Bubblesort

Popular method but not efficient. It works by repeatedly swapping adjacent elements that are out of order.

```
BUBBLESORT(A)
1  for i = 1 to A.length − 1
2      for j = A.length downto i + 1
3          if A[j] < A[j − 1]
4              exchange A[j] with A[j − 1]
```

If input is array A, then output will be denoted by A' with elements satisfying properties:

$$A'[1] \leq A'[2] \leq \ldots \leq A'[n] \qquad (1)$$

To show that bubblesort works correctly we need to establish (1) and also show that A' is a permutation of elements from A.

Loop invariant #1 (lines 2-4). At the start of each iteration of the for loop in lines 2-4,

$A[j] = \min\{A[k] : j \le k \le n\}$ and the subarray $A[j..n]$ is a permutation of elements that were originally in $A[j..n]$ at the time when loop started.

Initialization. $j=n$, subarray $A[j..n]$ consists only of one element $A[n]$. The loop invariant holds trivially.

Maintenance. Fix index j. By assumption, $A[j]$ is min element in $A[j..n]$ and $A[j..n]$ is a permutation of elements at the time loop started. Lines 3 and 4 exchange values of $A[j]$ and $A[j-1]$ if $A[j]$ is smaller than $A[j-1]$. If $A[j]$ was the min element in $A[j..n]$, then since we have only one possible exchange, $A[j-1]$ will become the smallest in $A[j-1..n]$. Since $A[j..n]$ is a permutation of elements at the start of the loop, by possibly exchanging $A[j]$ with $A[j-1]$, we get $A[j-1..n]$ also being a permutation of

elements from $A[j-1..n]$ present at the start of the loop.

Termination. At the end of the for loop, $j=i \Rightarrow A[i]$ is the smallest element in $A[i..n]$ and $A[i..n]$ is a permutation of elements from $A[i..n]$ present at the start of the loop.

Loop invariant #2 for lines 1–4

At the start of each iteration in for loop of lines 1–4, the subarray $A[1..i-1]$ consists of $i-1$ smallest elements that were originally present in $A[1..n]$, in sorted order, and $A[i..n]$ will have $n-i+1$ remaining elements of $A[1..n]$.

Initialization. $i=1 \Rightarrow A[1..\overset{=0}{1-1}]$ is empty array. Trivially satisfied.

Maintenance. Fix i. We assume that $A[1..i-1]$ contains $i-1$ smallest elements originally present (in sorted order)

in $A[1..n]$. We showed that loop invariant #1 when it finishes has $j=i$ and $A[i]$ is the smallest element from $A[i..n]$. Hence, $A[1..i]$ will contain i smallest elements originally present in $A[1..n]$. Also, from loop invariant #1 with $j=i$, $A[i..n]$ is a permutation of elements, hence, $A[i+1..n]$ is also a permutation, and $A[i+1..n]$ contains the rest $n-i$ of elements.

Termination. The for loop of lines 1-4 terminates when $i=n$, so that $i-1=n-1$. By the statement of the loop invariant, $A[1..i-1]$ is the subarray $A[1..n-1]$, and it consists of the $n-1$ smallest values originally in $A[1..n]$, in sorted order. The remaining element must be the largest value in $A[1..n]$, and it is $A[n]$. Therefore, the entire array $A[1..n]$ is sorted.

Let's analyze the running time for the bubblesort algorithm.

$n-(i+1)+1 = n-i$

```
BUBBLESORT(A)
1  for i = 1 to A.length − 1
2      for j = A.length downto i + 1        n .. i+1
3          if A[j] < A[j − 1]
4              exchange A[j] with A[j − 1]
```

The running time depends on the number of iterations of the <u>for</u> loop of lines 2–4. For a given value of i, this loop makes $n-i$ iterations, and i takes on the values $1, 2, \ldots, n-1$. The total number of iterations, therefore, is

$$\sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n(n-1) - \frac{n(n-1)}{2} =$$

$$= \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} \sim \frac{n^2}{2} \text{ for large } n.$$

Thus, the running time of bubblesort is $\Theta(n^2)$ in all cases. The worst-case running

time is the same as that of insertion sort.

Horner's Rule

Consider the n^{th} degree polynomial

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n, \quad a_n \neq 0$$

Given coefficients $a_0, a_1, \dots, a_n$ and x, evaluate $P(x)$.

Naïve evaluation: compute all powers of x, multiply by coefficients and add.

$$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x \cdot x + a_3 \cdot x \cdot x \cdot x + \dots$$

1 mult, 2 mult, 3 mult

$$\dots + a_n \cdot \underbrace{x \cdot x \dots x}_{n \text{ times}}$$

n mult

additions $= n$

multiplications $= 1 + 2 + \dots + n = \frac{n(n+1)}{2}$

A better way to compute powers of x, ie. x^4, is to use x^{n-1}:

$$x^n = x^{n-1} \cdot x$$

$$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x \cdot x + a_3 \cdot \underbrace{x \cdot x^2}_{\| \; x^3} + a_4 \cdot x \cdot x^3 + \dots$$

$$\dots + a_n \cdot x \cdot x^{n-1}$$

$\#$ add $= n$

$\#$ mult $= 1 + \underbrace{2 + 2 + \dots + 2}_{n-1 \text{ times}} = 2(n-1) + 1 = 2n - 1$

Horner's rule / Nested multiplication

$$P_3(x) = a_0 + a_1 x + a_2 x^2 = a_0 + x(a_1 + a_2 \cdot x)$$: 2 mult

3 mult

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + a_n \cdot x) \dots))$$

n mult

$\#$ mult $= n$

```
1  y = 0
2  for i = n downto 0
3      y = a_i + x · y
```

Inversions

Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an *inversion* of A.

(a) List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.

$A = \langle \overset{1}{2}, \overset{2}{3}, \overset{3}{8}, \overset{4}{6}, \overset{5}{1} \rangle$

The inversions are $(1,5), (2,5), (3,4), (3,5), (4,5)$.
(Remember that inversions are specified by indices rather than by the values in the array.)

(b) What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions?

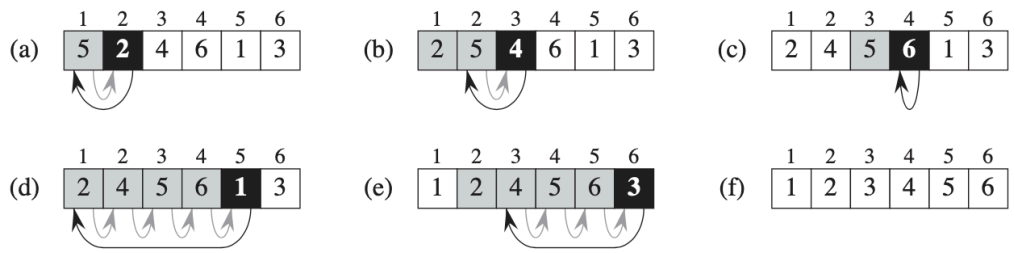
The array with elements from $\{1, 2, \ldots, n\}$ with the most inversions is $\langle n, n-1, n-2, \ldots, 2, 1 \rangle$.
For all $1 \le i < j \le n$, there is an inversion (i, j).

The number of such inversions is

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{(n-2)!(n-1)n}{2!(n-2)!} = \frac{n(n-1)}{2}$$

(c) What is the relationship between the running time of insertion sort and the number of inversions in the input array?

Recall the insertion sort:



INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Suppose that the array A starts out with an inversion (k, j). Then $k < j$ and $A[k] > A[j]$. At the time that the outer for loop of lines 1–8 sets $key = A[j]$, the value that started in $A[k]$ is still somewhere to the left of $A[j]$. That is, it's in $A[i]$, where $1 \le i < j$, and so the inversion has become (i, j). Some iteration of the while loop of lines 5–7 moves $A[i]$ one position to the right. Line 8 will eventually drop key to the left of this element, thus eliminating the inversion. Because line 5 moves only elements that are greater than key, it moves only elements that correspond to inversions. In other words, each iteration of the while loop of lines 5–7 corresponds to the elimination of one inversion.