

Today:

- Sorting algorithms: why?
- Insertion, Shell's, Heap, Merge sorts

Readings for Today:

DS(IJ), Chapter 8;

Public Service Announcement: Business Revolution: A 360 degree Overview on the Business of Technology. Speakers will present their careers in Venture Capital, Corporate Management, and Entrepreneurship. March 23, 2006, 7-9PM, 2060 VLSB. Complimentary Food. Hosted by the Xi Pledge Class of $AK\Psi$

- Sorting supports searching
- Binary search standard example
- Also supports other kinds of search:
 - Are there two equal items in this set?
 - Are there two items in this set that both have the same value for property X?
 - What are my nearest neighbors?
- Used in numerous unexpected algorithms, such as convex hull (smallest convex polygon enclosing set of points).

Some Definitions

- A sort is a *permutation* (re-arrangement) of a sequence of elements that brings them into order, according to some *total order*. A total order, \preceq , is:
 - **Total:** $x \preceq y$ or $y \preceq x$ for all x, y .
 - **Reflexive:** $x \preceq x$;
 - **Antisymmetric:** $x \preceq y$ and $y \preceq x$ iff $x = y$.
 - **Transitive:** $x \preceq y$ and $y \preceq z$ implies $x \preceq z$.
- However, our orderings may allow unequal items to be equivalent:
 - E.g., can be two dictionary definitions for the same word: if entries sorted only by word, then sorting could put either entry first.
 - A sort that does not change the relative order of equivalent entries is called *stable*.

Classifications

- *Internal sorts* keep all data in primary memory
- *External sorts* process large amounts of data in batches, keeping what won't fit in secondary storage (in the old days, tapes).
- *Comparison-based* sorting assumes only thing we know about keys is order
- *Radix sorting* uses more information about key structure.
- *Insertion sorting* works by repeatedly inserting items at their appropriate positions in the sorted sequence being constructed.
- *Selection sorting* works by repeatedly selecting the next larger (smaller) item in order and adding it one end of the sorted sequence being constructed.

Sorting by Insertion

- Simple idea:
 - starting with empty sequence of outputs.
 - add each item from input, *inserting* into output sequence at right point.
- Very simple, good for small sets of data.
- With vector or linked list, time for find + insert of one item is at worst $\Theta(k)$, where k is # of outputs so far.
- So gives us $O(N^2)$ algorithm. Can we say more?

Inversions

- Can run in $\Theta(N)$ comparisons if already sorted.
- Consider a typical implementation for arrays:

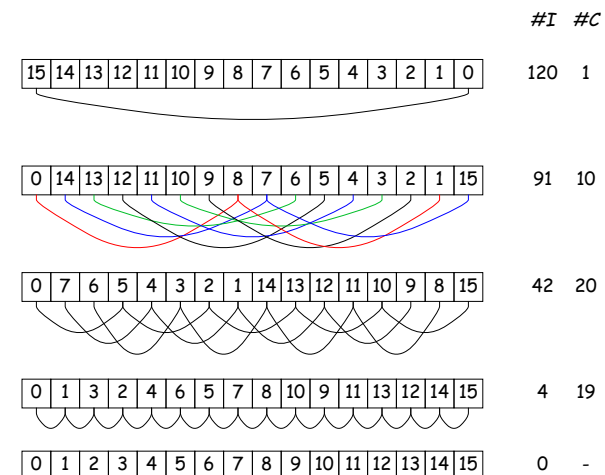

```
for (int i = 1; i < A.length; i += 1) {
    int j;
    Object x = A[i];
    for (j = i-1; j >= 0; j -= 1) {
        if (A[j].compareTo (x) <= 0) /* (1) */
            break;
        A[j+1] = A[j];
    }
    A[j+1] = x;
}
```
- #times (1) executes \approx how far x must move.
- If all items within K of proper places, then takes $O(KN)$ operations.
- Thus good for any amount of *nearly sorted* data.
- One measure of unsortedness: # of *inversions*: pairs that are out of order (= 0 when sorted, $N(N-1)/2$ when reversed).
- Each step of j decreases inversions by 1.

Shell's sort

Idea: Improve insertion sort by first sorting *distant* elements:

- First sort subsequences of elements $2^k - 1$ apart:
 - sort items #0, $2^k - 1, 2(2^k - 1), 3(2^k - 1), \dots$, then
 - sort items #1, $1 + 2^k - 1, 1 + 2(2^k - 1), 1 + 3(2^k - 1), \dots$, then
 - sort items #2, $2 + 2^k - 1, 2 + 2(2^k - 1), 2 + 3(2^k - 1), \dots$, then
 - etc.
 - sort items # $2^k - 2, 2(2^k - 1) - 1, 3(2^k - 1) - 1, \dots$,
 - Each time an item moves, can reduce #inversions by as much as $2^k + 1$.
- Now sort subsequences of elements $2^{k-1} - 1$ apart:
 - sort items #0, $2^{k-1} - 1, 2(2^{k-1} - 1), 3(2^{k-1} - 1), \dots$, then
 - sort items #1, $1 + 2^{k-1} - 1, 1 + 2(2^{k-1} - 1), 1 + 3(2^{k-1} - 1), \dots$,
 - :
- End at plain insertion sort ($2^0 = 1$ apart), but with most inversions gone.
- Sort is $\Theta(N^{1.5})$ (take CS170 for why!).

Example of Shell's Sort



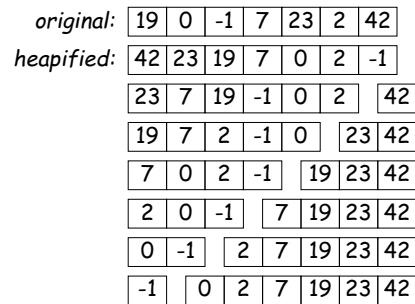
I: Inversions left.

C: Comparisons needed to sort subsequences.

Sorting by Selection: Heapsort

Idea: Keep selecting smallest (or largest) element.

- Really bad idea on a simple list or vector.
- But we've already seen it in action: use heap.
- Gives $O(N \lg N)$ algorithm (N remove-first operations).
- Since we remove items from end of heap, we can use that area to accumulate result:



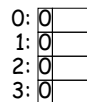
Merge Sorting

Idea: Divide data in 2 equal parts; recursively sort halves; merge results.

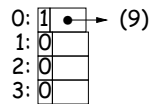
- Already seen analysis: $\Theta(N \lg N)$.
- Good for external sorting:
 - First break data into small enough chunks to fit in memory and sort.
 - Then repeatedly merge into bigger and bigger sequences.
 - Can merge K sequences of arbitrary size on secondary storage using $\Theta(K)$ storage.
- For internal sorting, can use binomial comb to orchestrate:

Illustration of Internal Merge Sort

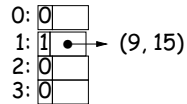
L: (9, 15, 5, 3, 0, 6, 10, -1, 2, 20, 8)



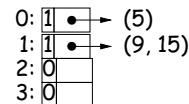
0 elements processed



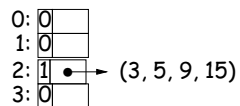
1 element processed



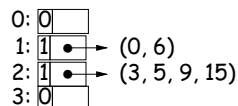
2 elements processed



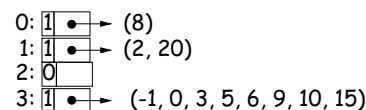
3 elements processed



4 elements processed



6 elements processed



11 elements processed