CMPU241 Analysis of Algorithms

Day 4
Proofs of Algorithm Correctness
Using Loop Invariants

To prove an algorithm is correct, you need to know how the algorithm transforms input to correct output.

E.g., an algorithm to find the maximum value element in a set of totally ordered data is correct if its output is the largest number in the set.

What outcome is correct if you are running a sorting algorithm on a set of comparable data elements?

All the elements are in some specified ordering, commonly ascending order.

A loop invariant proof generally starts by showing that the initial condition or basis fits some criteria, and argues that consecutive iterations of the loop uphold these criteria. We will use proof by induction on the number of loop iterations to prove our claims.

Unlike most proofs by induction, algorithms terminate, resulting in the entire data set upholding the loop invariant to produce the correct result.

The parts of the proof are:

1. State the loop invariant.
2. Show that the invariant holds at the start of the 1st iteration.
3. Assume the invariant holds up to some iteration k > 1 (inductive hypothesis).
4. Show the actions that occur in iteration k lead to the invariant holding for the k+1st iteration (inductive step uses inductive hypothesis).
5. Argue that, at termination, the algorithm produces the desired result.

FindMax(A[1...n])

INPUT: An array A of *n* comparable items

OUTPUT: The value of the maximum item in the array

```
1. max = A[1]
2. for ( i = 2;  i <=n;  i++ )
3.     if (A[i] > max)
4.         max = A[i]
5. return max
```

$$\sum_{i=2}^{n} 1 = n - 2 + 1$$

Does this algorithm have variable running time on input arrays with different contents or orderings of size *n* ?

Give the line number of the basic operation.

Loop invariant for Find-Max:

Loop invariant: **At the start of each iteration k, max contains the largest value in A[1…k-1].**

Base case: **k = 2. Since max is set to equal A[1] before the first iteration, max holds the largest value in A[1…k-1] = A[1…1] = A[1]. So the base case holds.**

Inductive hypothesis: **Assume the invariant holds through the beginning of iteration k, where 2 < k < n, so max is the largest value in A[1…k-1].**

Inductive Step (Maintenance): **Show the invariant holds at the end of iteration k, the beginning of iteration k +1. Show that max is the largest value in subarray A[1…k] at the beginning of iteration k+1.**

By the inductive hypothesis, we know that max is the largest value in A[1…k-1] at the start of iteration k. In iteration k, the maximum element in A[1…k] is found by comparing max to the value in A[k]. Due to the total ordering on comparable items, max is either unchanged in this iteration because max >= A[k] or max is set to A[k] because max < A[k]. In either case, at the beginning of iteration k+1, max is the largest value in A[1…k].

Termination: **The for loop ends when k = n+1. At that point, max is the largest value in**

A[1…(n+1)-1] = A[1…n].

Therefore, at the end of the algorithm, the value of the maximum item in A[1…n] is returned and the algorithm is correct.

QED

Example: Sorting Problem

The algorithmic problem known as *sorting* is defined as follows:

INPUT: An array A[1…n] of *n* totally ordered elements $\{a_1, a_2, \ldots, a_n\}$
OUTPUT: A permutation of the input array $\{a_1', a_2', \ldots, a_n'\}$ such that $a_1' \le a_2' \le \ldots \le a_n'$.

Example instances of input for the sorting problem:
{Mike, Sally, Herbert, Tony, Jill}
{101, 111111, 1111, 100, 1010, 101010}

Example instances of output for the given instances of the sorting problem above:
{Herbert, Jill, Mike, Sally, Tony}
{100, 101, 1010, 1111, 101010, 111111}

Sorting Algorithm

InsertionSort(A)

INPUT: An array A of *n* items $\{a_1, a_2, \ldots, a_n\}$

OUTPUT: A permutation of the input array $\{a_1', a_2', \ldots, a_n'\}$ such that $a_1' \le a_2' \le \ldots \le a_n'$.

```
1.  for ( j  = 2 to length[A] )
2.      key = A[j]
3.      i = j – 1
4.      while ( i > 0 and A[i] > key)
5.          A[i + 1] = A[i]
6.          i = i - 1
7.      A[i + 1] =  key
```

Does InsertionSort have best- and worst-case run times?

Real-time Analysis of InsertionSort

InsertionSort(A)	times
1. for j = 2 to length[A]	n
2. key = A[j]	n-1
3. i = j - 1	n-1
4. while i > 0 and A[i] > key	$\sum_{j=2}^{n} t_j$
5. A[i + 1] = A[i]	$\sum_{j=2}^{n} (t_j - 1)$
6. i = i - 1	$\sum_{j=2}^{n} (t_j - 1)$
7. A[i + 1] = key	n-1

Analysis of InsertionSort

```
InsertionSort(A)
c1  for j  = 2 to length[A]
c2      key = A[j]
c3      i = j - 1
c4      while i > 0 and A[i] > key
c5          A[i + 1] = A[i]
c6          i = i - 1
c7      A[i + 1] =  key
```

- Give an instance of best-case and worst-case inputs.

What is the real running time for the best case?

$c_1n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$
$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$ = **an** - b

What is the real running time for the worst case?
Add up terms on last slide.

Proving Correctness—Insertion Sort

Loop invariant:
At the start of each iteration of the for loop of lines 1-7, the subarray A[1...j-1] consists of the elements originally in A[1...j-1], but in sorted order.

```
Insertion-Sort(A)
1. for j = 2 to A.length
2.   key = A[j]
3.   i = j - 1
4.   while i>0 and A[i]>key
5.       A[i+1] = A[i]
6.       i = i - 1
7.   A[i+1] = key
```

We need to show...

1. ...that the loop invariant is true at the start of the first iteration (base case or *initialization*),
2. ... the invariant remains true for the next k < n iterations (inductive hypothesis (IHOP) or *maintenance*), and
3. ...the algorithm has the correct result when the loop terminates.

Proving Correctness—Base Case

Loop invariant:
Let j be the position of the current key in the array A.
At the start of each iteration of the for loop, the subarray A[1...j-1] consists of the elements originally in A[1...j-1], but in sorted order.

```
Insertion-Sort(A)
1. for j = 2 to A.length
2.   key = A[j]
3.   i = j - 1
4.   while i>0 and A[i]>key
5.       A[i+1] = A[i]
6.       i = i - 1
7.   A[i+1] = key
```

Base case (initialization): When j = 2, A[1...j-1] has a single element and is therefore trivially sorted.

Proving Correctness—Inductive Hypothesis

Loop invariant:
Let j be the position of the current key in the array A.
At the start of each iteration of the for loop, the subarray A[1...j-1] consists of the elements originally in A[1...j-1], but in sorted order.

```
Insertion-Sort(A)
1. for j = 2 to A.length
2.   key = A[j]
3.   i = j - 1
4.   while i>0 and A[i]>key
5.       A[i+1] = A[i]
6.       i = i - 1
7.   A[i+1] = key
```

Inductive Hypothesis (IH): Assume the invariant holds through the beginning of the iteration where 2 < j = k < n.

Proving Correctness—Inductive Step

Loop invariant:
Let j be the position of the current key in the array A.
At the start of each iteration of the for loop, the subarray A[1...j-1] consists of the elements originally in A[1...j-1], but in sorted order.

```
Insertion-Sort(A)
1. for j = 2 to A.length
2.   key = A[j]
3.   i = j - 1
4.   while i>0 and A[i]>key
5.       A[i+1] = A[i]
6.       i = i - 1
7.   A[i+1] = key
```

Inductive Step (Maintenance): Show the invariant holds at the end of the iteration when j = k, and the beginning of the next iteration when j = k+1.

Proving Correctness—Inductive Step

Loop invariant:
Let j be the position of the current key in the array A.
At the start of each iteration of the for loop, the subarray A[1...j-1] consists of the elements originally in A[1...j-1], but in sorted order.

```
Insertion-Sort(A)
1. for j = 2 to A.length
2.   key = A[j]
3.   i = j - 1
4.   while i>0 and A[i]>key
5.       A[i+1] = A[i]
6.       i = i - 1
7.   A[i+1] = key
```

When j = k, *key* = A[k]. By the IHOP, we know that the subarray A[1...k-1] is in sorted order. In iteration k, each key to the left of A[k]: A[k-1], A[k-2], A[k-3]..., is moved one position to the right until either a value less than *key* is found or until k-1 values have been shifted right, when the value of *key* is inserted. Due to the total ordering on comparable data, *key* will be inserted in the correct position in the values A[1...k], so at the beginning of iteration k+1, the subarray A[1...k] will contain only the elements that were originally in A[1...k], but in sorted order. Therefore, the loop invariant holds at the start of iteration k+1.

Proving Correctness—Termination

Loop invariant:
Let j be the position of the current key in the array A. At the start of each iteration of the for loop, the subarray A[1…j-1] consists of the elements originally in A[1…j-1], but in sorted order.

```
Insertion-Sort(A)
1. for j = 2 to A.length
2.   key = A[j]
3.   i = j - 1
4.   while i>0 and A[i]>key
5.        A[i+1] = A[i]
6.        i = i - 1
7.   A[i+1] = key
```

Termination: The for loop ends when j = n+1. By the IHOP, we have that the subarray A[1…n] is in sorted order. Therefore, the entire array is sorted and the algorithm is correct. QED

Proving Correctness—BubbleSort

Loop invariant:
At the start of each iteration *i* of the for loop in lines 1-4, the subarray A[1…i –1] is in sorted order with the (i–1) smallest elements of A in positions 1…i–1.

BubbleSort(A) // A.length = n
(assume problem statement = that of InsertionSort)
1. for i = 1 to n – 1
2. for j = n downto i+1
3. if A[j] < A[j-1]
4. swap A[j] with A[j-1]

We need to show…
1. … the loop invariant is true at the start of the first iteration (basis or *initialization*),
2. … the invariant remains true for the next k <= n iterations (inductive hypothesis (IH) or *maintenance*), and
3. …the algorithm has the correct result when the loop terminates.

Proving Correctness—BubbleSort Base Case

Loop invariant:
At the start of each iteration *i* of the for loop in lines 1-4, the subarray A[1…i –1] is in sorted order with the (i–1) smallest elements of A in positions 1…i–1.

BubbleSort(A)
1. for i = 1 to n – 1
2. for j = n downto i+1
3. if A[j] < A[j-1]
4. swap A[j] with A[j-1]

Proving Correctness—BubbleSort IH

Loop invariant:
At the start of each iteration *i* of the for loop in lines 1-4, the subarray A[1…i –1] is in sorted order with the (i–1) smallest elements of A in positions 1…i–1.

BubbleSort(A)
1. for i = 1 to n – 1
2. for j = n downto i+1
3. if A[j] < A[j-1]
4. swap A[j] with A[j-1]

Proving Correctness—BubbleSort inductive step

Loop invariant:
At the start of each iteration *i* of the for loop in lines 1-4, the subarray A[1…i –1] is in sorted order with the (i–1) smallest elements of A in positions 1…i–1.

BubbleSort(A)
1. for i = 1 to n – 1
2. for j = n downto i+1
3. if A[j] < A[j-1]
4. swap A[j] with A[j-1]

Proving Correctness—BubbleSort Termination

Loop invariant:
At the start of each iteration *i* of the for loop in lines 1-4, the subarray A[1…i –1] is in sorted order with the (i–1) smallest elements of A in positions 1…i–1.

BubbleSort(A)
1. for i = 1 to n – 1
2. for j = n downto i+1
3. if A[j] < A[j-1]
4. swap A[j] with A[j-1]