CSE 548: Analysis of Algorithms

Prerequisites Review 2
(Insertion Sort and Selection Sort)

Rezaul A. Chowdhury

Department of Computer Science
SUNY Stony Brook
Fall 2019

Insertion Sort

Input: An array $A[\,1 : \ n\,]$ of n numbers.

Output: Elements of $A[\,1 : \ n\,]$ rearranged in non-decreasing order of value.

```
INSERTION-SORT ( A )

for j = 2 to A.length
    key = A[j]
    // insert A[j] into the sorted sequence A[1..j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Loop Invariants

We use *loop invariants* to prove correctness of iterative algorithms

A loop invariant is associated with a given loop of an algorithm, and it is a formal statement about the relationship among variables of the algorithm such that

- **[Initialization]** It is true prior to the first iteration of the loop
- **[Maintenance]** If it is true before an iteration of the loop, it remains true before the next iteration
- **[Termination]** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct

Loop Invariants for Insertion Sort

```
INSERTION-SORT ( A )

for j = 2 to A.length
    key = A[j]
    // insert A[j] into the sorted sequence A[1..j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Loop Invariants for Insertion Sort

INSERTION-SORT (A)

for $j = 2$ **to** $A.length$

> **Invariant 1:** $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order

    $key = A[j]$

    // insert $A[j]$ into the sorted sequence $A[1..j-1]$

    $i = j - 1$

    while $i > 0$ *and* $A[i] > key$

        $A[i+1] = A[i]$

        $i = i - 1$

    $A[i+1] = key$

Loop Invariants for Insertion Sort

INSERTION-SORT (A)

1. ***for*** $j = 2$ ***to*** $A.length$

   Invariant 1: $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order

2. $key = A[j]$
3. // insert $A[j]$ into the sorted sequence $A[1..j-1]$
4. $i = j - 1$
5. ***while*** $i > 0$ ***and*** $A[i] > key$

   Invariant 2: $A[i..j]$ are each $\geq key$

6. $A[i+1] = A[i]$
7. $i = i - 1$
8. $A[i+1] = key$

Loop Invariant 1: Initialization

```
INSERTION-SORT ( A )

1.  for j = 2 to A.length
        Invariant 1: A[1..j − 1] consists of the elements
                     originally in A[1..j − 1], but in sorted order
2.      key = A[j]
3.      // insert A[j] into the sorted sequence A[1..j − 1]
4.      i = j − 1
5.      while i > 0 and A[i] > key
            Invariant 2: A[i..j] are each ≥ key
6.          A[i + 1] = A[i]
7.          i = i − 1
8.      A[i + 1] = key
```

At the start of the first iteration of the loop (in lines $1 - 8$): $j = 2$

Hence, subarray $A[1..j-1]$ consists of a single element $A[1]$, which is in fact the original element in $A[1]$.

The subarray consisting of a single element is trivially sorted.

Hence, the invariant holds initially.

Loop Invariant 1: Maintenance

```
INSERTION-SORT ( A )
for j = 2 to A.length
    Invariant 1: A[1..j − 1] consists of the elements
                 originally in A[1..j − 1], but in sorted order
    key = A[j]
    // insert A[j] into the sorted sequence A[1..j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        Invariant 2: A[i..j] are each ≥ key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

We assume that invariant 1 holds before the start of the current iteration.

Hence, the following holds: $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order.

For invariant 1 to hold before the start of the next iteration, the following must hold at the end of the current iteration:

$A[1..j]$ consists of the elements originally in $A[1..j]$, but in sorted order.

We use invariant 2 to prove this.

Loop Invariant 1: Maintenance
Loop Invariant 2: Initialization

INSERTION-SORT (A)

1. **for** $j = 2$ **to** $A.length$
   Invariant 1: $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order
2. $key = A[j]$
3. // insert $A[j]$ into the sorted sequence $A[1..j-1]$
4. $i = j - 1$
5. **while** $i > 0$ **and** $A[i] > key$
   Invariant 2: $A[i..j]$ are each $\geq key$
6. $A[i+1] = A[i]$
7. $i = i - 1$
8. $A[i+1] = key$

At the start of the first iteration of the loop (in lines $5-7$): $i = j - 1$

Hence, subarray $A[i..j]$ consists of only two entries: $A[i]$ and $A[j]$.

We know the following:

— $A[i] > key$ (explicitly tested in line 5)
— $A[j] = key$ (from line 2)

Hence, invariant 2 holds initially.

Loop Invariant 1: Maintenance
Loop Invariant 2: Maintenance

```
INSERTION-SORT ( A )

1.  for j = 2 to A.length
       Invariant 1: A[1..j − 1] consists of the elements
                    originally in A[1..j − 1], but in sorted order
2.     key = A[j]
3.     // insert A[j] into the sorted sequence A[1..j − 1]
4.     i = j − 1
5.     while i > 0 and A[i] > key
          Invariant 2: A[i..j] are each ≥ key
6.        A[i + 1] = A[i]
7.        i = i − 1
8.     A[i + 1] = key
```

We assume that invariant 2 holds before the start of the current iteration.

Hence, the following holds: $A[i..j]$ are each $\geq key$.

Since line 6 copies $A[i]$ which is known to be $> key$ to $A[i+1]$ which also held a value $\geq key$, the following holds at the end of the current iteration: $A[i+1..j]$ are each $\geq key$.

Before the start of the next iteration the check $A[i] > key$ in line 5 ensures that invariant 2 continues to hold.

Loop Invariant 1: Maintenance
Loop Invariant 2: Maintenance

```
INSERTION-SORT ( A )

1.  for j = 2 to A.length
        Invariant 1: A[1..j − 1] consists of the elements
                     originally in A[1..j − 1], but in sorted order
2.      key = A[j]
3.      // insert A[j] into the sorted sequence A[1..j − 1]
4.      i = j − 1
5.      while i > 0 and A[i] > key
            Invariant 2: A[i..j] are each ≥ key
6.          A[i + 1] = A[i]
7.          i = i − 1
8.      A[i + 1] = key
```

Observe that the inner loop (in lines $5 - 7$) does not destroy any data because though the first iteration overwrites $A[j]$, that $A[j]$ has already been saved in key in line 2.

As long as key is copied back into a location in $A[1..j]$ without destroying any other element in that subarray, we maintain the invariant that $A[1..j]$ contains the first j elements of the original list.

Loop Invariant 1: Maintenance
Loop Invariant 2: Termination

```
INSERTION-SORT ( A )
  for j = 2 to A.length
      Invariant 1: A[1..j − 1] consists of the elements
                   originally in A[1..j − 1], but in sorted order
      key = A[j]
      // insert A[j] into the sorted sequence A[1..j − 1]
      i = j − 1
      while i > 0 and A[i] > key
          Invariant 2: A[i..j] are each ≥ key
          A[i + 1] = A[i]
          i = i − 1
      A[i + 1] = key
```

When the inner loop terminates we know the following.

– $A[1..i]$ is sorted with each element $\leq key$
  - if $i = 0$, true by default
  - if $i > 0$, true because $A[1..i]$ is sorted and $A[i] \leq key$

– $A[i+1..j]$ is sorted with each element $\geq key$ because the following held before i was decremented: $A[i..j]$ is sorted with each item $\geq key$

– $A[i+1] = A[i+2]$ if the loop was executed at least once, and $A[i+1] = key$ otherwise

Loop Invariant 1: Maintenance
Loop Invariant 2: Termination

```
INSERTION-SORT ( A )
1.  for j = 2 to A.length
        Invariant 1: A[1..j − 1] consists of the elements
                     originally in A[1..j − 1], but in sorted order
2.      key = A[j]
3.      // insert A[j] into the sorted sequence A[1..j − 1]
4.      i = j − 1
5.      while i > 0 and A[i] > key
            Invariant 2: A[i..j] are each ≥ key
6.          A[i + 1] = A[i]
7.          i = i − 1
8.      A[i + 1] = key
```

When the inner loop terminates we know the following.

— $A[1..i]$ is sorted with each element $\leq key$

— $A[i+1..j]$ is sorted with each element $\geq key$

— $A[i+1] = A[i+2]$ or $A[i+1] = key$

Given the facts above, line 8 does not destroy any data, and gives us $A[1..j]$ as the sorted permutation of the original data in $A[1..j]$.

Loop Invariant 1: Termination

```
INSERTION-SORT ( A )
for j = 2 to A.length
    Invariant 1: A[1..j − 1] consists of the elements
                 originally in A[1..j − 1], but in sorted order
    key = A[j]
    // insert A[j] into the sorted sequence A[1..j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        Invariant 2: A[i..j] are each ≥ key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

When the outer loop terminates we know that $j = A.length + 1$.

Hence, $A[1..j-1]$ is the entire array $A[1..A.length]$, which is sorted and contains the original elements of $A[1..A.length]$.

Worst Case Runtime of Insertion Sort (Upper Bound)

INSERTION-SORT (A)

	cost	times
for $j = 2$ ***to*** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// insert $A[j]$ into the sorted sequence $A[1..j-1]$	0	$n-1$
$i = j - 1$	c_4	$n-1$
while $i > 0$ ***and*** $A[i] > key$	c_5	$\sum_{2 \le j \le n} j$
$A[i+1] = A[i]$	c_6	$\sum_{2 \le j \le n} (j-1)$
$i = i - 1$	c_7	$\sum_{2 \le j \le n} (j-1)$
$A[i+1] = key$	c_8	$n-1$

Running time, $T(n) \le c_1 n + c_2(n-1) + c_4(n-1)$

$$+c_5 \sum_{j=2}^{n} j + c_6 \sum_{j=2}^{n} (j-1) + c_7 \sum_{j=2}^{n} (j-1) + c_8(n-1)$$

$$= 0.5(c_5 + c_6 + c_7)n^2 + 0.5(2c_1 + 2c_2 + 2c_4 + c_5 - c_6 - c_7 + 2c_8)n$$
$$-(c_2 + c_4 + c_5 + c_8)$$

$$\Rightarrow T(n) = O\left(n^2\right)$$

Best Case Runtime of Insertion Sort (Lower Bound)

INSERTION-SORT (A)	cost	times
for $j = 2$ ***to*** $A.length$	c_1	n
$key = A[j]$	c_2	
// insert $A[j]$ into the sorted sequence $A[1..j-1]$	0	
$i = j - 1$	c_4	$n-1$
while $i > 0$ ***and*** $A[i] > key$	c_5	
$A[i+1] = A[i]$	c_6	0
$i = i - 1$	c_7	
$A[i+1] = key$	c_8	$n-1$

Running time, $T(n) \geq c_1 n + c_2(n-1) + c_4(n-1)$
$+ c_5(n-1) + c_8(n-1)$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$$\Rightarrow T(n) = \Omega(n)$$

Selection Sort

Input: An array $A[\,1 : n\,]$ of n numbers.

Output: Elements of $A[\,1 : n\,]$ rearranged in non-decreasing order of value.

```
SELECTION-SORT ( A )

for j = 1 to A.length
    // find the index of an entry with the smallest value in A[j..A.length]
    min = j
    for i = j + 1 to A.length
        if A[i] < A[min]
            min = i
    // swap A[j] and A[min]
    A[j] ↔ A[min]
```