Sorting IV

Stable sorting
Radix Sort
Quick sort ← 6.06
Sorting races

Stable Sorting
Preserves input order among equal elements

4 1 3* 4 3
1 3* 3 4* 4

Counting sort is stable
Merge sort is stable

Selection sort: Find maximum element and put it at end of array (swap with element at end of array)

Heap: not stable!

3 2a 2b → 2b 2a 3

Define 2a < 2b
Radix Sort

- Herman Hollerith card-sorting machine for 1890 census
- Digit by Digit sort by mechanical machine

1) Examine given column of each card in a deck
2) Distribute the card into one of 10 bins
3) Gather cards bin by bin, so cards with first place punched are on top of cards with second place punched, etc.

MSB vs LSB?

Sort on most significant digit first or least significant digit first?

MSB strategy: Cards in 9 of 10 bins must be put aside, leading to a large number of intermediate piles

LSB strategy: Can gather sorted cards in bins appropriately to create a deck!
Digit sort needs to be stable, else will get wrong result!

**Analysis**

Assume counting sort is auxiliary stable sort. $\Theta(n+k)$ complexity.

Suppose we have $n$ words of $b$ bits each.

One pass of counting sort $\Theta(n+2^b)$

$b$ passes of counting sort $\Theta(b(n+2^b)) = \Theta(nb)$

$\frac{b}{n}$ passes $\Theta\left(\frac{b}{n}(n+2^r)\right)$ minimized when $r = \log n \quad \Theta\left(\frac{bn}{\log n}\right)$
QuickSort   "enrichment"

Divide: partition the array into two subarrays around a pivot \( x \) such that elements in lower subarray \( \leq x \leq \) elements in upper subarray. Linear time

Conquer: Recursively sort the two subarrays

Combine: Trivial

If we can choose a pivot such that two subarrays are roughly equal

\[
T(n) = 2T(n/2) + \Theta(n)
\]

\[
\Rightarrow T(n) = \Theta(n \log n)
\]

If one array is much bigger

\[
T(n) = T(n-1) + \Theta(n)
\]

\[
\Rightarrow T(n) = \Theta(n^2)
\]

Average case \( \Theta(n \log n) \) assuming input array is randomized!
**Sorting Races**

http://cg.scsc.carleton.ca/~morin/misc/sortalg

**Bubble sort**: Repeatedly step thru list to be sorted. Compare 2 items, swap if they are in the wrong order. Continue thru list, until no swaps. Repeat pass thru list until no swaps.

Θ(n²)

**Shell sort**: Improves insertion sort by comparing elements separated by gaps.

Θ(n log²n)

**Bubble sort** vs. **Insertion Sort**

- last element put in place
- each element put in right place

**Heap sort** vs. **Merge sort** vs. **QuickSort**