Sorting

This handout explains three quadratic sorting algorithms. Each explanation assumes we are sorting an array of size n in ascending order from position position [0] through [n-1]. (Technically, we should say "non-decreasing" rather than "ascending", since we could have duplicates.)

Bubble Sort

As was demonstrated in class, the main idea of bubble sort is as follows.

- Make n-1 passes through the array, numbered i=n-1, n-2, etc. down to 1
  - On pass i, we consider the part of the array 0 through i
    We walk through that part of the array, looking at each pair of adjacent elements, and swapping them if they are out of order.
  - We also can keep track of whether any swaps happen during each pass. If we make a pass that has NO swaps at all, we can stop, because we know the array is already sorted.

Some things we know about bubble sort:

- After pass i, we know that the element in position i through the end of the array (n-1) is in proper place in the array. (This was explained in lecture.)
  - Corrolary 1: After pass 1, we know that elements 1 through n-1 are sorted and in their correct place in the array.
  - Corrolary 2: After pass 1, we know the entire array is sorted, because it is not possible for element [0] to be out of place if every other element is in place.

Given this, here are two example "worked problems" for bubble sort.

The part in bold is what YOU would write if given the problem.

initial values	60	50	40	30	20
i=4	**50**	**40**	**30**	**20**	**60**
i=3	**40**	**30**	**20**	**50**	**60**
i=2	**30**	**20**	**40**	**50**	**60**
i=1	**20**	**30**	**40**	**50**	**60**

initial values	10	40	30	60	50
i=4	**10**	**30**	**40**	**50**	**60**
i=3	**10**	**30**	**40**	**50**	**60**
i=2	DONE: no swaps on previous pass				

Insertion Sort

As demonstrated in class, the main idea of insertion sort is that we take elements from one array, and insert them, in turn, into a sorted array that we are building up one element a time. The simplest approach is to insert into a completely separate array. Later, we'll discuss "in-place" approaches to insertion sort. Each "pass" is the insertion of one new element, and it has two parts (a) figure out into which index the new element should go (b) copy all the elements from the end of the array down to that position one index higher (c) put the new element in place.

Here are two examples of solved problems using insertion sort.

initial values	60	50	20	30	10
i=0	**60**				
i=1	**50**	**60**			
i=2	**20**	**50**	**60**		
i=3	**20**	**30**	**50**	**60**	
i=4	**10**	**20**	**30**	**50**	**60**

initial values	10	40	30	60	50
i=0	**10**				
i=1	**10**	**40**			
i=2	**10**	**30**	**40**		
i=3	**10**	**30**	**40**	**60**	
i=4	**10**	**30**	**40**	**50**	**60**

Selection Sort

As demonstrated in class, the main idea of selection sort is to first choose whether we are going to work from the "big end" or the "little end".

If we work from the "big end", we have an outer loop where i goes from n-1 down to 1.

- for i = n-1 down to 1
  - Look at portion of array from 0 through i, and find the "index of the maximum element"
  - Swap a[i] with a[indexOfMax]

Here are two solved problems for selection sort:

initial values	60	90	70	50	10
i=4	60	10	70	50	90
i=3	60	10	50	70	90
i=2	50	10	60	70	90
i=1	10	50	60	70	90

initial values	10	40	30	60	50
i=4	**10**	**40**	**30**	**50**	**60**
i=3	**10**	**40**	**30**	**50**	**60**
i=2	**10**	**30**	**40**	**50**	**60**
i=1	**10**	**30**	**40**	**50**	**60**