Linear-time Sorting

Note Title 9/27/2007

Last time: Any comparison-based sorting alg has worst-case complexity $\Omega(n \log n)$

Can we sort in a way not based on comparing elements?

Consider sorting n elements that are all integers $\{0, 1, \ldots, K-1\}$

E.g. $K = 10$

Idea: keep 10 lists & put all elements of value i into list i

Using an array: Count # of occurences of each element & then you can put them in order

Counting Sort

basic idea we've just described.

We'll need one more property

Stable sort - any two equivalent element are kept in same relative order

d, a, b, c, b $\xrightarrow{\text{stable}}$ a, b, b, c, d

Radix

```
CountingSort(input, output, k) {
for d = 0 to #digits-1
    int n = input.length;
    for (i=0; i<k; i++)
        count[i] = 0;
    for (j=0; j<n; j++)
        count[input[j]]++;
    for (i=1; i<k; i++)
        count[i] += count[i-1];
    for (j=n-1; j>=0; j--)
        output[--count[input[j]]] = input[j];
}
```

array of same length

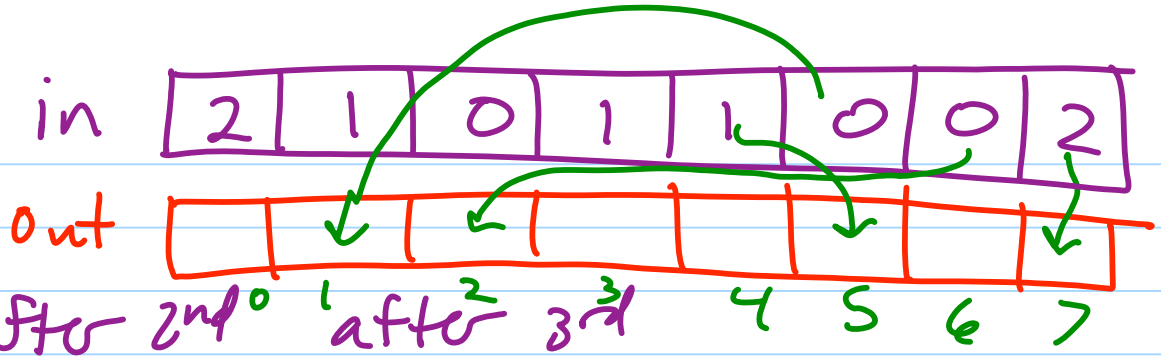
at end
count[i] is # of elements that are i

count[i] is # of elements ≤ i

radix sort

digitizer.getDigit(input[j], d) current digit

k=3



count	after 1st loop	after 2nd loop	after 3rd loop
0	0	3	~~3~~ ~~2~~ 1
1	0	3	~~6~~ 5
2	0	2	~~8~~ 7

Time complexity

$\Theta(n+k)$

two loops over counters $\Theta(k)$
two loops over elements $\Theta(n)$

When $k = O(n)$ then this is a $\Theta(n)$ sort.

↖ linear time

Suppose I want to sort social security number

9 digits

What would "k" be if we wanted to use counting sort? 10^9