Review of loop invariants using Insertion Sort

Dr. AMIT KUMAR @JUET

Insertion Sort

- Problem: sort n numbers in $A[1..n]$.
- Input: n, numbers in A
- Output: A in sorted order: $\forall\ i \in [2..n],\ A[i\text{-}1] <= A[i]$

```
for j=2 to length(A)
    do key=A[j]
       i=j-1
       while i>0 and A[i]>key
         do A[i+1]=A[i]
            i--
       A[i+1]=key
```

Loop Invariant

- A **loop invariant** is a condition [among program variables] that is necessarily true immediately before and immediately after each iteration of a **loop**. (Note that this says nothing about its truth or falsity part way through an iteration.)

Loop Invariants

- **Invariants** – statements about an algorithm that remain valid
- We must show three things about loop invariants:
  - **Initialization** – statement is true before first iteration
  - **Maintenance** – *if* it is true before an iteration, *then* it remains true before the next iteration
  - **Termination** – when loop terminates the invariant gives a useful property to show the correctness of the algorithm

Example: Insertion Sort

- **Invariant**: *at the start of each* ***for*** *loop, A[1…j-1] consists of elements originally in A[1…j-1] but in sorted order*

```
for j=2 to length(A)
   do key=A[j]
      i=j-1
      while i>0 and A[i]>key
        do A[i+1]=A[i]
           i--
      A[i+1]=key
```

Example: Insertion Sort

- **Invariant**: *at the start of each **for** loop, A[1…j-1] consists of elements originally in A[1…j-1] but in sorted order*

```
for j=2 to length(A)
   do key=A[j]
      i=j-1
      while i>0 and A[i]>key
        do A[i+1]=A[i]
           i--
      A[i+1]:=key
```

- **Initialization**: *j = 2,* the invariant trivially holds because *A*[1] is a sorted array.

Example: Insertion Sort

- **Invariant**: *at the start of each **for** loop, A[1 . . . j-1] consists of elements originally in A[1 . . . j-1] but in sorted order*

```
for j=2 to length(A)
    do key=A[j]
       i=j-1
       while i>0 and A[i]>key
         do A[i+1]=A[i]
            i--
       A[i+1]:=key
```

- **Maintenance**: the inner **while** loop finds the position *i* with *A*[*i*] *<= key*, and shifts *A*[*j-1*], *A*[*j-2*], …, *A*[*i+1*] right by one position. Then *key*, formerly known as *A*[*j*], is placed in position *i+1* so that $A[i] \le A[i+1] < A[i+2]$.

$A[1...j-1]$ sorted + $A[j] \rightarrow A[1...j]$ sorted

Example: Insertion Sort

- **Invariant**: *at the start of each* ***for*** *loop, A[1...j-1] consists of elements originally in A[1...j-1] but in sorted order*

```
for j=2 to length(A)
    do key=A[j]
       i=j-1
       while i>0 and A[i]>key
         do A[i+1]=A[i]
            i--
       A[i+1]:=key
```

- **Termination**: the loop terminates, when *j=n+1*. Then the invariant states: *"A[1...n] consists of elements originally in A[1...n] but in sorted order."*