For solutions of homeworks (all weeks)
+ lecture notes (this week only)
⇒ TA Jenny Lam's 161 web page

Integer sorting
model: collection of n records (objects)
each having an integer sorting key
(stored with object ... computed from object ...
or keys might be in auxiliary dictionary structure)

keys belong to some limited range
today: $0 \le \text{key} < K$ for some parameter K
(so there are K diff. key values)
other algorithms: keys are machine integers —
can perform arithmetic on them
in const. time per operation

Fastest known time bounds

random. $O(n\sqrt{\log\log n})$	$O(n \log(\frac{\log K}{\log n}))$	$O(n\sqrt{\log\log K})$	
bucket sort ($O(n+K)$)	radix sort $O(n \frac{\log K}{\log n})$		

Bucket sort — typical application

has $K \leq n$ — so fewer keys than data items, many repeated keys

e.g. 330 students — sort by (rounded) course average — number in range $0 \leq x \leq 100$ ($n = 330$ $K = 101$)

```
def bucketsort(L, n, K):                    (max key)
    (if just given L, then n=len(L) K=max(L)+1)
    create an array B of K buckets                    O(K)
        B[i] = new empty list for all 0 ≤ i < K
    for each x in L:                                  O(n)
        add x to end of bucket B[key(x)]
    output = empty list                               for linked lists - O(K)
    for i = 0, 1, 2, ... K-1:                         for array list/Python lists
        concatenate B[i] onto end of output           O(n)
```

total $O(n \times K)$

data: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

key: 2 1 4 3 9 8 6 2 0 1 5 7 2 3 8 4 1 4 3 6 7 9 5 8 7 0

n = 26, K = 10

output: I Z B J Q A H M D N S C P R

K W G T L U Y F O X E V

To use bucket sort as a subroutine in radix sort (next time):

need a key property "stability"

property that some sorting algorithms have, others don't

For any two items with same key ordering in output should be same as ordering in input

E.g. in example: A, H, M all have key = 2

input order: A H M

output: A H M ✓

Bucket sort as described here —
stable because
- when we loop through items adding them to buckets, we use input order
- adding to end of bucket preserves order
- concatenation preserves order

Heapsort — not stable as described
heapify scrambles ordering

Quicksort - in-place partition - not stable
(swaps reverse orderings)
quicksort that partitions stably
(preserving ordering of sets < pivot and > pivot)
is stable

in-class version doesn't work with equal keys

Mergesort - stable if we partition
first $\lfloor n/2 \rfloor$ then second $\lceil n/2 \rceil$ items
+ break ties in merge step by
selecting elements from first list

To make a comparison sort stable:

- store original position of each item
- if any comparison finds equal keys, break tie using positions