Simple sorting II

- bubblesort is very simple, but not very efficient - typically not used except on small sets of data
- a couple of other simple (but not very efficient) sorting algorithms include insertionsort and selectionsort
- insertion sort builds sorted list one element at a time, taking each element and inserting it in its correct position in the list so far
- selectionsort also builds one element at a time, at each step getting the next smallest remaining value

insertion sort

- goes through list of unsorted elements one at a time
- builds a sorted list as it goes
- each new element goes in its correct position in the list built so far (searches through the sorted-so-far list, looking for right position)
- once the last element has been processed the entire list will have been sorted

insertion sort example

- suppose initial values were (17, 3, 11, 9, 6, 10)
- first gets 17, only item in sorted list so far (17), values still to be sorted are (3,11,9,6,10)
- takes 3, places in correct position: (3,17), (11,9,6,10)
- takes 11, places in correct position: (3,11,17), (9,6,10)
- takes 9, places in correct position: (3,9,11,17), (6,10)
- takes 6, places in correct position: (3,6,9,11,17), (10)
- takes 10, places in correct position (3,6,9,10,11,17)
- done!

insertion sort approach

- given array to sort, sort “in place”, taking next element to process and working backwards through sorted array, swapping elements until correct position is reached
- e.g. suppose in previous example we are at the point of (3,11,17), (9,6,10), about to process the 9
- suppose it's all in one array, [3, 11, 17, 9, 6, 10], our current array position is 3 (where the 9 is) and we know what's in positions 0..2 is already sorted
- we keep swapping 9 backwards while it is smaller than what's before it: [3,11,17,9,6,10] =>[3,11,9,17,6,10]=>[3,9,11,17,6,10]

insertion sort algorithm

```
// initially element in first position is treated as a sorted 1-element list

currPos=1 // position of next element to be sorted

while currPos < N

    index=currPos // current position of element we're moving backwards

    while index > 0 and array[index-1] > array[index]
        // still need to keep moving backwards, swap into previous position
        swap(array[index], array[index-1])
        index--

    currPos++ // moves on to next element we'll be sorting
```

usefulness of insertion sort

- while this isn't more efficient than bubblesort, there are times it can be handy
- suppose the user frequently does lookups on stored data, but once in awhile needs to insert one more element
- we want to keep the data sorted for the sake of all those lookups
- if the array is already sorted, then when they add one new element we can put it at the end of the existing ones, and just use the inner “insert” loop to get it to the right spot
- this is less costly than completely re-sorting

selection sort

- filling positions of sorted list one element at a time
- at each position, go through the remaining unsorted elements, pick smallest, and move to current position
- can work “in place” in an array if we “move the element” by swapping it with what's in the position we're trying to fill

selection sort example

- suppose initial array contents were [17, 3, 11, 9, 6, 10]
- pos 0: find smallest element in positions 0..5: i.e. 3 from pos 1. swap it with what's in position 0 (i.e. 17), giving [3,17,11,9,6,10]
- pos 1: find smallest element in positions 1..5, i.e. the 6, and swap with what's in position 1 (17), giving [3,6,11,9,17,10]
- pos 2: find smallest (9) and swap with what's in position 2 (11), giving [3,6,9,11,17,10]
- pos 3: find smallest (10) swap with (11), gives [3,6,9,10,17,11]
- pos 4: find smallest (11), swap with (17), gives [3,6,9,10,11,17]
- can stop after filling second to last position (biggest is in last)

selection sort algorithm

```
// assuming we're sorting array of size N
currPos = 0
while currPos < N-1
    // find next smallest
    smallest = array[currPos]
    smallPos = currPos
    index = currPos + 1
    while index < N
        if array[index] < smallest
            smallest = array[index]
            smallPos = index
        index++
    // swap next smallest with whatever used to be in current position
    swap(array[currPos], array[smallPos])
    currPos++ // move on to next position to be filled
```