Questions

1. Suppose we want to sort a list from smallest to largest. Show the contents of the array after each pass through the outer loop for the following algorithms, and the list of five elements.

   (a) bubble sort

   (b) selection sort

   (c) insertion sort

index	initial	after 1	after 2	.....
0	3.2			
1	4.1			
2	-1.0			
3	6.0			
4	-5.6			

2. When is insertion sort slowest?

3. When is selection sort slowest?

4. Consider the problem of finding repeated elements in a list. For example, suppose the list contains numbers and you want to determine which of the numbers occurs two or more times in the list. How would you solve this problem? Assume that you cannot use any space other than the list, and a few temporary variables.

   Here I am not asking for a Java solution. Just give the basic idea. Also consider how much time it takes as a function N of the number of elements in the list.

Answers

1. (a) bubblesort

```
index   initial   after 1   after 2   after 3   after 4
-----   -------   -------   -------   -------   -------
0        3.2       3.2      -1.0       -1.0      -5.6
1        4.1      -1.0       3.2       -5.6      -1.0
2       -1.0       4.1      -5.6        3.2       3.2
3        6.0      -5.6       4.1        4.1       4.1
4       -5.6       6.0       6.0        6.0       6.0
```

and it requires one more pass to ensure all is in order (and nothing swaps).

(b) selection sort (The * indicates that the array is sorted up to that element.)

```
index  initial    after 1  after 2    after 3    after 4
-----  ------    -------   -------    -------   -------
0      3.2         -5.6*     -5.6      -5.6       -5.6
1      4.1          4.1      -1.0*     -1.0       -1.0
2     -1.0         -1.0       4.1       3.2*       3.2
3      6.0          6.0       6.0       6.0        4.1*   and we're done
4     -5.6          3.2       3.2       4.1        6.0
```

(c) insertion sort (The * indicates that the array is sorted up to that element.)

```
index   initial   after 1   after 2   after 3   after 4
-----   -------   -------   -------   -------   -------
0        3.2*      3.2      -1.0      -1.0      -5.6
1        4.1       4.1*      3.2       3.2      -1.0
2       -1.0      -1.0       4.1*      4.1       3.2
3        6.0       6.0       6.0       6.0*      4.1
4       -5.6      -5.6      -5.6      -5.6       6.0*
```

2. Insertion sort is slowest when the initial list is sorted *in the wrong order* since in that case the maximum number of shifts needs to be done in each pass through the outer loop.

3. Selection sort always takes the same amount of time (independent of the ordering). It always examines the same number of pairs of elements.

4. Here is a simple way to do it. It takes time $O(N^2)$ which isn't very efficient, but it works. Later in the course we'll discuss another way to do it (hash tables) that is faster.

```
for i = 1 to N-1     /// elements indexed from 0 to N-1
    for j = 0 to i
        if list[i] == list[j]
           print list[i]
```

Note that if more than two copies of an element is in the list, then this element will be printed out more than once.