CMPU241 Analysis of Algorithms

Proofs of Algorithm Correctness
Using Loop Invariants

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

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

All the elements are in some sorted order (increasing or decreasing).

A loop invariant generally refers to the actions inside a loop, 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 generally use proof by induction on the number of loop iterations. specifically on the loop counter variable.

The algorithms we study terminate, so we need to show that the ending value of the loop counter upholds the invariant and also why that means the algorithm is correct.

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

Need to know:

1. What value loop counter has at start of algorithm and show loop invariant holds at the start of the first iteration (basis).
2. Assume the loop invariant holds up to the beginning of some iteration k, where 1 < k < n (inductive hypothesis).
3. Show the invariant holds at the start of iteration k+1 (maintenance (ind step)).
4. Know what loop counter equals in the last iteration and show the invariant holds in that iteration and that the result is therefore correct (termination).

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

State a loop invariant for the FindMax algorithm.

Proving Correctness—Insertion Sort

```
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
```

Proving Correctness—BubbleSort

```
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 (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.