Sorting

Ming-Hwa Wang, Ph.D.
COEN 279/AMTH 377 Design and Analysis of Algorithms
Department of Computer Engineering
Santa Clara University

Sorting

- ¼ of all mainframe cycles are spent sorting data
- internal sorting - in main memory
- external sorting - on disk or tape
- comparison-based sorting - use only >, <, and assign
- an inversion in an array of numbers is any ordered pair (i, j) having the property that $i < j$ but $A[i] > A[j]$
- the average number of inversions in an array of N distinct numbers is $N(N-1)/4$
- any algorithm that sorts by exchanging adjacent elements requires $\Omega(N^2)$ time on average
- to get sub-quadratic running time, exchanges between elements that are far apart and must eliminate more than one inversion per exchange
- use sentinel $A[0]$, avoid explicit swap
- stable/unstable sorting
- memory issue: sort in place
- indirect sorting: for sorting large record, swapping pointer to the record instead of swapping record
- A general-purpose sorting algorithm cannot make assumptions about the type of input it can expect to see, but must make decisions based on ordering information only. Any general-purpose sorting algorithm requires $\Theta(N \lg N)$ comparisons.

Applications of Sorting

searching, closest pair, element uniqueness, frequency distribution, selection, convex hulls (sorted by x-coordinate, inset points from left to right, the rightmost points is always on the boundary and adding this new point might cause others to be deleted)

Decision Tree

- the model for sorting
- each node is annotated by a_i:a_j for some $1 \le i, j \le n$
- each leaf is annotated by a permutation of input data
- the execution of sorting corresponds to tracing a path from the root to a leaf
- general lower bound for sorting: use decision trees

number of comparisons = height of decision tree

Bubble Sort

- $O(N^2)$

Selection Sort

- $O(N^2)$

Incremental Insertion Sort

- for pass $P = 2$ through N, move the P^{th} element left until it correct place is found among the first P elements
- $O(N^2)$, but $O(N)$ for presorted input. average $\Theta(N^2)$
- good for small size ($N \le 20$) and near sorted input

Shellsort or Diminishing Increment Sort

- comparing elements that are distant, the distance between comparisons decreases as the algorithm runs until the last phase, in which adjacent elements are compared
- increment sequence, $h_1, h_2, .., h_t$ with $h_1 = 1$
  - Shell: $h_t = \lfloor N/2 \rfloor$ and $h_k = \lfloor h_{k+1}/2 \rfloor$, $\Theta(N^2)$
    - pairs of increments are not relatively prime, and thus the small increment can have little effect
  - Hibbard: 1, 3, 7, .., $2^k - 1$, $\Theta(N^{3/2})$
  - best: {1, 5, 19, 41, 109, ..}, $9*4^i-9*2^i+1$ or $4^i-3.2^i+1$
- h_k-sorted: using h_k, for every i, $A[i] \le A[i + h_k]$
  - an h_k-sorted file is then h_{k-1}-sorted remains h_k-sorted

Heapsort

- building a binary heap of N elements takes $O(N)$, and then perform N DeleteMin operations takes $O(N \lg N)$, need an extra array to copy back the result
- to avoid the extra copy, use a maxheap

Mergesort

- merging two sorted lists and put the output in a third list
- divide-and-conquer, using recursion
- to avoid temporary arrays declared locally for each recursion, use only one temporary array
- cornerstone of most external sorting, $O(N \lg N)$

Quicksort

- fastest in practice, worst-case $O(N^2)$, average $O(N \lg N)$
- algorithm to sort an array S consists of 4 steps:

- If the number of elements in S is 0 or 1, then return.
- Pick any element v in S. This is called the pivot.
- Partition S-{v} (the remaining elements in S) into two disjoint groups:
  $S_1 = \{x \in S\text{-}\{v\} \mid x \le v\}$ and $S_2 = \{x \in S\text{-}\{v\} \mid x \ge v\}$
- Return {quicksort(S_1) followed by v followed by quicksort(S_2)}.

- picking the pivot:
  - use the first element, $O(N^2)$ if presorted
  - choose the pivot randomly, random number generation is expensive but not reduce the average running time
  - median, too expensive to compute
  - median-of-three partitioning, even good for presorted input
- partitioning strategy:
  - To handle keys that are equal to the pivot, i and j ought to do the same thing.
  - for small file, use insertion sort instead
  - Quickselect makes only 1 recursive call instead of 2, worst-case $O(N^2)$, average $O(N)$
  - if use median-of-median-of-five as the pivot, $O(N)$

Counting/Bucket Sort

- not a general-purpose sorting algorithm
- The input A_1, A_2, .., A_N must consist of only positive numbers smaller than M. Keep an array Count of size M, which is initialized to all 0's. When A_i is read, increment Count[i]. After all input is read, scan the Count array, printing out the sorted list. $O(N+M)$.
- the input must in [0, 1). Put $A[i]$ in $B[\lfloor NA[i] \rfloor]$, insert sort $B[i]$ and then concatenate them. $O(N)$

Radix/Distribution Sort

- the least significant digit sort first
- performance depends on the distribution

External Sorting

- disk has access delay to spin the disk and move the disk head, and tape can only be accessed sequentially
- assume the internal memory can hold and sort M records at a time, and each set of sorted record is called a run
- multiway Merge: k-way merge using heap requires $\lceil \log_k(N/M) \rceil$ passes and $2k$ tapes, e.g., in 2-way merge, input data was in tape 1, read in M records at a time, sort them to become a run and write the runs alternatively to 2 tapes. Then do merge until the run length become N.
- polyphase Merge: using Fibonacci number only need k+1 tapes

  K^{th} order Fibonacci number:
  $F^k(N) = 0$ for $0 \le N \le k\text{-}2$, $F^k(k\text{-}1) = 1$
  $F^k(N) = F^k(N\text{-}1) + F^k(N\text{-}2) + .. + F^k(N\text{-}k)$
- replacement Selection: when using DeleteMin, write the smallest number out to a tape, then read another record from the input tape. If the data read is greater than the one just write out, it can include in this run. If the input is randomly distributed, replacement selection can produce runs of average length $2M$

Comparison Network

- parallel model of computation: many comparisons can be performed simultaneously
- a comparison network is comprised solely of wires and comparators
- a comparator is a device with two inputs, x and y, and two output x' and y', where $x' = \min(x, y)$ and $y' = \max(x, y)$
- a wire transmits a value from place to place
- a comparison network contains n input wires and n output wires, draw wires horizontally and comparators vertically, input on left and output right
- The run time of a comparison network is the depth of the network, where the depth of an input wire is 0, and a comparator's input wires have depth d_x and d_y, then its output wires have depth $\max(d_x, d_y) + 1$. The depth of a comparison network is the maximum depth of an output wire, or the maximum depth of a comparator.
- using comparison network to implement all kinds of sorting algorithm, e.g., insertion sort, merge sort (using Sorter[n]), etc.

Sorting Network

- a sorting network is a comparison network for which the output sequence is monotonically increasing for every input sequence
- the n-input, n-output sorting network in the family Sorter is named Sorter[n]
- the zero-one principle: if a sorting network works correctly when each input is drawn from the set {0, 1}, then it works correctly on arbitrary input numbers
- A bitonic sequence is a sequence that either monotonically increases and then monotonically decreases, or else monotonically decreases and then monotonically increases. A sequence that is either monotonically increasing or monotonically decreasing is also bitonic. The reversal of a bitonic sequence is bitonic, The zero-one bitonic sequences have the form $0^i1^j0^k$ or $1^i0^j1^k$, for some $i, j, k \ge 0$.
- A half-cleaner is a comparison network of depth 1 in which input line i is compared with line $i + n/2$, assume n is even. When a bitonic 0/1

sequence is applied as input to a half-cleaner, the half-cleaner produces an output sequence in which smaller values are in the top half, larger vales are in the bottom half, and both halves are bitonic. In fact, at least one of the halves is clean, i.e., consisting of either all 0's or all 1's.

- A bitonic sorter is comprised of several stages of half-cleaners to sort bitonic sequences. The first stage of Bitonic-Sorter[n] consists of Half-Cleaner[n] to produce two bitonic sequences of half the size such that every element in the top half is at least as small as every element in the bottom half, and using two copies of Bitonic-Sorter[$n/2$] to sort the two halves recursively. The depth $D(n)$ of Bitonic-Sorter[n] is $\lg n$ since $D(n) = 0$ if $n = 1$, otherwise $D(n) = D(n/2)+1$.
- A merging network merges two sorted input sequences into one sorted output sequence. Given two sorted sequences X and Y, if we reverse the order of the second sequence to Y^R and then concatenate the two sequences, the resulting sequence is bitonic. To merge X and Y, we simply perform bitonic sort on XY^R. We can construct Merger[n] by modifying the first half-cleaner of Bitonic-Sorter[n] by comparing input i with input n-i+1 (performing the reversal of the second input implicitly).
- The Sorter[n] uses the merging network to implement a parallel version of merge sort. For $k = 1, 2, \ldots, \lg n$, stage k consists of $n/2^k$ copies of merger[2^k]. The depth $D(n) = 0$ if $n = 1$, otherwise $D(n) = D(n/2) + \lg n$. Therefore, $D(n) = \Theta(\lg^2 n)$.