5	4	1	4	1	2

– Made a second array with length equal to the original array's length

temp

- Two 1s
- Two 2
- One 3
- Three 4s
- Two 5s

0	2	2	1	3	2
0	1	2	3	4	5

Counting Sort

Calculating the number of elements less than or equal to each slot number (Stable)

for *i* ← 1 **to** *k*
  do *temp*[*i*] ← temp[*i*] + temp[*i* - 1]

temp

before

0	2	4	5	8	10
0	1	2	3	4	5

for *j* ← *length*[orig] - 1 to 0
  do
    temp[orig[*j*]] ← temp[orig[*j*]] - 1
    out[temp[orig[*j*]]] ← orig[*j*]

orig

5	3	2	4	5	4	1	4	1	2
0	1	2	3	4	5	6	7	8	9

temp

0	0	3	5	7	10
0	1	2	3	4	5

out

1	1		2				4		
0	1	2	3	4	5	6	7	8	9

Counting Sort

Final Result

- Two 1s
- Two 2
- One 3
- Three 4s
- Two 5s

1	1	2	2	3	4	4	4	5	5

Counting Sort

Alternative version:

Going through the temp array, use the information to fill the original array with sorted numbers (Unstable):

Advantage: Don't need output array

orig

1	1	2	2	3	4	4	4	5	5
0	1	2	3	4	5	6	7	8	9

temp

0	2	2	1	3	2
0	1	2	3	4	5

Counting Sort

- Advantages
  - Fast
  - Stable
    - Duplicates maintain order
- Disadvantages
  - Requires additional memory

Radix sort

or

Old School

Definition

- is the algorithm used by the card-sorting machines
- A multiple pass sort algorithm that distributes each item to a bucket according to part of the item's key beginning with the least significant part of the key. After each pass, items are collected from the buckets, keeping the items in order, then redistributed according to the next most significant part of the key

How it works…

- The cards are organized into 80 columns
- in each column a hole can be punched in one of 12 places
- The sorter mechanically examines a given column of each card in a deck and distributes the card into one of 12 bins depending on which place has been punched

- An operator gathers the cards bin by bin, so that cards with the first place punched are on top of cards with the second place punched, and so on.
- For decimal digits, only 10 places are used in each column.
- The other two places are used for encoding nonnumeric characters
- A *d*-digit number would then occupy a field of *d* columns

- Unfortunately, since the cards in 9 of the 10 bins must be put aside to sort each of the bins, this procedure generates many intermediate piles of cards that must be kept track of.
- It sorts on the *least significant* digit first.
- http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/radixsort.html

Bucket Sort

- ***Bucket sort*** runs in linear time when the input is drawn from a uniform distribution.
- Like counting sort, bucket sort is fast because it assumes something about the input.
- Whereas counting sort assumes that the input consists of integers in a small range, bucket sort assumes that the input is generated by a random process that distributes elements uniformly over the interval [0, 1).

Idea

- divide the interval [0, 1) into *n* equal-sized subintervals, or ***buckets***
- distribute the *n* input numbers into the buckets
- Assumption: Since the inputs are uniformly distributed over [0, 1), we don't expect many numbers to fall into each bucket

- sort the numbers in each bucket
- go through the buckets in order, listing the elements in each.

Algorithm

- Let be ***S*** be a sequence of ***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 sequences (buckets)
- Phase 1: Empty sequence ***S*** by moving each item (***k***, ***o***) into its bucket ***B***[***k***]
- Phase 2: For ***i*** = 0, **…**, ***N*** - 1, move the items of bucket ***B***[***i***] to the end of sequence ***S***

Analysis

- Phase 1 takes ***O***(***n***) time
- Phase 2 takes ***O***(***n*** + ***N***) time
- Bucket-sort takes ***O***(***n*** + ***N***) time

Example

- http://www.cs.iitm.ernet.in/tell/foc_selfstudy/code/bucket.html

References

- Thomas H. Cormen et al. Introduction to Algorithms, 2nd Edition. MIT Press © 2001
- Counting Sort Algorithm. Worcester Polytechnical Institute <http://www.cs.wpi.edu/~dobrush/cs504/s02/projects/counting_sort.htm> 3/18/2003
- CS2 Lab 7. Rochester Institute of Technology <http://www.cs.rit.edu/~cs2/NewLabs/07/act1.html> 3/18/2003

References

part 2

- http://www.nist.gov/dads/HTML/radixsort.html
- http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/radixsort.html
- http://www.datastructures.net/presentations
- http://www.cs.iitm.ernet.in/tell/foc_selfstudy/code/bucket.html