Lecture 5: Loop Invariants and Insertion-sort

COMS10007 - Algorithms

Dr. Christian Konrad

11.01.2019

Structure of a Proof by Induction

1. **Statement to Prove:**
   $P(n)$ holds for all $n \in \mathbb{N}$
   (or $n \in \mathbb{N} \cup \{0\}$)
   (or n integer and $n \geq k$)
   (or similar)
2. **Induction hypothesis:**
   Assume that $P(n)$ holds
3. **Induction step:**
   Prove that $P(n+1)$ also holds
   If domino n falls then domino $n+1$ falls as well
4. **Base case:** Prove that $P(1)$ holds
   Domino 1 falls

Examples

Example: $n! \geq 2^n$ for $n \geq 4$

1. Base case ($n = 4$): $4! = 24 \geq 16 = 2^4$ ✓
2. Induction hypothesis: $n! \geq 2^n$ holds for n
3. Induction step:

$$(n+1)! = (n+1) \cdot n! \geq (n+1) \cdot 2^n \geq 2 \cdot 2^n = 2^{n+1} \checkmark$$

This also implies that $2^n \in O(n!)$

Example: $3^n - 1$ is an even number, for every $n \geq 1$

1. Base case ($n = 1$): $3^1 - 1 = 2$ ✓
2. Induction hypothesis: $3^n - 1$ is an even number
3. Induction step:

$$3^{n+1} - 1 = 3 \cdot 3^n - 1 = 3^n + 2 \cdot 3^n - 1 = 2 \cdot 3^n + 3^n - 1 \checkmark$$

Spot the Flaw

Example: $a^n = 1$, for every $a \neq 0$ and n nonnegative integer

1. Base case ($n = 0$): $a^0 = 1$
2. Induction hypothesis: $a^m = 1$, for every $0 \leq m \leq n$ (strong induction)
3. Induction step:

$$a^{n+1} = a^{2n-(n-1)} = \frac{a^{2n}}{a^{n-1}} = \frac{a^n \cdot a^n}{a^{n-1}} = \frac{1 \cdot 1}{1} = 1 \ldots .$$

Problem: a^1 is computed as $\frac{a^0 a^0}{a^{-1}}$ and induction hypothesis does not holds for $n = -1$

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
$m \leftarrow A[0]$
for $i = 1, \ldots, n-1$ **do**
  if $A[i] > m$ **then**
    $m \leftarrow A[i]$
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.*

$$\begin{aligned} m_{i+1} &= \max\{m_i, A[i]\} = \\ &= \max\{\max\{A[j] \ : \ 0 \le j < i\}, A[i]\} \\ &= \max\{A[j] \ : \ 0 \le j \le i\} \ . \checkmark \end{aligned}$$

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 m contains the maximum ✓

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 $(n+1-1)! = n!$ ✓

Algorithm computes the factorial function

Example: Insertion Sort

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	5
15	7	3	9	8	1

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

Example:

Require: Array A of n numbers
  for $j = 1, \dots, n-1$ **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	$j = 1$	2	3	4	5
15	7	3	9	8	1

$v \leftarrow 7$

Example:

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
```

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

$v \leftarrow 7$

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**
      $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    $A[i+1] \leftarrow v$

$i = -1$	$j = 1$	2	3	4	5
15	15	3	9	8	1

$v \leftarrow 7$

Example:

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

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

$v \leftarrow 7$

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

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

$v \leftarrow 3$

Example:

Require: Array A of n numbers

for $j = 1, \dots, n-1$ **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	$i = 1$	$j = 2$	3	4	5
7	15	3	9	8	1

$v \leftarrow 3$

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**
      $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    $A[i+1] \leftarrow v$

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

$v \leftarrow 3$

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**
      $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    $A[i+1] \leftarrow v$

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

$v \leftarrow 3$

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**
      $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    $A[i+1] \leftarrow v$

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

$v \leftarrow 3$

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**
      $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    $A[i+1] \leftarrow v$

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

$v \leftarrow 3$

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	$j = 3$	4	5
3	7	15	**9**	8	1

Example:

Require: Array A of n numbers
  for $j = 1, \dots, n-1$ **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	$j = 3$	4	5
3	7	**9**	15	8	1

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	$j = 4$	5
3	7	9	15	**8**	1

Example:

Require: Array A of n numbers
  for $j = 1, \dots, n-1$ **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	$j = 4$	5
3	7	**8**	9	15	1

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	$j = 5$
3	7	8	9	15	**1**

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 $j = 1, \dots, n-1$ **do**
  $v \leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i \geq 0$ **and** $A[i] > v$ **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  $A[i+1] \leftarrow v$

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:

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

Average-case Runtime of Insertion-sort

Property: Roughly half the elements left of $A[j]$ are smaller than $A[j]$ and roughly half are larger than $A[j]$

- Need to move $A[j]$ roughly to position $j/2$ (in the worst case, we move $A[j]$ to position 0, i.e., twice as far)
- Since
$$\sum_{j=1}^{n-1} \frac{j}{2}\Theta(1) = \Theta(n^2) \ ,$$
the average-case runtime is $\Theta(n^2)$.