Counting Sort

Adapted from the CLRS book slides

SORTING IN LINEAR TIME

Counting sort

Depends on a *key assumption*: numbers to be sorted are integers in $\{0, 1, \ldots, k\}$.

Input: $A[1:n]$, where $A[j] \in \{0, 1, \ldots, k\}$ for $j = 1, 2, \ldots, n$. Array A and values n and k are given as parameters.

Output: $B[1:n]$, sorted.

Auxiliary storage: $C[0:k]$

PSEUDOCODE

```
COUNTING-SORT(A, n, k)
let B[1 : n] and C[0 : k] be new arrays
for i = 0 to k
    C[i] = 0
for j = 1 to n
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i.
// Copy A to B, starting from the end of A.
for j = n downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] − 1    // to handle duplicate values
return B
```

SORTING IN LINEAR TIME (continued)

Do an example for $A = \langle 2_1, 5_1, 3_1, 0_1, 2_2, 3_2, 0_2, 3_3 \rangle$. *[Subscripts show original order of equal keys in order to demonstrate stability.]*

i	0	1	2	3	4	5
$C[i]$ after second **for** loop	2	0	2	3	0	1
$C[i]$ after third **for** loop	2	2	4	7	7	8

Sorted output is $\langle 0_1, 0_2, 2_1, 2_2, 3_1, 3_2, 3_3, 5_1 \rangle$.

SORTING IN LINEAR TIME (continued)

Counting sort is ***stable*** (keys with same value appear in same orde
they did in input) because of how the last loop works.

Analysis

$\Theta(n + k)$, which is $\Theta(n)$ if $k = O(n)$.

How big a k is practical?

- Good for sorting 32-bit values? No.
- 16-bit? Probably not.
- 8-bit? Maybe, depending on n.
- 4-bit? Probably (unless n is really small).

Counting sort will be used in radix sort.