Unit 04: Proofs

Anthony Estey

CSC 225: Algorithms and Data Structures I

University of Victoria

*Thanks to Dr. Rich Little at the University of Victoria for providing some of the contents of these slides

Unit 04 Overview

- Supplemental Reading:
  - Algorithm Design and Analysis. *Michael Goodrich and Roberto Tamassia*
    - Pages 10, 19-25
- Learning Objectives: (You should be able to…)
  - write proofs using the following proof techniques: *proof by counterexample*, *direct proofs*, *proof by contrapositive*, *and proof by contradiction*
  - understand the steps necessary for and how to write a proof by induction
  - understand the steps necessary for and be able to write a loop invariant proof
  - determine the loop invariant by examining the pseudocode for an algorithm

Introduction

- Why is there a proofs unit in this course?

- As we explore different algorithms and data structures, we will make strong claims about the correctness or speed of the algorithm

- This unit overviews several ways we will justify, or ***prove***, these claims

Techniques covered

- In this unit, we will first review some common proof techniques:
  - Counterexample
  - Direct proof
  - Contrapositive
  - Contradiction
  - Induction

- And also introduce a new technique you may not have seen before:
  - Loop Invariant

Counterexample

- To prove something is true, in general, you must do so for all possible values
- To prove something is false, a single example must be presented illustrating that it doesn't work (is false).

- Worksheet example:

$$a^2 + b^2 = (a + b)^2$$

Direct Proof

- To prove $P \Rightarrow Q$ directly, we consider an element x for which $P(x)$ is true and show $Q(x)$ is also true.

- Worksheet example:

  If n is an odd integer then $3n + 7$ is an even integer.

Proof by Contrapositive

- The *contrapositive* of $P \Rightarrow Q$ is the implication $\neg Q \Rightarrow \neg P$.
- A proof by contrapositive of $P \Rightarrow Q$ is a direct proof of $\neg Q \Rightarrow \neg P$.
- Another way of putting it: the contrapositive of “if A, then B” is “if not B, then not A.”

- Worksheet examples:
  a) Let n be an integer. If $5n - 7$ is even, then n is odd.
  b) Let A and B be sets. If $A \cup B = A$, then $B \subseteq A$

Proof by contradiction

- To show that $P \Rightarrow Q$ is true by contradiction we show that $\neg(P \Rightarrow Q) \Rightarrow \perp$ (a contradiction).
- Since $\neg(P \Rightarrow Q)$ is logically equivalent to $(P \wedge \neg Q)$, we want to show that $(P \wedge \neg Q) \Rightarrow \perp$ (a contradiction).

- Worksheet example:

$\sqrt{2}$ is irrational

Induction

- Formal description from the textbook:
  - Let $S_1, S_2, S_3 \dots$ be statements such that:
    - *i.* S_1 is true; and
    - ii. Whenever S_k is true, where $k \in \mathbb{N}$, then S_{k+1} is true
  - Then all of the statements $S_1, S_2, S_3 \dots$ are true.

- In general, a proof by induction consists of two cases:
  1. **Base case**: prove the statement holds for $n = 0$
  2. **Inductive Step:** prove that ***if*** the statement holds for any given case $n = k$ (called the *inductive hypothesis*), ***then*** it must also hold for the next case, $n = k + 1$.

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1 + 3 + 5 + \cdots + (2n - 1) = n^2$)

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1 + 3 + 5 + \cdots + (2n - 1) = n^2$)

1. **Base case:**
   - When $n = 1$, the statement is $1 = 1^2$, which is **true**

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1 + 3 + 5 + \cdots + (2n - 1) = n^2$)

2. **Inductive Step:**
   - *Inductive hypothesis*: Assume the statement holds for $n = k$, for any $k \geq 1$.
     So, here we assume $1 + 3 + 5 + \cdots + (2k - 1) = k^2$
   - Show $n = k + 1$ also holds. That is: $1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1 + 3 + 5 + \cdots + (2n - 1) = n^2$)

2. **Inductive Step:**
   - *Inductive hypothesis*: Assume the statement holds for $n = k$, for any $k \geq 1$.
     So, here we assume $1 + 3 + 5 + \cdots + (2k - 1) = k^2$
   - Show $n = k + 1$ also holds. That is: $1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$

$$1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$$

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1+3+5+\cdots+(2n-1)=n^2$)

2. **Inductive Step:**
   - *Inductive hypothesis*: Assume the statement holds for $n=k$, for any $k \geq 1$.
     So, here we assume $1+3+5+\cdots+(2k-1)=k^2$
   - Show $n=k+1$ also holds. That is: $1+3+5+\cdots+(2(k+1)-1)=(k+1)^2$

$$1+3+5+\cdots+(2(k+1)-1)=(k+1)^2$$

$$1+3+5+\cdots(2(k)-1)+(2(k+1)-1)=$$

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1+3+5+\cdots+(2n-1)=n^2$)

2. **Inductive Step:**
   - *Inductive hypothesis*: Assume the statement holds for $n=k$, for any $k \geq 1$.
     So, here we assume $1+3+5+\cdots+(2k-1)=k^2$
   - Show $n=k+1$ also holds. That is: $1+3+5+\cdots+(2(k+1)-1)=(k+1)^2$

$$1+3+5+\cdots+(2(k+1)-1)=(k+1)^2$$

$$1+3+5+\cdots(2(k)-1)+(2(k+1)-1)=$$

$$k^2+(2(k+1)-1)=$$

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1 + 3 + 5 + \cdots + (2n - 1) = n^2$)

2. **Inductive Step:**
   - *Inductive hypothesis*: Assume the statement holds for $n = k$, for any $k \geq 1$.
     So, here we assume $1 + 3 + 5 + \cdots + (2k - 1) = k^2$
   - Show $n = k + 1$ also holds. That is: $1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$

$$1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$$

$$1 + 3 + 5 + \cdots (2(k) - 1) + (2(k + 1) - 1) =$$

$$k^2 + (2(k + 1) - 1) =$$

$$k^2 + 2k + 1 =$$

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1 + 3 + 5 + \cdots + (2n - 1) = n^2$)

2. **Inductive Step:**
   - *Inductive hypothesis*: Assume the statement holds for $n = k$, for any $k \geq 1$.
     So, here we assume $1 + 3 + 5 + \cdots + (2k - 1) = k^2$
   - Show $n = k + 1$ also holds. That is: $1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$

$$1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$$

$$1 + 3 + 5 + \cdots (2(k) - 1) + (2(k + 1) - 1) =$$

$$k^2 + (2(k + 1) - 1) =$$

$$k^2 + 2k + 1 = (k + 1)^2$$

Induction example

- Prove that the sum of the first n odd integers is equal to n^2.
  (The sum of integers $1 + 3 + 5 + \cdots + (2n - 1) = n^2$)

2. **Inductive Step:**
   - *Inductive hypothesis*: Assume the statement holds for $n = k$, for any $k \geq 1$.
     So, here we assume $1 + 3 + 5 + \cdots + (2k - 1) = k^2$
   - Show $n = k + 1$ also holds. That is: $1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$

$$1 + 3 + 5 + \cdots + (2(k + 1) - 1) = (k + 1)^2$$

$$1 + 3 + 5 + \cdots (2(k) - 1) + (2(k + 1) - 1) =$$

$$k^2 + (2(k + 1) - 1) =$$

$$k^2 + 2k + 1 = (k + 1)^2$$

   - It follows by induction that $1 + 3 + 5 + \cdots + (2n - 1) = n^2$.

Recap: Proof by induction

- In general, a proof by induction consists of:

1. Base case:
    - Prove the statement for $n = 0$ (or whatever the base case is)
    - This is typically easy – substitute 0 into n for both sides of the equation and show that they are equal
2. Induction Step
    i. Inductive hypothesis: Assume that the statement holds for $n = k$, where k is any positive integer
    ii. Given the assumption from the inductive hypothesis, show the statement also holds for $n = k + 1$.

    (Typically involves substitution from the assumption made in the I.H.)

Loop Invariant

- What is a loop invariant?
- An **invariant** is a ***property*** that is always true at particular points in a program
- A **loop invariant** is a ***property*** that is true before (and after) each iteration of a loop
  - We want to prove it is true before the first entering the loop
  - We want to prove it *holds* true for all iterations
- One way of thinking of a loop is that it starts with a true invariant and does work to keep the invariant true for the next iteration of the loop

Loop Invariant

- What is a loop invariant?
- An **invariant** is a ***property*** that is always true at particular points in a program
- A **loop invariant** is a ***property*** that is true before (and after) each iteration of a loop

Loop Invariant Proof

- Formal description:
  - To prove a statement S is correct, define S in terms of smaller statements $S_1, S_2, S_3 \dots S_n$ where
    1. *i.* S_1 is true before the loop
    2. *ii.* S_k is true before iteration k, where $1 \leq k \leq n$, and based on this assumption we then must show that S_{k+1} is true after iteration k
    3. iii. Thus, S_n implies S is true by induction
- In general, a loop invariant consists of the following cases:
  1. **Base case (initialization)**: prove the invariant holds before the loop starts
  2. **Inductive Step (maintenance):** prove that ***if*** the invariant holds right before beginning iteration *k* (called the *inductive hypothesis*), ***then*** it must also hold at the end of that iteration (before beginning the next iteration)
  3. ***Termination**: make sure the loop will eventually end!

Loop Invariant Example

- What is the loop invariant?
  - What property is true before we enter the loop, that we want to stay true throughout each iteration, and remain true once the loop has terminated?

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

- Since the algorithm finds the max value, let's try:

$\boxed{currentMax \text{ holds the maximum value found in the first } k \text{ elements of the array}}$

Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

Algorithm arrayMax(A, n)
    Input: An array A storing $n \geq 1$ integers
    Output : The maximum element in A

```
currentMax ← A[0]
for k ← 1 to n − 1 do
    if currentMax < A[k] then
        currentMax ← A[k]
    end
end
return currentMax
```

- Base case (initialization):
  - Before entering the loop, $currentMax$ holds the value of $A[0]$
  - Our loop invariant states that $currentMax$ must hold the maximum value in the first k elements in the array.
  - Since before we make the first iteration of the loop (the iteration when $k = 1$), the range of elements from 0 to k has only one element, $A[0]$.
  - So it is trivially true that $currentMax$ holds the maximum value in this range
- We have proven the loop invariant is true before entering the loop

Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

Loop Invariant Example

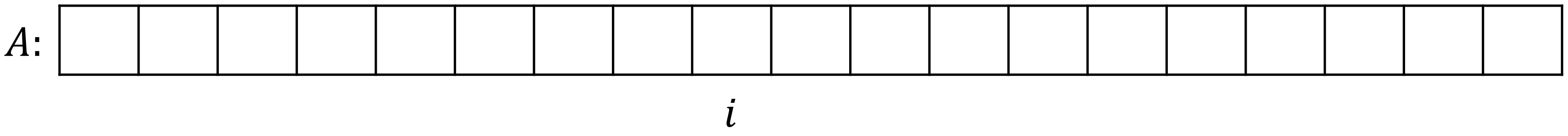
$currentMax$ holds the maximum value found in the first k elements of the array

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:

   \- Assume the invariant holds up to iteration i

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$



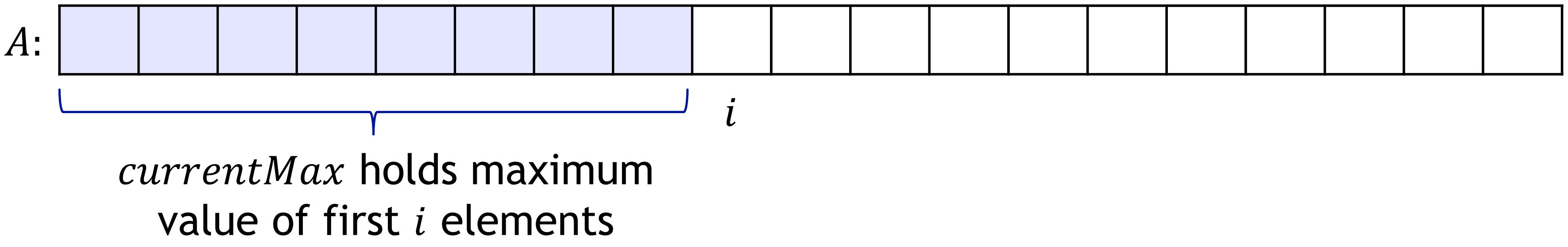
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$



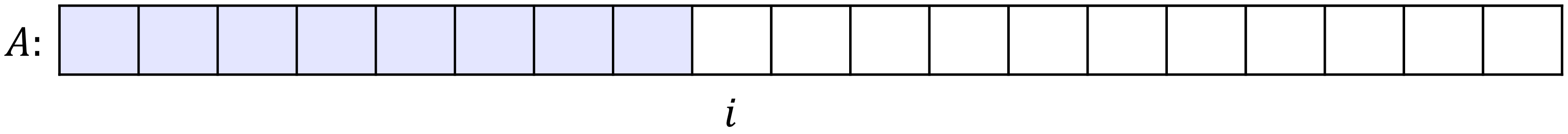
Loop Invariant Example

> $currentMax$ holds the maximum value found in the first k elements of the array

Algorithm arrayMax(A, n)
    Input: An array A storing $n \geq 1$ integers
    Output : The maximum element in A
    $currentMax \leftarrow A[0]$
    for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
        if $currentMax < A[k]$ **then**
            $currentMax \leftarrow A[k]$
        end
    end
    return $currentMax$

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+1



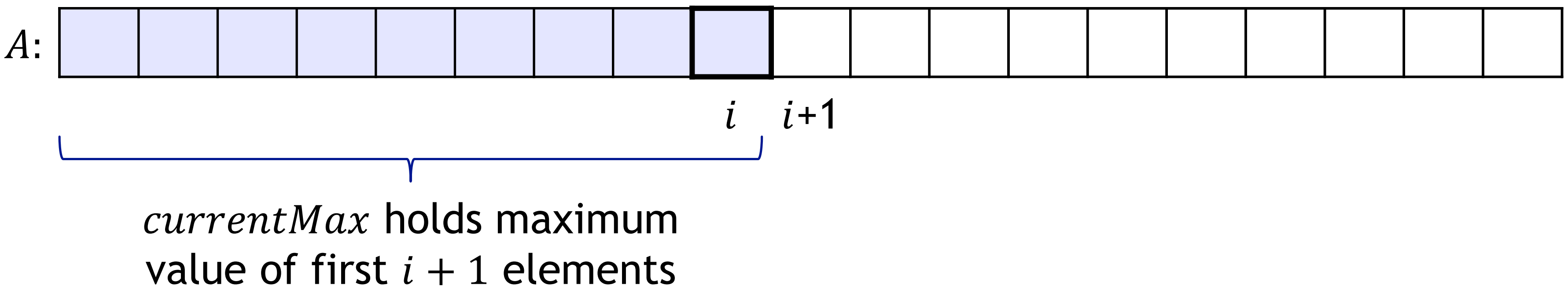
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+*1*

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$



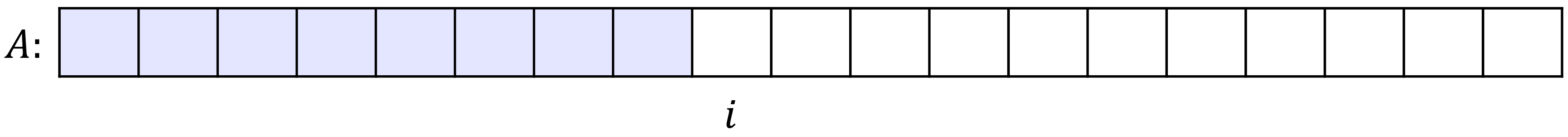
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+*1*



Now, let's show the loop invariant holds

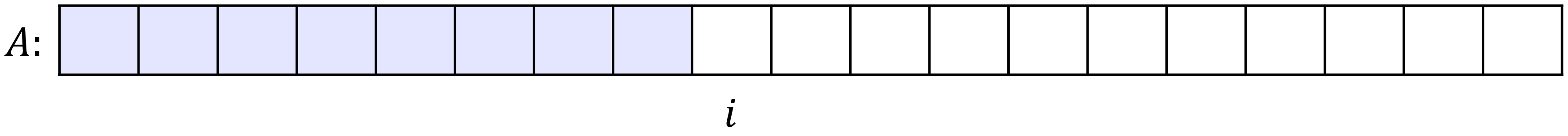
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

▶ Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+*1*



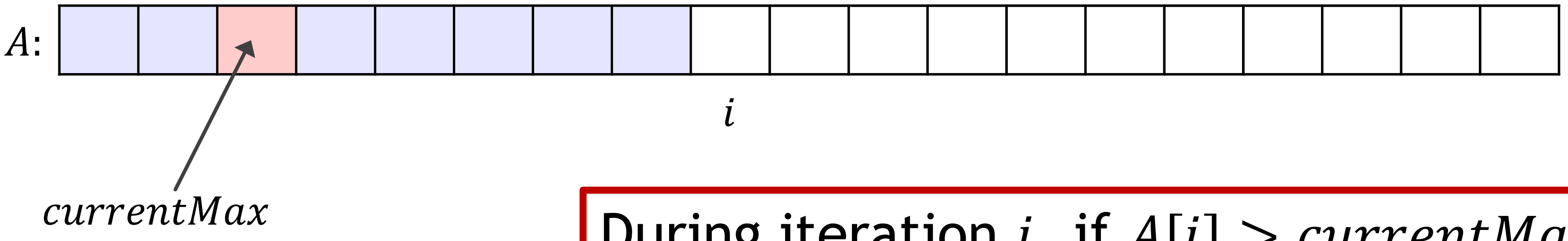
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+*1*

Algorithm arrayMax(A, n)
    Input: An array A storing $n \geq 1$ integers
    Output : The maximum element in A
    $currentMax \leftarrow A[0]$
    for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
        if $currentMax < A[k]$ **then**
            $currentMax \leftarrow A[k]$
        end
    end
    return $currentMax$



During iteration i, if $A[i] > currentMax$, then we assign $currentMax$ the value of $A[i]$

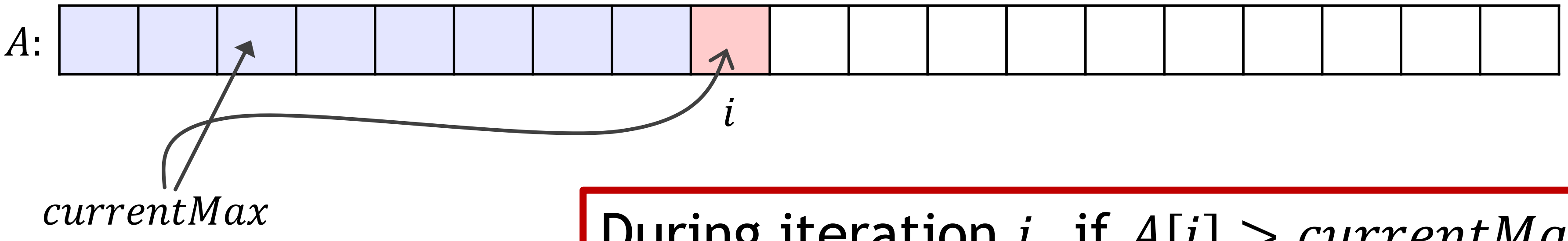
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $k \leftarrow 1$ to $n - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+*1*



During iteration i, if $A[i] > currentMax$, then we assign $currentMax$ the value of $A[i]$

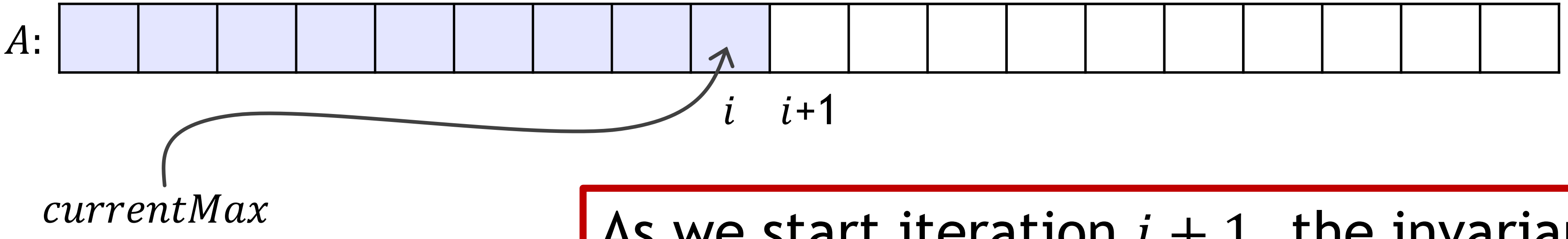
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+1

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$



As we start iteration $i + 1$, the invariant holds

Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

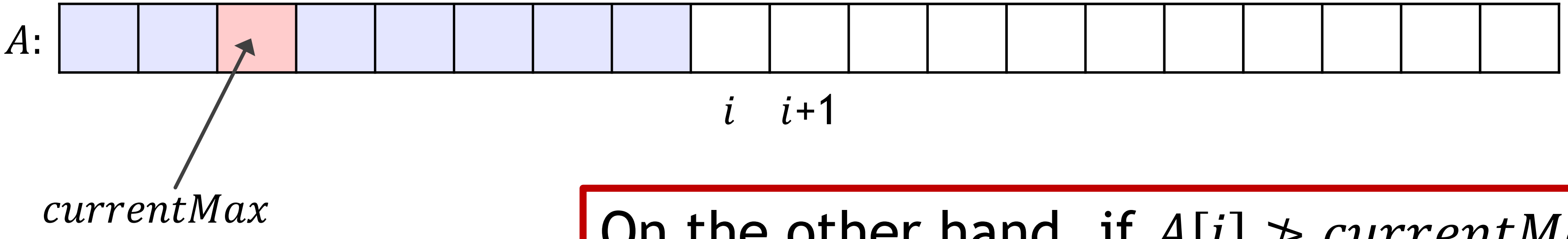
- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:

   - Assume the invariant holds up to iteration i

2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+1

Algorithm arrayMax(A, n)
 Input: An array A storing $n \geq 1$ integers
 Output : The maximum element in A
 $currentMax \leftarrow A[0]$
 for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
  if $currentMax < A[k]$ **then**
   $currentMax \leftarrow A[k]$
  end
 end
 return $currentMax$



On the other hand, if $A[i] \ngtr currentMax$, then the value of $currentMax$ does not change.

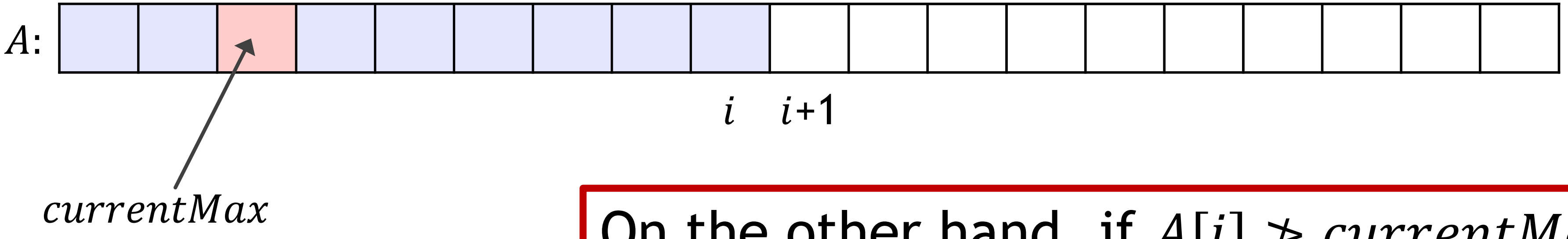
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+*1*



On the other hand, if $A[i] \not> currentMax$, then the value of $currentMax$ does not change.

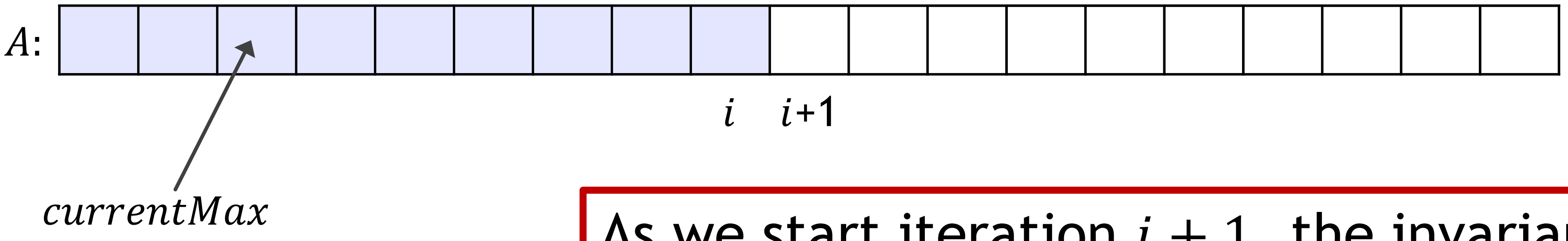
Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n} - 1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

- Induction Step (maintenance):

1. First, consider the *inductive hypothesis*:
   - Assume the invariant holds up to iteration i
2. Given the assumption from the inductive hypothesis, show the invariant holds at the start of the next iteration, i+*1*



As we start iteration $i + 1$, the invariant holds

Loop Invariant Example

$currentMax$ holds the maximum value found in the first k elements of the array

- Termination:
  - The loop ends after $\mathrm{n}-1$ iterations
  - When it ends, we were about to enter the nth iteration
  - Therefore, by our recently proven loop invariant, at this point we know $currentMax$ holds the maximum value found in the first n elements…
- Which means that the maximum value in the array is returned!

Algorithm arrayMax(A, n)
  Input: An array A storing $n \geq 1$ integers
  Output : The maximum element in A
  $currentMax \leftarrow A[0]$
  for $\mathrm{k} \leftarrow 1$ to $\mathrm{n}-1$ **do**
    if $currentMax < A[k]$ **then**
      $currentMax \leftarrow A[k]$
    end
  end
  return $currentMax$

Loop Invariants: Why?

- They are used for software verification and validation methods
- They can be essential in understanding the effects of a loop, or even help us in solving a problem (writing the loop):
  - What do we need to be true before we first enter the loop?
  - How can we ensure this property stays true after each iteration?
  - Does the loop eventually terminate?

- This allows us to verify the solution works!