Counting basic steps: Insertion sort

We count the number of basic steps for insertion sort (to sort an array `b`) in two different situations: the best and worst cases. We also discuss the expected case. Let `n = b.length`.

Recall that insertion sort has this postcondition:

Postcondition: `b[0..n-1]` is sorted

The algorithm, in the box to the right, is developed with this invariant:

Invariant `P`: `0 ≤ k ≤ n` and `b[0..k-1]` is sorted.

```
k= 0;
// Invariant P (to the left)
while (k != b.length) {
   //Push b[k]down to its sorted
   //        position in b[0..k];
   j= k;
   while (0 < j  &&  b[j-1] > b[j]) {
        swap b[j-1] and b[j];
        j= j – 1;
    }

   k= k+1;
}
```

To simplify the discussion, we have written part of the code in blue and part in orange. Blue is: initialization to truthify the invariant, the outer while-loop condition, and the increment of `k` to make progress toward termination. This part is always executed the same way, and we can easily determine how many basic steps are executed:

`k= 0;`	`1` basic step
`k!=b.length`	`n+1` basic steps (true `n` times, false once)
`k= k+1;`	`n` basic steps (executed `n` times).

The orange part is the code needed to ensure that the loop invariant remains true when `k` is incremented. This code is executed differently depending on the values in the array, and we handle two different cases and discuss a third.

1. Best-case analysis. Array `b` is already sorted, so that `b[j-1] ≤ b[j]` is always true. We count the number of times each basic step is executed:

`j= k;`	`n` times (once per iteration of the main loop)
`0 < j  && b[j-1] > b[j]`	`n` times (once per each iteration of the main loop, and it is always false)
Swap `b[j-1]` and `b[j]`	never executed
`j= j-1;`	never executed

Adding up the blue and orange basic steps, we get:

`4n + 2` basic steps. The algorithm takes time linear in `n`.

2. Worst-case analysis. Array `b` is in descending order, so that `b[j-1] > b[j]` is always true. This means the loop condition `0 < j  &&  b[j-1] > b[j]` is false only when `0 = j`. Therefore, `k` iterations of the inner loop are executed. For example, suppose that `k` is `10` when the orange code it to be executed. Then `10` iterations of the inner loop will be executed.

We count the number of times each basic step is executed. This is more complicated than in the best-case analysis because the number of orange basic steps is different at each iteration of the outer loop.

`j= k;`	`n` times (once per iteration of the main loop)
`0 < j  && b[j-1] > b[j]`	Iteration with `k = 0`, `1` time; with `k = 1`, `2` times, … with `k = n-1`, `n` times
Swap `b[j-1]` and `b[j]`	Iteration with `k = 0`, `0` time; with `k = 1`, `1` times, … with `k = n-1`, `n-1` times
`j= j-1;`	Iteration with `k = 0`, `0` time; with `k = 1`, `1` times, … with `k = n-1`, `n-1` times

We get these numbers of orange basic steps, showing first the contribution from each line above:

```
      1 +
      (0 + 1 + … + n) +
      (0 + 1 + … + (n-1) +
      (0 + 1 + … + (n-1)
=       <using this formula three times:  1+2+…+h = h (h+1)/2>
      1 + n (n+1)/2 + n(n-1)/2 + n(n-1)/2
=       <arithmetic>
      1 + n(n+1)/2 + n(n-1)
```

Counting basic steps: Insertion sort

Adding in the blue basic steps shown near the top of the previous page, we get this many basic steps in the worst case:

$$n(n+1)/2 + n(n-1) + 2n + 3$$

This number is *quadratic* in `n`, it is proportional to n^2.

3. Average- or expected-case analysis. We are usually interested in the average-case analysis, often call the expected-case analysis because it is was is usually expected when the algorithm is run. This kind of analysis is usually too complicated to carry out in detail in this course CS2110. It consists basically of looking at the all possible ways in which array `b` could be configured with different values, finding the number of basic steps with each configuration, and then averaging them.

But we can get an informal idea on this as follows. In the best case, execution of the orange loop above does 0 iterations. In the worst case, it does k iterations. On the average, we can expect it to do `k/2` iterations. If we carry out the above kind of analysis, assuming that `k/2` iterations of the inner loop are executed, we will find out that the number of basic steps executed is proportional to n^2.