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.

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

FindMax(A[1...n])

INPUT: An array A of *n* comparable items OUTPUT: The value of the maximum item in the array

```
    max = A[1]
    for ( k = 2; k <=n; k++)</li>
    if (A[k] > max)
    max = A[k]
    return max
```

Loop invariant? Let k be the position of the current max in the array A. At the start of iteration k of the for loop, 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].

Inductive hypothesis: Assume the invariant holds through the beginning of the iteration where 1<= k < n, when 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 A[1...k].

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 or it is set to the value in 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]. Therefore at the end of the algorithm, the value of the maximum item in A[1...n] is returned and the algorithm is correct. OED

Proving Correctness—Insertion Sort

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

We need to show ...

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

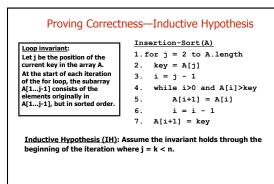
Let j be the position of the current key in the array A. 2. key = A[j] At the start of each iteration of the for loop, the subarray A[1...j-1] consists of the elements originally in A[1...j-1], but in sorted order. 3. i=j-1 4. while i>0 and A[i]>key 5. A[i+1] = A[i]6. i = i - 1 7. A[i+1] = key Base case (initialization): When j = 2, A[1...j-1] has a single element and is therefore trivially sorted.

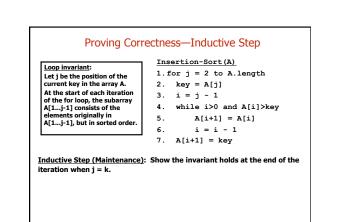
Proving Correctness—Base Case

Loop invariant:

Insertion-Sort(A)

1.for j = 2 to A.length





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.

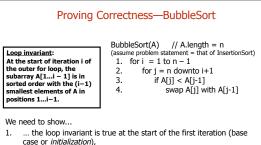
Insertion-Sort(A)		
1.fc	or $j = 2$ to A.length	
2.	key = A[j]	
з.	i = j - 1	
4.	while i>0 and A[i]>key	
5.	A[i+1] = A[i]	
6.	i = i - 1	
7.	A[i+1] = key	

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, A[k-1], A[k-2], A[k-3] and so on are each 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. In this iteration, key will be inserted in the right 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.

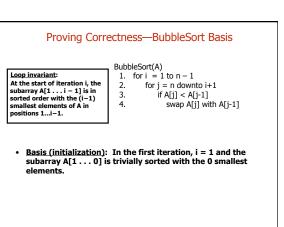
Proving Correctness—Termination

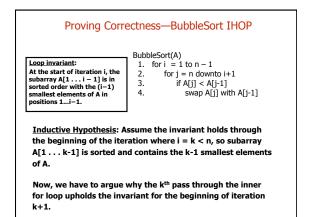
Insertion-Sort(A) 1.for j = 2 to A.length 2. key = A[j] 3. i = j - 1 4. while i>0 and A[i]>key 5. A[i+1] = A[i] 6. i = i - 1 7. A[i+1] = key

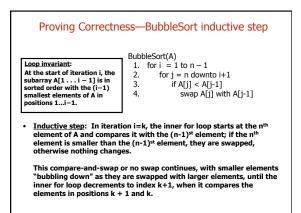
<u>Termination</u>: The for loop ends when j = n+1. By the IHOP, we have that the subarray A[1...n] is in sorted order. Therefore, the entire array is sorted and the algorithm is correct. QED

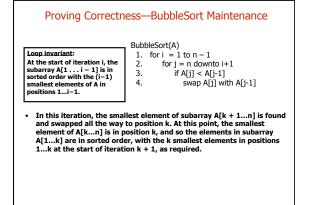


- case or *initialization*),
 ... the invariant remains true for the next k <= n iterations (inductive hypothesis (IHOP) or *maintenance*), and
- 3. ...the algorithm has the correct result when the loop terminates.









Proving Correctness—BubbleSort Termination

Loop invariant: At the start of iteration i, the subarray $A[1i-1]$ is in sorted order with the (i-1) smallest elements of A in positions 1i-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]
--	--

When the loop terminates, i = n. The invariant says that subarray A[1...n-1] is in sorted order and the n-1 smallest elements of A are in positions 1...n-1. So the element in position n must be the largest in A. Therefore, the entire array is sorted in ascending order and the algorithm is correct. QED