- Questions
- What is sorting and where do we encounter it?
  - Google warmest down jacket
  - Having sorted data may make future operations much more efficient
- Why study sorting in this class?
  - Not to implement—standard libraries implement, see list.sort()
  - Excellent context for practicing analysis and design decisions
  - Very practical: you will be expected to know it
- How would you sort a list?
  - Bubble sort: go through the list swapping out of order elements
    - on the first pass, largest element "bubbles" up to the end
    - repeat this process, once you make a pass with no swaps, the list is sorted
  - Selection sort: find the smallest element, swap it to the beginning
    - repeat with finding the second smallest, and so on
    - "selects" the smallest element

Bogosort

- randomize the list until it ends up sorted

insertion sort

- diagram (SLIDE)

pseudocode

- for i from 1 to n-1
    find where element i should be inserted into the sorted portion of the list (0 to i-1)
    insert element i and shift other elements over

- **quick check**: fill in table (SLIDE)

worst-case analysis

- just carefully count up the steps
- i=1, 1 comparison + 1 shift
  i=2, 2 comparisons + 2 shifts
  i=3, 3 comparisons + 3 shifts
  ...
  i=n-1, n-1 comparisons + n-1 shifts
  sum of 1..n-1 is n(n-1)/2
  n(n-1)/2 + n(n-1)/2
  (n^2 - n)/2 + (n^2 - n)/2
  n^2 - n
  O(n^2)

Analysis practice

```python
def contains(nums, x):
    for num in nums:
        if num == x:
            return True
    return False
```

- O(n)

```python
def max(a, b):
    """assume a and b are numbers"""
    if a >= b:
        return a
    return b
```

- O(1), same number of operations no matter the input

```python
def required_bits(x):
    bits = 0
    while x >= 1:
        bits += 1
        x = x / 2
    return bits
```

- $O(\log_2(n))$

selection sort

- diagram (SLIDE)
- pseudocode
  - for i from 0 to n-2
    find index of smallest element, j, in range i to n-1
    swap elements at i and j
- **quick check**: fill in table (SLIDE)
- worst-case analysis
  - i=0, n-1 comparisons + 1 swap
    i=1, n-2 comparisons + 1 swap

i=2, n-3 comparisons + 1 swap

…

i=n-2, 1 comparisons + 1 swap

n(n-1)/2 + n

(n^2 - n)/2 + n

n^2/2 - n/2 + n

n^2/2 + n/2

O(n^2)