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. For which input is insertion sort slowest?

3. For which input is selection sort slowest?

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

(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. Every time a new element at position i is reached, all other elements at positions 0 to $i-1$ have to be shifted.

3. On the one hand, one could answer that selection sort always takes the same amount of time (independent of the element ordering), since it always has to examine the same number of pairs of elements. On the other hand, selection sort has to do more than compare elements: there is a bit of extra work to do to swap elements or even adjust what is the currently smallest minimum in the rest. Again, if the elements are in the reversed order, then this extra is maximized. Note however that this extra work is constant for each comparison. The number of comparisons remains $O(N^2)$ and so the total work (number of things to do) also is $O(N^2)$.

