Linear Sorting

• How fast can we sort?
  – we’ll see a lower bound of $\Omega(n \log n)$
  – then we’ll beat it by changing the rules!
• Comparison sorting
  – only operation is comparing pairs of elements
  – all sorts so far included
    • insertion sort, selection sort, merge sort, heapsort, treesort, quicksort

Lower bounds
  – need $\Omega(n)$ just to examine all input
  – show $\Omega(n \log n)$ lower bound seen in other sorts
• Decision tree is the key
  – abstraction of comparison sort
  – represents specific algorithm on inputs of given size
  – gets rid of details
  – counting only comparisons

Decision Trees

• What’s the longest path?
• Depends on algorithm
  – Insertion sort $\Theta(n^2)$
  – Merge sort $\Theta(n \log n)$
  – Heapsort $\Theta(n \log n)$
  – quicksort $\Theta(n^2)$
• Can average length of paths
  – average running time
Lemma:
Any binary tree of height $h$ has $\leq 2^h$ leaves
$l = \#$ of leaves and $h =$ height, then
\[ l \leq h \]

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

Proof of lemma:
Basis: $h = 0$. Tree is one node: a leaf. $2^0 = 1$
Inductive step: Assume it’s true for any $h-1$. Show for $h$. Extend tree of height $h-1$ by making as many new leaves as possible. That means each current leaf becomes the parent of two new leaves.
\[
\text{# of leaves for height } h = 2 \times (\text{# of leaves for height } h-1)
\]
\[= 2 \times 2^{h-1}
\]
\[= 2^h
\]
Corollary: Heapsort and merge sort are asymptotically optimal comparison sorts

Proof of theorem:
\[ l \geq n! \]
By lemma, $n! \leq l \leq 2^h$ or $2^h \geq n!$
Take logs: $h \geq \log(n!)$
\[n! > (n/e)^n \text{ by Stirling’s approximation}
\[h \geq \log(n/e)^n = n\log(n/e) = n\log n - n\log e
\]
\[= \Omega(n\log n)
\]

Counting Sort

• Depends on numbers to be sorted are integers in $\{0, 1, 2, \ldots, k\}$
• Input: $A[1..n]$ where $A[j] \in \{0,1,..,k\}$ for $j=1,2,\ldots,n$. $A$, $n$, and $k$ are parameters
• Output: $B[1..n]$ sorted. $B$ is a parameter
• Auxiliary storage: $C[0..k]$
COUNTING-SORT($A, B, n, k$)

for $i \leftarrow 0$ to $k$
  do $C[i] \leftarrow 0$

for $j \leftarrow 1$ to $n$
  do $C[A[j]] \leftarrow C[A[j]] + 1$

for $i \leftarrow 1$ to $k$
  do $C[i] \leftarrow C[i] + C[i - 1]$

for $j \leftarrow n$ downto 1
  do $B[C[A[j]]] \leftarrow A[j]$
      $C[A[j]] \leftarrow C[A[j]] - 1$

---

**Counting Sort**

- Counting sort is *stable*
  - preserves order of items
  - items with same value appear in same order in sorted array as in unsorted array
  - stable because last loop starts from the end!

**Analysis**

- $\Theta(n+k)$, if $k=n$, $\Theta(n)$
- How big can $k$ be?
  - 32 bit values would require a 4 billion entries in the $C$ array!
  - 16 bit values would require 65000 entries in the $C$ array
  - 8 bit values would require 512 entries
  - 4 bit values would require 16 entries
- Counting sort used in radix sort
  - because of stable property
Radix Sort

**RADIX-SORT**(A, d)

for i ← 1 to d
    do use a stable sort to sort array A on digit i
        • Sort numbers digit by digit
        • Start with least significant digit
        • Digit sort must be stable!
            – otherwise, the early sorting won’t be preserved!

Correctness

Induction on number of digits
For one digit trivial since counting sort works
Assume first i-1 digits sorted. Show sorting
digit i leaves digits 1,2, ..., i sorted.
– if two digits in position i are different no problem
– if two digits are the same, counting sort preserves order of lower digits, which are correct!

Radix Sort Analysis

\(\Theta(n+k)\) per pass (digits in \(\{0,1,2,\ldots,k\}\)
d passes
\(\Theta(d(n+k))\) total
If \(k = O(n)\), time = \(\Theta(dn)\)
Let there be \(n\) words with \(b\) bits/word which have \(r\)-bit digits. So, \(d = \lceil b/r \rceil\)
Use counting sort with \(k = 2^{r-1}\)
Example: 32-bit words, 8-bit digits, \(b = 32, r=8\),
\(d = \lceil 32/8 \rceil = 4\), \(k = 2^4-1 = 255\)
Choosing r

- Balance b/r and n+2^r
- r ≈ lg n gives \( \Theta(b/(lg n)(n+n)) = \Theta(bn/(lgn)) \)
- If r < lg n, then b/r > b/(lg n) and n+2^r doesn’t improve
- If r > lg n, then n+2^r gets big
  - r=2(lg n) \( \Rightarrow 2^r=(2^{lg n})^2=n^2 \)
- To sort 2^{16} 32-bit numbers use r = 16.
  - b/r = 2 passes

Comparison to other sorts

- one million (2^{20}) 32-bit numbers
- Radix-sort: ceiling(32/20) = 2 passes
- merge sort/quick sort= lg n= 20 passes
- Each radix sort is two passes.
  - take the census
  - move data
- Uses keys as array indexes via counting sort

Bucket Sort

- Divide [0,1) into n equal sized buckets
- distribute the n values into the buckets
- sort each bucket
- go through bucket in order
- Input: A[1..n], where 0 \leq A[i] < 1 for all i
- Auxilliary array: B[0..n-1] of linked lists, empty

\[
\text{BUCKET-SORT}(A, n) \\
\text{for } i \text{ from 1 to } n \\
\text{do insert } A[i] \text{ into list } B[n \cdot A[i]] \\
\text{for } i \text{ from 0 to } n - 1 \\
\text{do sort list } B[i] \text{ with insertion sort} \\
\text{concatenate lists } B[0], B[1], \ldots, B[n - 1] \text{ together in order} \\
\text{return the concatenated lists}
\]
Correctness

- if same bucket, insertion sort fixes it
- if lower bucket, concatenation step fixes it

Analysis
- Performance relies on no bucket getting too many values
- All steps except insertion sort require $$\Theta(n)$$ total
- If each bucket gets constant number of elements, $$O(1)$$ sorting $$\Rightarrow O(n)$$ sorting for all buckets
- Expect each bucket to have few elements since average should be 1 element per bucket.
- Let’s check this

Insertion sort is $$O(n^2)$$, so

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

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

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

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

Let $$X_{ij} = I \{ A[j] \text{ falls in bucket } i \}$$

$$\Pr \{ A[j] \text{ falls in bucket } i \} = 1/n$$

$$n_i = \sum_{j=1}^{n} X_{ij}$$

Then

$$E[X_i^2] = E \left[ \left( \sum_{j=1}^{n} X_{ij} \right)^2 \right]$$

$$= E \left[ \sum_{j=1}^{n} X_{ij} + 2 \sum_{j<k}^{n} X_{ij} X_{ik} \right]$$

$$= \sum_{j=1}^{n} E[X_{ij}^2] + 2 \sum_{j<k}^{n} E[X_{ij} X_{ik}]$$

$$E[X_i^2] = 0^2 \cdot \Pr \{ A[j] \text{ doesn’t fall in bucket } i \} +$$

$$1^2 \cdot \Pr \{ A[j] \text{ falls in bucket } i \}$$

$$= 0 \left( 1 - \frac{1}{n} \right) + 1 \cdot \frac{1}{n} = \frac{1}{n}$$

$$E[X_i X_{k}] = E[X_i] E[X_k] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$$
So, $E\left[n^2\right] = \sum_{j=1}^{n} \frac{1}{n} + 2 \sum_{j=1}^{n} \sum_{k=j+1}^{n} \frac{1}{n^2}$

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

$= 1 + \frac{n-1}{n} = 2 - \frac{1}{n}$

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

$= \Theta(n) + O(n) = \Theta(n)$

### Bucket sort

- Not comparison sort
- Probablistic analysis
- Not a randomize algorithm!
  - if distribution of the input isn’t uniform, then some buckets can get full!
- Performance is off, but the algorithm works correctly!