Analysis of Insertion Sort

Antonio Carzaniga

Faculty of Informatics
University of Lugano

February 28, 2012

Outline

- Sorting
- Insertion Sort
- Analysis
Sorting

- **Input:** a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$

- **Output:** a sequence $\langle b_1, b_2, \ldots, b_n \rangle$ such that
  - $\langle b_1, b_2, \ldots, b_n \rangle$ is a permutation of $\langle a_1, a_2, \ldots, a_n \rangle$
  - $\langle b_1, b_2, \ldots, b_n \rangle$ is sorted

\[
b_1 \leq b_2 \leq \cdots \leq b_n
\]

- Typically, $A$ is implemented as an array

\[
A = \begin{pmatrix}
0 & 8 & 3 & 8 & 6 & 11 & 8 & 9 & 42 & 3 & 4 & 5 & 6 & 6 & 7 & 8 & 9 & 11
\end{pmatrix}
\]

© 2006-2007 Antonio Carzaniga

Insertion Sort

- **Idea:** it is like sorting a hand of cards
  - scan the sequence left to right
  - pick the value at the current position $a_j$
  - insert it in its correct position in the sequence $\langle a_1, a_2, \ldots, a_{j-1} \rangle$ so as to maintain a sorted subsequence $\langle a_1, a_2, \ldots, a_j \rangle$

\[
A = \begin{pmatrix}
0 & 3 & 4 & 8 & 8 & 9 & 11 & 12 & 14 & 20
\end{pmatrix}
\]

© 2006-2007 Antonio Carzaniga
Insertion Sort (2)

Insertion-Sort($A$)
1. for $i = 2$ to length($A$)
2. \hspace{1em} $j = i$
3. \hspace{1em} while $j > 1$ and $A[j - 1] > A[j]$
4. \hspace{2em} swap $A[j]$ and $A[j - 1]$
5. \hspace{1em} $j = j - 1$

- Is Insertion-Sort correct?
- What is the time complexity of Insertion-Sort?
- Can we do better?

Complexity of Insertion-Sort

Outer loop (lines 1–5) runs exactly $n - 1$ times (with $n =$ length($A$))

- What about the inner loop (lines 3–5)?
  - best, worst, and average case?
Complexity of Insertion-Sort (2)

Insertion-Sort(A)
1 for i = 2 to length(A)
2     j = i
3     while j > 1 and A[j - 1] > A[j]
4          swap A[j] and A[j - 1]
5          j = j - 1

■ **Best case:** the inner loop is *never* executed
  ▶ what case is this?

■ **Worst case:** the inner loop is executed exactly $j - 1$ times for every iteration of the outer loop
  ▶ what case is this?

© 2006–2007 Antonio Carzaniga 7

Complexity of Insertion-Sort (3)

■ The worst-case complexity is when the inner loop is executed exactly $j - 1$ times, so

$$T(n) = \sum_{j=2}^{n} (j - 1)$$

$T(n)$ is the *arithmetic series* $\sum_{k=1}^{n-1} k$, so

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

**[T(n) = \Theta(n^2)]**

■ Best-case is $T(n) = \Theta(n)$

■ Average-case is $T(n) = \Theta(n^2)$

© 2006–2007 Antonio Carzaniga 8
Correctness

- Does Insertion-Sort terminate for all valid inputs?
- If so, does it satisfy the conditions of the sorting problem?
  - $A$ contains a permutation of the initial value of $A$
  - $A$ is sorted: $A[1] \leq A[2] \leq \cdots \leq A[\text{length}(A)]$
- We want a formal proof of correctness
  - does not seem straightforward...

The Logic of Algorithmic Steps

Example:

```plaintext
SortTwo(A)
1  // A must be an array of 2 elements
3     t = A[1]
5     A[2] = t
```
Loop Invariants

- We formulate a \textit{loop-invariant} condition $C$
  - $C$ must remain true \textit{through} a loop
  - $C$ is relevant to the problem definition: we use $C$ at the end of a loop to prove the correctness of the result

- Then, we only need to prove that the algorithm terminates

Loop Invariants (2)

- Formulation: this is where we try to be smart
  - the \textit{invariant} must reflect the \textit{structure} of the algorithm
  - it must be the basis to prove the correctness of the solution

- Proof of validity (i.e., that $C$ is indeed a loop invariant): typical \textit{proof by induction}
  - \textit{initialization}: we must prove that the \textit{invariant $C$ is true before entering the loop}
  - \textit{maintenance}: we must prove that \textit{if $C$ is true at the beginning of a cycle then it remains true after one cycle}
Loop Invariant for Insertion-Sort

Insertion-Sort($A$)
1 \textbf{for} $i = 2$ \textbf{to} length($A$) \\
2 \hspace{1em} $j = i$ \\
3 \hspace{1em} \textbf{while} $j > 1$ \textbf{and} $A[j - 1] > A[j]$ \\
4 \hspace{2em} \textbf{swap} $A[j]$ \textbf{and} $A[j - 1]$ \\
5 \hspace{2em} $j = j - 1$

- The main idea is to insert $A[i]$ in $A[1..i - 1]$ so as to maintain a \textit{sorted subsequence} $A[1..i]$

- \textit{Invariant:} (outer loop) the subarray $A[1..i - 1]$ consists of the elements originally in $A[1..i - 1]$ \textit{in sorted order}

Loop Invariant for Insertion-Sort (2)

Insertion-Sort($A$)
1 \textbf{for} $i = 2$ \textbf{to} length($A$) \\
2 \hspace{1em} $j = i$ \\
3 \hspace{1em} \textbf{while} $j > 1$ \textbf{and} $A[j - 1] > A[j]$ \\
4 \hspace{2em} \textbf{swap} $A[j]$ \textbf{and} $A[j - 1]$ \\
5 \hspace{2em} $j = j - 1$

- \textbf{Initialization:} $j = 2$, so $A[1..j - 1]$ is the single element $A[1]$
  - $A[1]$ is trivially sorted
Loop Invariant for Insertion-Sort (3)

Insertion-Sort(A)
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1

■ Maintenance: informally, if A[1 .. i − 1] is a permutation of the original A[1 .. i − 1] and A[1 .. i − 1] is sorted (invariant), then if we enter the inner loop:
  ▶ shifts the subarray A[k .. i − 1] by one position to the right
  ▶ inserts key, which was originally in A[i] at its proper position 1 ≤ k ≤ i − 1, in sorted order

Termination: Insertion-Sort terminates with i = length(A) + 1; the invariant states that
  ▶ A[1 .. i − 1] is a permutation of the original A[1 .. i − 1]
  ▶ A[1 .. i − 1] is sorted

Given the termination condition, A[1 .. i − 1] is the whole A
So Insertion-Sort is correct!
Summary

- You are given a problem $P$ and an algorithm $A$
  - $P$ formally defines a correctness condition
  - assume, for simplicity, that $A$ consists of one loop

1. Formulate an invariant $C$

2. **Initialization** (for all valid inputs)
   - prove that $C$ holds right before the first execution of the first instruction of the loop

3. **Management** (for all valid inputs)
   - prove that if $C$ holds right before the first instruction of the loop, then it holds also at the end of the loop

4. **Termination** (for all valid inputs)
   - prove that the loop terminates, with some exit condition $X$

5. Prove that $X \land C \Rightarrow P$, which means that $A$ is correct

Exercise: Analyze Selection-Sort

Selection-Sort($A$)
1  $n = \text{length}(A)$
2  for $i = 1$ to $n - 1$
3    $\text{smallest} = i$
4    for $j = i + 1$ to $n$
5      if $A[j] < A[\text{smallest}]$
6        $\text{smallest} = j$
7  swap $A[i]$ and $A[\text{smallest}]$

- Correctness?
  - loop invariant?

- Complexity?
  - worst, best, and average case?
Exercise: Analyze Bubblesort

Bubblesort(A)
1   for i = 1 to length(A)
2       for j = length(A) downto i + 1
4               swap A[j] and A[j - 1]

■ Correctness?
  ▶ loop invariant?

■ Complexity?
  ▶ worst, best, and average case?