Loop Invariants and Insertion-sort

COMS10018 - Algorithms

Dr Christian Konrad

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i, it is also true before iteration $i+1$

Example:
Computing the maximum

Invariant: Before iteration i:
$m = \max\{A[j] \ : \ 0 \le j < i\}$

```
Require: Array of n positive integers A
  m ← A[0]
  for i = 1, ..., n − 1 do
    if A[i] > m then
      m ← A[i]
  return m
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] \ : \ 0 \le j < 1\}$ ✓
- *Induction step.*
  Case $A[i] > m_i$**:** $m_{i+1} = A[i] > m_i = \max\{A[j] \ : \ 0 \le j < i\} \ \Rightarrow m_{i+1} = \max\{A[j] \ : \ 0 \le j < i+1\}$
  Case $A[i] \le m_i$**:** $m_{i+1} = m_i = \max\{A[j] \ : \ 0 \le j < i\} = \max\{A[j] \ : \ 0 \le j < i+1\}$ ✓

Loop Invariants - More Formally

Main Parts:

- **Initialization:** It is true prior to the first iteration of the loop.

  before iteration $i = 1 : m = A[0] = \max\{A[j] \ : \ j < 1\}$ ✓

- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.

  before iteration $i > 1 : m = \max\{A[j] \ : \ j < i\}$ ✓

- **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

  At the end of the loop, i.e., after iteration $n - 1$ (or before a virtual iteration n) $m = m_n = \max\{A[j] \ : \ j < n\}$ ✓

Example

Require: n integer
$s \leftarrow 1$
for $j = 2, \ldots, n$ **do**
$s \leftarrow s \cdot j$
return s

Invariant: At beginning of iteration j: $s = (j-1)!$

1. Let s_j be the value of s prior to iteration j
2. **Initialization:** $s_2 = 1 = (2-1)!$ ✓
3. **Maintenance:** $s_{j+1} = s_j \cdot j = (j-1)! \cdot j = j!$ ✓
4. **Termination:** After iteration n, i.e., before iteration $n+1$, the value of s is $s_{n+1} = (n+1-1)! = n!$ ✓

Algorithm computes the factorial function

Sorting Problem

- **Input:** An array A of n numbers
- **Output:** A reordering of A s.t. $A[0] \leq A[1] \leq \cdots \leq A[n-1]$

```
Require: Array A of n numbers
  for j = 1, ..., n − 1 do
    v ← A[j]
    i ← j − 1
    while i ≥ 0 and A[i] > v do
      A[i + 1] ← A[i]
      i ← i − 1
    A[i + 1] ← v
```

INSERTION-SORT

Example:

Require: Array A of n numbers
  for $j = 1, \ldots, n-1$ **do**
    $v \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i \geq 0$ **and** $A[i] > v$ **do**
      $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    $A[i+1] \leftarrow v$

0	1	2	3	4	$j = 5$
1	3	7	8	9	15

Loop Invariant of Insertion-sort

```
for j = 1, ..., n − 1 do
    v ← A[j]
    i ← j − 1
    while i ≥ 0 and A[i] > v do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← v
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j-1]$ consists of the elements originally in $A[0, j-1]$, but in sorted order

- **Initialization:** $j = 1$: subarray $A[0]$ is sorted ✓
- **Maintenance:** *Informally*, element $A[j]$ is inserted at the right place within $A[0, j]$. A formal argument would require another loop invariant for the inner loop. ✓
- **Termination:** After iteration $j = n - 1$ (i.e., before iteration $j = n$) the loop invariant states that A is sorted. ✓

Worst-case Runtime of Insertion-sort

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case
- All other operations take time $O(1)$. Hence:

$$\sum_{j=1}^{n-1} j \cdot O(1) = O(1) \sum_{j=1}^{n-1} j = O(1) \frac{n(n-1)}{2} = O(1)(n^2 - n) = O(n^2) .$$

Best-case Runtime: $O(n)$
E.g., if input is already sorted