

Loop Invariants and Insertion-sort

COMS10017 - (Object-Oriented Programming and) Algorithms

Dr Christian Konrad

Loop Invariants

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

```
Require: Array of  $n$  positive integers  $A$   
 $m \leftarrow A[0]$   
for  $i = 1, \dots, n - 1$  do  
  if  $A[i] > m$  then  
     $m \leftarrow A[i]$   
return  $m$ 
```

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$   
 $m \leftarrow A[0]$   
for  $i = 1, \dots, n - 1$  do  
    if  $A[i] > m$  then  
         $m \leftarrow A[i]$   
return  $m$ 
```

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

Proof:

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$   
 $m \leftarrow A[0]$   
for  $i = 1, \dots, n - 1$  do  
    if  $A[i] > m$  then  
         $m \leftarrow A[i]$   
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.*

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$:

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓
- *Induction step.*

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓
- *Induction step.*
Case $A[i] > m_i$:

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓
- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} =$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓
- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i]$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓
- *Induction step.*
Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓
- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\} \Rightarrow m_{i+1}$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓
- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\}$
 $\Rightarrow m_{i+1} = \max\{A[j] : 0 \leq j < i + 1\}$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓

- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\}$
 $\Rightarrow m_{i+1} = \max\{A[j] : 0 \leq j < i + 1\}$

Case $A[i] \leq m_i$:

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓

- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\}$
 $\Rightarrow m_{i+1} = \max\{A[j] : 0 \leq j < i + 1\}$

Case $A[i] \leq m_i$: $m_{i+1} =$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓

- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\}$
 $\Rightarrow m_{i+1} = \max\{A[j] : 0 \leq j < i + 1\}$

Case $A[i] \leq m_i$: $m_{i+1} = m_i =$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓

- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\}$
 $\Rightarrow m_{i+1} = \max\{A[j] : 0 \leq j < i + 1\}$

Case $A[i] \leq m_i$: $m_{i+1} = m_i = \max\{A[j] : 0 \leq j < i\}$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓

- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\}$
 $\Rightarrow m_{i+1} = \max\{A[j] : 0 \leq j < i + 1\}$

Case $A[i] \leq m_i$: $m_{i+1} = m_i = \max\{A[j] : 0 \leq j < i\} =$
 $\max\{A[j] : 0 \leq j < i + 1\}$

Loop Invariants

Definition: A *loop invariant* is a property P that, if true before iteration i , it is also true before iteration $i + 1$

Example:

Computing the maximum

Invariant: Before iteration i :
 $m = \max\{A[j] : 0 \leq j < i\}$

```
Require: Array of  $n$  positive integers  $A$ 
 $m \leftarrow A[0]$ 
for  $i = 1, \dots, n - 1$  do
  if  $A[i] > m$  then
     $m \leftarrow A[i]$ 
return  $m$ 
```

Proof: Let m_i be the value of m before iter. i ($\rightarrow m_1 = A[0]$).

- *Base case.* $i = 1$: $m_1 = A[0] = \max\{A[j] : 0 \leq j < 1\}$ ✓

- *Induction step.*

Case $A[i] > m_i$: $m_{i+1} = A[i] > m_i = \max\{A[j] : 0 \leq j < i\}$
 $\Rightarrow m_{i+1} = \max\{A[j] : 0 \leq j < i + 1\}$

Case $A[i] \leq m_i$: $m_{i+1} = m_i = \max\{A[j] : 0 \leq j < i\} =$
 $\max\{A[j] : 0 \leq j < i + 1\}$ ✓

Main Parts:

Main Parts:

- **Initialization:** It is true prior to the first iteration of the loop.

Main Parts:

- **Initialization:** It is true prior to the first iteration of the loop.
before iteration $i = 1 : m = A[0] = \max\{A[j] : j < 1\} \checkmark$

Main Parts:

- **Initialization:** It is true prior to the first iteration of the loop.
before iteration $i = 1 : m = A[0] = \max\{A[j] : j < 1\} \checkmark$
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.

Main Parts:

- **Initialization:** It is true prior to the first iteration of the loop.
before iteration $i = 1 : m = A[0] = \max\{A[j] : j < 1\} \checkmark$
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.
before iteration $i > 1 : m = \max\{A[j] : j < i\} \checkmark$

Main Parts:

- **Initialization:** It is true prior to the first iteration of the loop.
before iteration $i = 1 : m = A[0] = \max\{A[j] : j < 1\} \checkmark$
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.
before iteration $i > 1 : m = \max\{A[j] : j < i\} \checkmark$
- **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

Main Parts:

- **Initialization:** It is true prior to the first iteration of the loop.

before iteration $i = 1 : m = A[0] = \max\{A[j] : j < 1\} \checkmark$

- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.

before iteration $i > 1 : m = \max\{A[j] : j < i\} \checkmark$

- **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

At the end of the loop, i.e., after iteration $n - 1$ (or before a virtual iteration n) $m = m_n = \max\{A[j] : j < n\} \checkmark$

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```


Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** s_{j+1}

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** $s_{j+1} = s_j \cdot j$

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** $s_{j+1} = s_j \cdot j = (j - 1)! \cdot j$

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** $s_{j+1} = s_j \cdot j = (j - 1)! \cdot j = j! \checkmark$

Example

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** $s_{j+1} = s_j \cdot j = (j - 1)! \cdot j = j! \checkmark$
- 4 **Termination:**


```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** $s_{j+1} = s_j \cdot j = (j - 1)! \cdot j = j! \checkmark$
- 4 **Termination:** After iteration n , i.e., before iteration $n + 1$, the value of s is $s_{n+1} = (n + 1 - 1)! = n!$

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** $s_{j+1} = s_j \cdot j = (j - 1)! \cdot j = j! \checkmark$
- 4 **Termination:** After iteration n , i.e., before iteration $n + 1$, the value of s is $s_{n+1} = (n + 1 - 1)! = n! \checkmark$

```
Require:  $n$  integer  
 $s \leftarrow 1$   
for  $j = 2, \dots, n$  do  
     $s \leftarrow s \cdot j$   
return  $s$ 
```

Invariant: At beginning of iteration j : $s = (j - 1)!$

- 1 Let s_j be the value of s prior to iteration j
- 2 **Initialization:** $s_2 = 1 = (2 - 1)! \checkmark$
- 3 **Maintenance:** $s_{j+1} = s_j \cdot j = (j - 1)! \cdot j = j! \checkmark$
- 4 **Termination:** After iteration n , i.e., before iteration $n + 1$, the value of s is $s_{n+1} = (n + 1 - 1)! = n! \checkmark$

Algorithm computes the factorial function

Sorting Problem

- **Input:** An array A of n numbers
- **Output:** A reordering of A s.t. $A[0] \leq A[1] \leq \dots \leq A[n-1]$

Example: Insertion Sort

Sorting Problem

- **Input:** An array A of n numbers
- **Output:** A reordering of A s.t. $A[0] \leq A[1] \leq \dots \leq A[n-1]$

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
     $v \leftarrow A[j]$   
     $i \leftarrow j - 1$   
    while  $i \geq 0$  and  $A[i] > v$  do  
         $A[i + 1] \leftarrow A[i]$   
         $i \leftarrow i - 1$   
     $A[i + 1] \leftarrow v$ 
```

INSERTION-SORT

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	2	3	4	5
15	7	3	9	8	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	$j = 1$	2	3	4	5
15	7	3	9	8	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	$j = 1$	2	3	4	5
15	7	3	9	8	1

$v \leftarrow 7$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

$i = 0$	$j = 1$	2	3	4	5
15	7	3	9	8	1

$v \leftarrow 7$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

$i = -1$	$j = 1$	2	3	4	5
15	15	3	9	8	1

$v \leftarrow 7$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

$i = -1$	$j = 1$	2	3	4	5
7	15	3	9	8	1

$v \leftarrow 7$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	$j = 2$	3	4	5
7	15	3	9	8	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	$j = 2$	3	4	5
7	15	3	9	8	1

$v \leftarrow 3$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	$i = 1$	$j = 2$	3	4	5
7	15	3	9	8	1

$v \leftarrow 3$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

$i = 0$	1	$j = 2$	3	4	5
7	15	15	9	8	1

$v \leftarrow 3$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

$i = -1$	1	$j = 2$	3	4	5
7	7	15	9	8	1

$v \leftarrow 3$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

$i = -1$	1	$j = 2$	3	4	5
7	7	15	9	8	1

$v \leftarrow 3$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

$i = -1$	1	$j = 2$	3	4	5
3	7	15	9	8	1

$v \leftarrow 3$

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	2	$j = 3$	4	5
3	7	15	9	8	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	2	$j = 3$	4	5
3	7	9	15	8	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	2	3	$j = 4$	5
3	7	9	15	8	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	2	3	$j = 4$	5
3	7	8	9	15	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	2	3	4	$j = 5$
3	7	8	9	15	1

Example:

```
Require: Array  $A$  of  $n$  numbers  
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

0	1	2	3	4	$j = 5$
1	3	7	8	9	15

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do  
   $v \leftarrow A[j]$   
   $i \leftarrow j - 1$   
  while  $i \geq 0$  and  $A[i] > v$  do  
     $A[i + 1] \leftarrow A[i]$   
     $i \leftarrow i - 1$   
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do
   $v \leftarrow A[j]$ 
   $i \leftarrow j - 1$ 
  while  $i \geq 0$  and  $A[i] > v$  do
     $A[i + 1] \leftarrow A[i]$ 
     $i \leftarrow i - 1$ 
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j - 1]$ consists of the elements originally in $A[0, j - 1]$, but in sorted order

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do
   $v \leftarrow A[j]$ 
   $i \leftarrow j - 1$ 
  while  $i \geq 0$  and  $A[i] > v$  do
     $A[i + 1] \leftarrow A[i]$ 
     $i \leftarrow i - 1$ 
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j - 1]$ consists of the elements originally in $A[0, j - 1]$, but in sorted order

- **Initialization:** $j = 1$:

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do
   $v \leftarrow A[j]$ 
   $i \leftarrow j - 1$ 
  while  $i \geq 0$  and  $A[i] > v$  do
     $A[i + 1] \leftarrow A[i]$ 
     $i \leftarrow i - 1$ 
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j - 1]$ consists of the elements originally in $A[0, j - 1]$, but in sorted order

- **Initialization:** $j = 1$: subarray $A[0]$ is sorted ✓

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do
   $v \leftarrow A[j]$ 
   $i \leftarrow j - 1$ 
  while  $i \geq 0$  and  $A[i] > v$  do
     $A[i + 1] \leftarrow A[i]$ 
     $i \leftarrow i - 1$ 
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j - 1]$ consists of the elements originally in $A[0, j - 1]$, but in sorted order

- **Initialization:** $j = 1$: subarray $A[0]$ is sorted ✓
- **Maintenance:**

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do
   $v \leftarrow A[j]$ 
   $i \leftarrow j - 1$ 
  while  $i \geq 0$  and  $A[i] > v$  do
     $A[i + 1] \leftarrow A[i]$ 
     $i \leftarrow i - 1$ 
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j - 1]$ consists of the elements originally in $A[0, j - 1]$, but in sorted order

- **Initialization:** $j = 1$: subarray $A[0]$ is sorted ✓
- **Maintenance:** *Informally*, element $A[j]$ is inserted at the right place within $A[0, j]$. A formal argument would require another loop invariant for the inner loop. ✓

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do
   $v \leftarrow A[j]$ 
   $i \leftarrow j - 1$ 
  while  $i \geq 0$  and  $A[i] > v$  do
     $A[i + 1] \leftarrow A[i]$ 
     $i \leftarrow i - 1$ 
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j - 1]$ consists of the elements originally in $A[0, j - 1]$, but in sorted order

- **Initialization:** $j = 1$: subarray $A[0]$ is sorted ✓
- **Maintenance:** *Informally*, element $A[j]$ is inserted at the right place within $A[0, j]$. A formal argument would require another loop invariant for the inner loop. ✓
- **Termination:**

Loop Invariant of Insertion-sort

```
for  $j = 1, \dots, n - 1$  do
   $v \leftarrow A[j]$ 
   $i \leftarrow j - 1$ 
  while  $i \geq 0$  and  $A[i] > v$  do
     $A[i + 1] \leftarrow A[i]$ 
     $i \leftarrow i - 1$ 
   $A[i + 1] \leftarrow v$ 
```

Loop Invariant: At beginning of iteration j of the outer **for** loop, the subarray $A[0, j - 1]$ consists of the elements originally in $A[0, j - 1]$, but in sorted order

- **Initialization:** $j = 1$: subarray $A[0]$ is sorted ✓
- **Maintenance:** *Informally*, element $A[j]$ is inserted at the right place within $A[0, j]$. A formal argument would require another loop invariant for the inner loop. ✓
- **Termination:** After iteration $j = n - 1$ (i.e., before iteration $j = n$) the loop invariant states that A is sorted. ✓

Worst-case Runtime of Insertion-sort

Worst-case Runtime:

Worst-case Runtime:

- We have two nested loops

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case
- All other operations take time $O(1)$. Hence:

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case
- All other operations take time $O(1)$. Hence:

$$\sum_{j=1}^{n-1} j \cdot O(1)$$

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case
- All other operations take time $O(1)$. Hence:

$$\sum_{j=1}^{n-1} j \cdot O(1) = O(1) \sum_{j=1}^{n-1} j$$

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case
- All other operations take time $O(1)$. Hence:

$$\sum_{j=1}^{n-1} j \cdot O(1) = O(1) \sum_{j=1}^{n-1} j = O(1) \frac{n(n-1)}{2}$$

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case
- All other operations take time $O(1)$. Hence:

$$\sum_{j=1}^{n-1} j \cdot O(1) = O(1) \sum_{j=1}^{n-1} j = O(1) \frac{n(n-1)}{2} = O(1)(n^2 - n) = O(n^2).$$

Best-case Runtime:

Worst-case Runtime:

- We have two nested loops
- The outer loop goes from $j = 1$ to $j = n - 1$
- The inner loop goes from $i = j - 1$ down to $i = 0$ in worst case
- All other operations take time $O(1)$. Hence:

$$\sum_{j=1}^{n-1} j \cdot O(1) = O(1) \sum_{j=1}^{n-1} j = O(1) \frac{n(n-1)}{2} = O(1)(n^2 - n) = O(n^2).$$

Best-case Runtime: $O(n)$

E.g., if input is already sorted