Sorting Lower Bound
Radix Sort

CSE 373
Data Structures
Lecture 15

Reading

- Reading
  - Sections 7.8-7.11

How fast can we sort?

- Heapsort, Mergesort, and Quicksort all run in O(N log N) best case running time
- Can we do any better?
- No, if the basic action is a comparison.

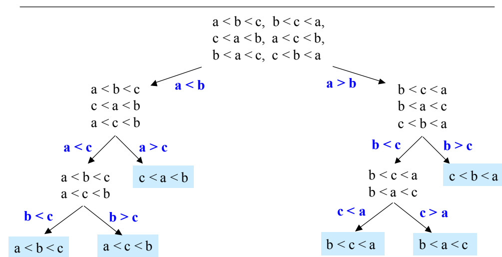
Sorting Model

- Recall our basic assumption: we can <u>only compare two elements at a time</u>
  - we can only reduce the possible solution space by half each time we make a comparison
- Suppose you are given N elements
  - Assume no duplicates
- How many possible orderings can you get?
  - Example: a, b, c (N = 3)

Permutations

- How many possible orderings can you get?
  - Example: a, b, c (N = 3)
  - (a b c), (a c b), (b a c), (b c a), (c a b), (c b a)
  - 6 orderings = 3·2·1 = 3! (ie, “3 factorial”)
  - All the possible permutations of a set of 3 elements
- For N elements
  - N choices for the first position, (N-1) choices for the second position, …, (2) choices, 1 choice
  - N(N-1)(N-2)⋯(2)(1)= <u>N! possible orderings</u>

Decision Tree



The leaves contain all the possible orderings of a, b, c

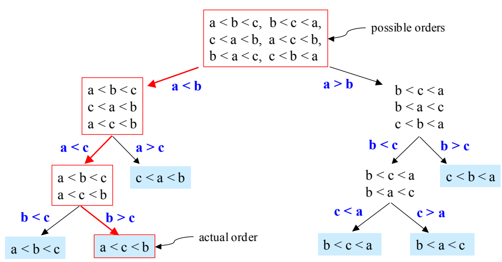
Decision Trees

- A Decision Tree is a Binary Tree such that:
  - › Each node = a set of orderings
    - ie, the remaining solution space
  - › Each edge = 1 comparison
  - › Each leaf = 1 unique ordering
  - › How many leaves for N distinct elements?
    - N!, ie, a leaf for each possible ordering
- Only 1 leaf has the ordering that is the desired correctly sorted arrangement

Decision Trees and Sorting

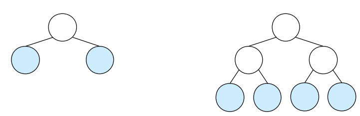
- Every sorting algorithm corresponds to a decision tree
  - › Finds correct leaf by choosing edges to follow
    - ie, by making comparisons
  - › Each decision reduces the possible solution space by one half
- Run time is ≥ maximum no. of comparisons
  - › maximum number of comparisons is the length of the longest path in the decision tree, i.e. the height of the tree

Decision Tree Example



How many leaves on a tree?

- Suppose you have a binary tree of height d . How many leaves can the tree have?
  - › d = 1 → at most 2 leaves,
  - › d = 2 → at most 4 leaves, etc.



Lower bound on Height

- A binary tree of height d has at most 2^d leaves
  - › depth d = 1 → 2 leaves, d = 2 → 4 leaves, etc.
  - › Can prove by induction
- Number of leaves, $L \leq 2^d$
- Height $d \geq \log_2 L$
- The decision tree has N! leaves
- So the decision tree has height $d \geq \log_2(N!)$

log(*N*!) is Ω(*N*log*N*)

$$\begin{aligned}\log(N!) &= \log\big(N\cdot(N-1)\cdot(N-2)\cdots(2)\cdot(1)\big)\\ &= \log N + \log(N-1) + \log(N-2) + \cdots + \log 2 + \log 1\\ &\geq \log N + \log(N-1) + \log(N-2) + \cdots + \log\frac{N}{2}\\ &\geq \frac{N}{2}\log\frac{N}{2}\\ &\geq \frac{N}{2}(\log N - \log 2) = \frac{N}{2}\log N - \frac{N}{2}\\ &= \Omega(N\log N)\end{aligned}$$

select just the first N/2 terms

each of the selected terms is ≥ logN/2

Ω(N log N)

- Run time of any comparison-based sorting algorithm is **Ω(N log N)**
- Can we do better if we don't use comparisons?

Radix Sort: Sorting integers

- Historically goes back to the 1890 census.
- Radix sort = multi-pass bucket sort of integers in the range 0 to B^P-1
- Bucket-sort from least significant to most significant "digit" (base B)
- Requires P(B+N) operations where P is the number of passes (the number of base B digits in the largest possible input number).
- If P and B are constants then O(N) time to sort!

Radix Sort Example

Input data: 478, 537, 9, 721, 3, 38, 123, 67

Bucket sort by 1's digit

0	1	2	3	4	5	6	7	8	9
	721		3 123				537 67	478 38	9

After 1st pass: 721, 3, 123, 537, 67, 478, 38, 9

This example uses B=10 and base 10 digits for simplicity of demonstration. Larger bucket counts should be used in an actual implementation.

Radix Sort Example

After 1st pass: 721, 3, 123, 537, 67, 478, 38, 9

Bucket sort by 10's digit

0	1	2	3	4	5	6	7	8	9
03 09		721 123	537 38			67	478		

After 2nd pass: 3, 9, 721, 123, 537, 38, 67, 478

Radix Sort Example

After 2nd pass: 3, 9, 721, 123, 537, 38, 67, 478

Bucket sort by 100's digit

0	1	2	3	4	5	6	7	8	9
003 009 038 067	123			478	537		721		

After 3rd pass: 3, 9, 38, 67, 123, 478, 537, 721

Invariant: after k passes the low order k digits are sorted.

Implementation Options

- List
  - › List of data, bucket array of lists.
  - › Concatenate lists for each pass.
- Array / List
  - › Array of data, bucket array of lists.
- Array / Array
  - › Array of data, array for all buckets.
  - › Requires counting.

Array / Array

i	Data Array	Count Array	Address Array	Target Array	Bucket
0	478	0	0	721	1
1	537	1	0	3	3
2	9	0	1	123	
3	721	2	1	537	7
4	3	0	3	67	
5	38	0	3	478	8
6	123	0	3	38	
7	67	2	3	9	9
8		2	5		
9		1	7		

Bucket i ranges from add[i] to add[i+1]-1

add[0] := 0
add[i] := add[i-1] + count[i-1], i > 0

Array / Array

- Pass 1 (over A)
  - Calculate counts and addresses for 1st "digit"
- Pass 2 (over T)
  - Move data from A to T
  - Calculate counts and addresses for 2nd "digit"
- Pass 3 (over A)
  - Move data from T to A
  - Calculate counts and addresses for 3rd "digit"
- …
- In the end an additional copy may be needed.

Choosing Parameters for Radix Sort

- N number of integers – given
- m bit numbers - given
- B number of buckets
  - $B = 2^r$ – calculations can be done by shifting.
  - N/B not too small, otherwise too many empty buckets.
  - P = m/r should be small.
- Example – 1 million 64 bit numbers. Choose $B = 2^{16}$ =65,536. 1 Million / B ≈ 15 numbers per bucket. P = 64/16 = 4 passes.

Properties of Radix Sort

- Not in-place
  - needs lots of auxiliary storage.
- Stable
  - equal keys always end up in same bucket in the same order.
- Fast
  - $B = 2^r$ buckets on m bit numbers

$$O\left(\frac{m}{r}\left(n+2^r\right)\right) \text{ time}$$

Internal versus External Sorting

- So far assumed that accessing A[i] is fast – Array A is stored in internal memory (RAM)
  - Algorithms so far are good for internal sorting
- What if A is so large that it doesn't fit in internal memory?
  - Data on disk or tape
  - Delay in accessing A[i] – e.g. need to spin disk and move head

Internal versus External Sorting

- Need sorting algorithms that minimize disk/tape access time
  - External sorting – Basic Idea:
    - Load chunk of data into RAM, sort, store this "run" on disk/tape
    - Use the Merge routine from Mergesort to merge runs
    - Repeat until you have only one run (one sorted chunk)
    - Text gives some examples

Summary of Sorting

- Sorting choices:
  - $O(N^2)$ – Bubblesort, Insertion Sort
  - O(N log N) average case running time:
    - Heapsort: In-place, not stable.
    - Mergesort: O(N) extra space, stable.
    - Quicksort: claimed fastest in practice but, $O(N^2)$ worst case. Needs extra storage for recursion. Not stable.
  - O(N) – Radix Sort: fast and stable. Not comparison based. Not in-place.