# Program Verification: While Loops

HE Mingxin, Max

CS104: program07 @ yeah.net

CS108: mxhe1 @ yeah.net

I2ML(H) Spring 2023 (CS104|CS108)

# Exercises 15 : Reading and More

Record your time spent (in 0.1 hours) with brief tasks and durations in your learning log by hand writing!

1) Read textB-ch04-ProgramVerification.pdf (cont.)

# Outline

# Key Points

Key points to learn:

## Partial correctness for while loops

- Determine whether a given formula is an invariant for a while loop.
- Find an invariant for a given while loop.
- Prove that a Hoare triple is satisfied under partial correctness for a program containing while loops.

## Total correctness for while loops

- Determine whether a given formula is a variant for a while loop.
- Find a variant for a given while loop.
- Prove that a Hoare triple is satisfied under total correctness for a program containing while loops.

# Proving Total Correctness of While Loops

- Partial correctness
- Termination

# Proving Partial Correctness of While Loops

```
(| P |)
(| I |)                          implied (A)
while ( B ) {
    (| (I ∧ B) |)                partial-while
    C
    (| I |)    <justify based on C - a subproof>
}
(| (I ∧ (¬B)) |)                 partial-while
(| Q |)                          implied (B)
```

Proof of implied (A): $(P \rightarrow I)$
Proof of implied (B): $((I \land (\neg B)) \rightarrow Q)$

$I$ is called a loop invariant. We need to determine $I$!

# What is a loop invariant?

A loop invariant is:

- A relationship among the variables. (A predicate formula involving the variables.)
- The word "invariant" means something that does not change.
- It is true before the loop begins.
- It is true at the start of every iteration of the loop and at the end of every iteration of the loop.
- It is true after the loop ends.

# Proving partial correctness of while loops

Indicate the places in the program where the loop invariant is true.

$(\!|\, (x \geq 0) \,|\!)$

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

$(\!|\, (y = x!) \,|\!)$

# Proving partial correctness of a while loop

Steps to follow:

- Find a loop invariant.
- Complete the annotations.
- Prove any implied's.

How do we find a loop invariant???

# How do we find a loop invariant?

First, we need to understand the purpose of an invariant.

- The postcondition is the ultimate goal of our while loop.
- At every iteration, we are making progress towards the postcondition.
- The invariant is describing the progress we are making at every iteration.

# Partial While - Example 1

**Example 1:**
Prove that the following triple is satisfied under partial correctness.

$(\!|\, (x \geq 0) \,|\!)$
```
y = 1;
z = 0;
while (z != x) {
   z = z + 1;
   y = y * z;
}
```
$(\!|\, (y = x!) \,|\!)$

# Finding a loop invariant

Step 1: Write down the values of all the variables every time the while test is reached.

$(\!|\, (x \geq 0) \,|\!)$

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

$(\!|\, (y = x!) \,|\!)$

# Finding a loop invariant

Step 2: Find relationships among the variables that are true for every while test. These are our candidate invariants.

Come up with some invariants in the next 2 minutes.

$(\!|\, (x \geq 0) \,|\!)$

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

$(\!|\, (y = x!) \,|\!)$

| x | z | y |
|---|---|---|
| 5 | 0 | $1 = 0!$ |
| 5 | 1 | $1 = 1!$ |
| 5 | 2 | $2 = 2!$ |
| 5 | 3 | $6 = 3!$ |
| 5 | 4 | $24 = 4!$ |
| 5 | 5 | $120 = 5!$ |

# CQ 1 Is this a loop invariant?

**CQ 1:** Is $(\neg(z = x))$ a loop invariant?

(A) Yes (B) No (C) I don't know...

$(\!|\, (x \geq 0) \,|\!)$
```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```
$(\!|\, (y = x!) \,|\!)$

| x | z | y |
|---|---|---|
| 5 | 0 | $1 = 0!$ |
| 5 | 1 | $1 = 1!$ |
| 5 | 2 | $2 = 2!$ |
| 5 | 3 | $6 = 3!$ |
| 5 | 4 | $24 = 4!$ |
| 5 | 5 | $120 = 5!$ |

# CQ 2 Is this a loop invariant?

**CQ 2:** Is $(z \leq x)$ a loop invariant?

(A) Yes (B) No (C) I don't know...

$(\!| (x \geq 0) |\!)$
```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```
$(\!| (y = x!) |\!)$

| x | z | y |
|---|---|---|
| 5 | 0 | $1 = 0!$ |
| 5 | 1 | $1 = 1!$ |
| 5 | 2 | $2 = 2!$ |
| 5 | 3 | $6 = 3!$ |
| 5 | 4 | $24 = 4!$ |
| 5 | 5 | $120 = 5!$ |

# CQ 3 Is this a loop invariant?

**CQ 3:** Is $(y = z!)$ a loop invariant?

(A) Yes (B) No (C) I don't know...

$(\!|\, (x \geq 0) \,|\!)$

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

$(\!|\, (y = x!) \,|\!)$

| x | z | y |
|---|---|---|
| 5 | 0 | $1 = 0!$ |
| 5 | 1 | $1 = 1!$ |
| 5 | 2 | $2 = 2!$ |
| 5 | 3 | $6 = 3!$ |
| 5 | 4 | $24 = 4!$ |
| 5 | 5 | $120 = 5!$ |

# CQ 4 Is this a loop invariant?

**CQ 4:** Is $(y = x!)$ a loop invariant?

(A) Yes (B) No (C) I don't know...

$(\!|(x \geq 0)|\!)$

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

$(\!|(y = x!)|\!)$

| x | z | y |
|---|---|---|
| 5 | 0 | $1 = 0!$ |
| 5 | 1 | $1 = 1!$ |
| 5 | 2 | $2 = 2!$ |
| 5 | 3 | $6 = 3!$ |
| 5 | 4 | $24 = 4!$ |
| 5 | 5 | $120 = 5!$ |

# CQ 5 Is this a loop invariant?

**CQ 5:** Is $((z \leq x) \wedge (y = z!))$ a loop invariant?

(A) Yes (B) No (C) I don't know...

$(\!|\, (x \geq 0) \,|\!)$

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

$(\!|\, (y = x!) \,|\!)$

| x | z | y |
|---|---|---|
| 5 | 0 | $1 = 0!$ |
| 5 | 1 | $1 = 1!$ |
| 5 | 2 | $2 = 2!$ |
| 5 | 3 | $6 = 3!$ |
| 5 | 4 | $24 = 4!$ |
| 5 | 5 | $120 = 5!$ |

# Finding a loop invariant

Step 3: Try each candidate invariant until we find one that works for our proof.

$(\!|\, (x \geq 0) \,|\!)$

```
y = 1;
z = 0;
while (z != x) {
   z = z + 1;
   y = y * z;
}
```

$(\!|\, (y = x!) \,|\!)$

| x | z | y |
|---|---|---|
| 5 | 0 | $1 = 0!$ |
| 5 | 1 | $1 = 1!$ |
| 5 | 2 | $2 = 2!$ |
| 5 | 3 | $6 = 3!$ |
| 5 | 4 | $24 = 4!$ |
| 5 | 5 | $120 = 5!$ |

# How do we find an invariant?

A recap of the steps to find an invariant:

- Write down the values of all the variables every time the while test is reached.
- Find relationships among the variables that are true for every while test. These are our candidate invariants.
- Try each candidate invariant until we find one that works for our proof.

# Partial While - Example 1 (($z \leq x$) as the invariant)

$(\!|\ (x \geq 0)\ |\!)$
$(\!|\ (0 \leq x)\ |\!)$ implied (A)
```
y = 1;
```
$(\!|\ (0 \leq x)\ |\!)$ assignment
```
z = 0;
```
$(\!|\ (z \leq x)\ |\!)$ assignment
```
while (z != x) {
```
$(\!|\ ((z \leq x) \wedge (\neg(z = x)))\ |\!)$ partial−**while**
$(\!|\ (z + 1 \leq x)\ |\!)$ implied (B)
```
  z = z + 1;
```
$(\!|\ (z \leq x)\ |\!)$ assignment
```
  y = y * z;
```
$(\!|\ (z \leq x)\ |\!)$ assignment
```
}
```
$(\!|\ ((z \leq x) \wedge (\neg(\neg(z = x))))\ |\!)$ partial−**while**
$(\!|\ (y = x!)\ |\!)$ implied (C)

# CQ 7 Is there a proof for implied (A)?

We used $(z \leq x)$ as the invariant.

**CQ 7:** Is there a proof for implied (A)?

$$((x \geq 0) \rightarrow (0 \leq x))$$

(A) Yes
(B) No
(C) I don't know.

# CQ 8 Is there a proof for implied (B)?

We used $(z \leq x)$ as the invariant.

**CQ 8:** Is there a proof for implied (B)?

$$(((z \leq x) \wedge (\neg(z = x))) \rightarrow (z + 1 \leq x))$$

(A) Yes

(B) No

(C) I don't know.

# CQ 9 Is there a proof for implied (C)?

We used $(z \leq x)$ as the invariant.

**CQ 9:** Is there a proof for implied (C)?

$$(((z \leq x) \wedge (\neg(\neg(z = x)))) \rightarrow (y = x!))$$

(A) Yes

(B) No

(C) I don't know.

# Partial While - Example 2

**Example 2:**

Prove that the following triple is satisfied under partial correctness.

$(\!|\,(x \geq 0)\,|\!)$

```
y = 1;
z = 0;
while (z < x) {
   z = z + 1;
   y = y * z;
}
```

$(\!|\,(y = x!)\,|\!)$

Let's try using $(y = z!)$ as the invariant in our proof.

# Which invariant leads to a valid proof?

To check whether an invariant leads to a valid proof,
we need to check whether all of the implied's can be proved.

# CQ 11 Is there a proof for implied (A)?

We used $(y = z!)$ as the invariant.

**CQ 11:** Is there a proof for implied (A)?

$$((x \geq 0) \rightarrow (1 = 0!))$$

(A) Yes

(B) No

(C) I don't know.

# CQ 12 Is there a proof for implied (B)?

We used $(y = z!)$ as the invariant.

**CQ 12:** Is there a proof for implied (B)?

$$(((y = z!) \wedge (z < x)) \rightarrow (y * (z + 1) = (z + 1)!))$$

(A) Yes

(B) No

(C) I don't know.

# CQ 13 Is there a proof for implied (C)?

We used $(y = z!)$ as the invariant.

**CQ 13:** Is there a proof for implied (C)?

$$(((y = z!) \land (\neg(z < x))) \rightarrow (y = x!))$$

(A) Yes

(B) No

(C) I don't know.

# CQ 14 Is there a proof for implied (C)?

We used $((y = z!) \land (z \leq x))$ as the invariant.

**CQ 14:** Is there a proof for implied (C)?

$$(((y = z!) \land (z \leq x)) \land (\neg(z < x))) \rightarrow (y = x!))$$

(A) Yes

(B) No

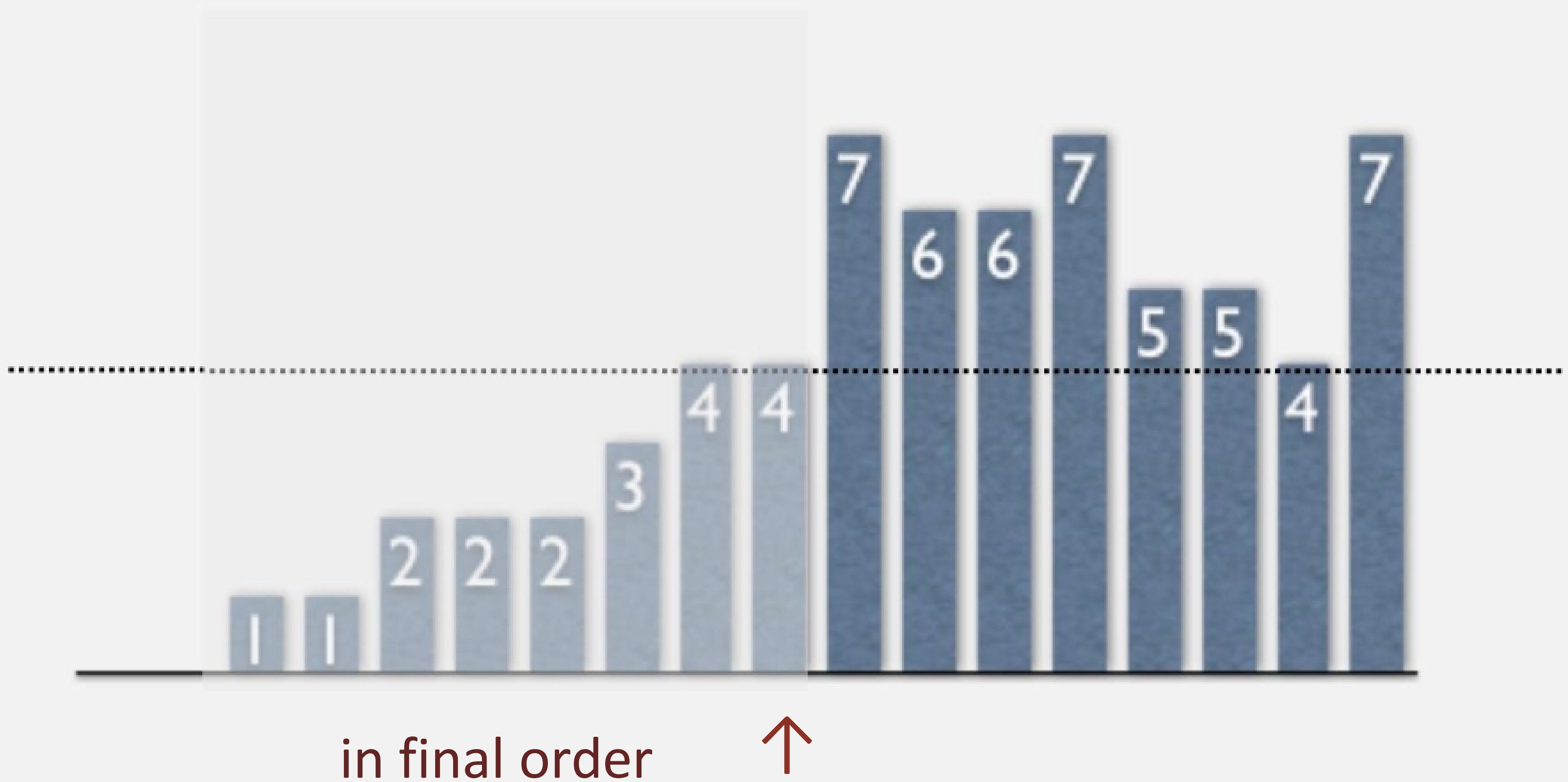
(C) I don't know.

# Selection sort

Algorithm. ↑ scans from left to right.

Invariants.

- Entries the left of ↑ (including ↑) fixed and in ascending order.
- No entry to right of ↑ is smaller than any entry to the left of ↑.



# Insertion sort

**Algorithm.** ↑ scans from left to right.

**Invariants.**

- Entries to the left of ↑ (including ↑) are in ascending order.
- Entries to the right of ↑ have not yet been seen.

# Quicksort Partitioning

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |
| ↑ lo | ↑ i | | | | | | | | | | | | | | ↑ j hi |

https://algs4.cs.princeton.edu/lectures/demo/23DemoPartitioning.pdf

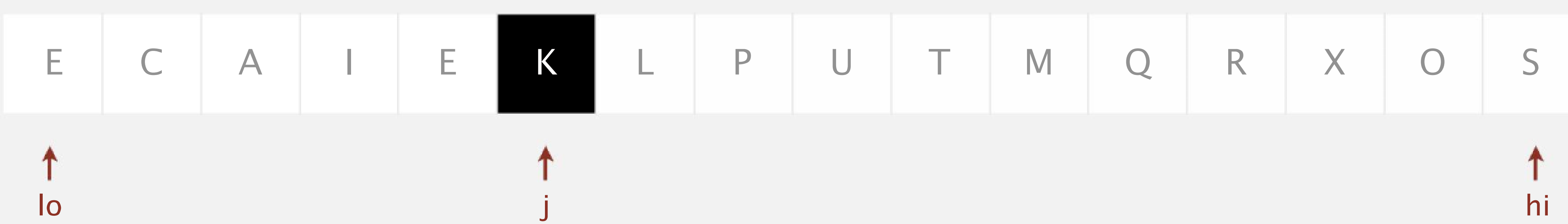
# Quicksort Partitioning

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

When pointers cross.

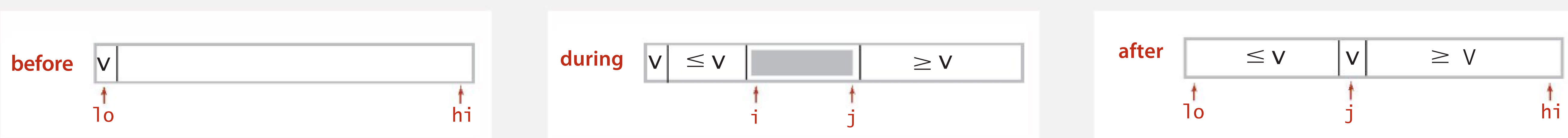
- Exchange `a[lo]` with `a[j]`.




partitioned!

# Quicksort: Java code for partitioning

```
private static int partition(Comparable[] a, int lo, int hi) {

   int i = lo, j = hi+1;

   while (true) {

      while (less(a[++i], a[lo]))              find item on left to swap
         if (i == hi) break;

      while (less(a[lo], a[--j]))              find item on right to swap
         if (j == lo) break;

      if (i >= j) break;                       check if pointers cross
      exch( a, i, j);                          swap
   }

   exch( a, lo, j);                            swap with partitioning item
   return j;                                   return index of item now known to be in place
}
```



# Quicksort: Java implementation

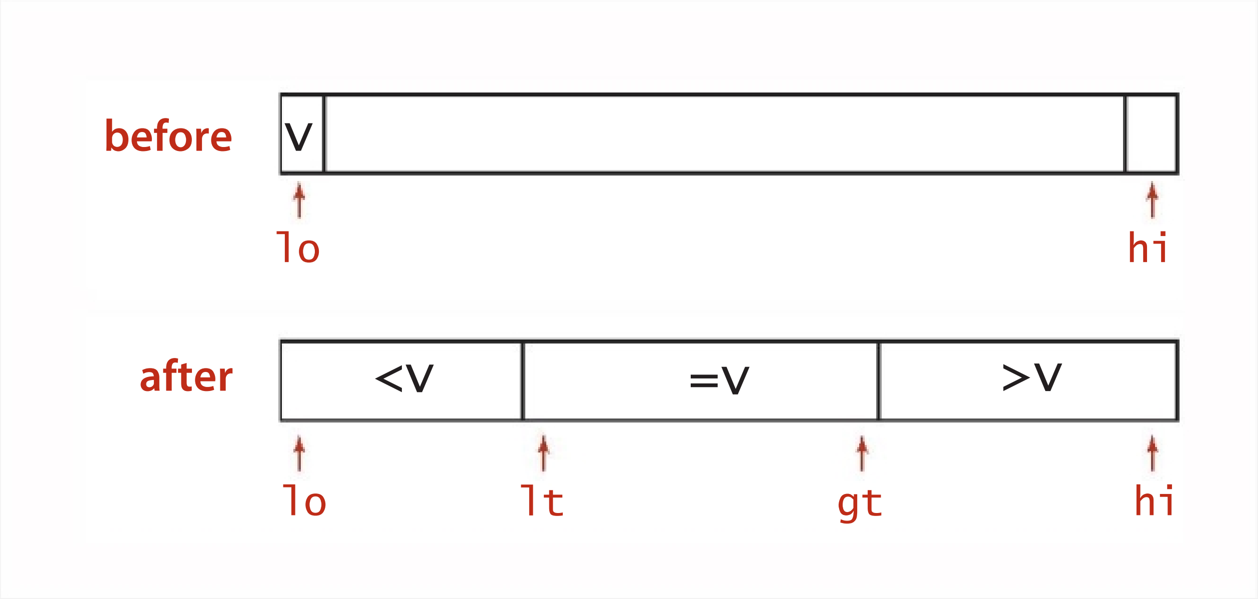
```
public class Quick {

   private static int partition(Comparable[] a, int lo, int hi) {
       /* see previous slide */
   }

   public static void sort (Comparable[] a) {

      sort( a, 0, a.length - 1);
   }


   private static void sort (Comparable[] a, int lo, int hi) {
      if (hi <= lo) return;
      int j = partition( a, lo, hi);
      sort( a, lo, j-1);
      sort( a, j+1, hi);
  }
}
```

Call overloaded sort method

# 3-way Partitioning

Goal. Partition array into three parts so that:

- Entries between `lt` and `gt` equal to the partition item.
- No larger entries to left of `lt`.
- No smaller entries to right of `gt`.

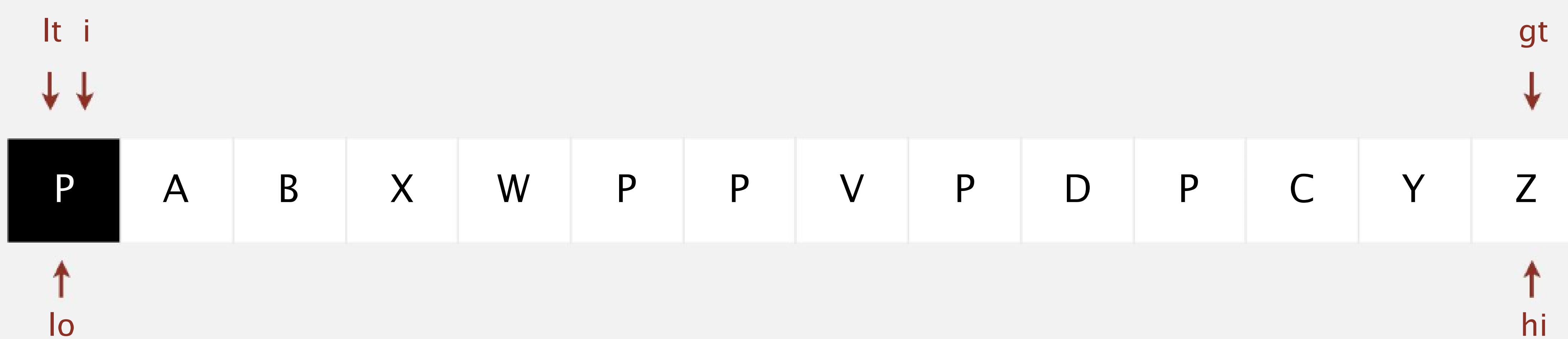



Dutch national flag problem. [Edsger Dijkstra]

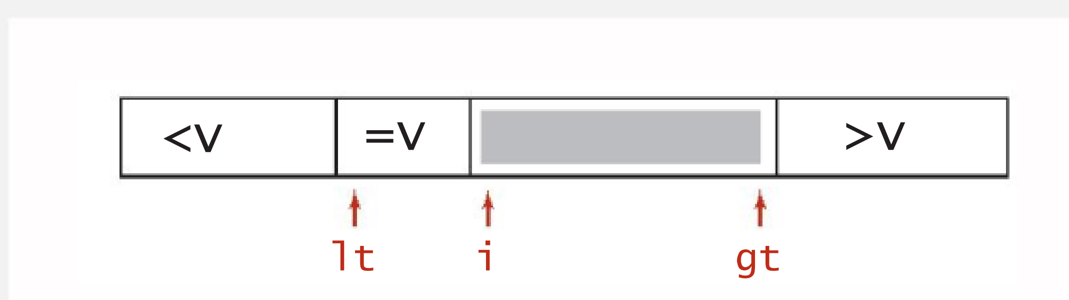
- Conventional wisdom until mid 1990s: not worth doing.
- Now incorporated into C library `qsort()` and Java 6 system sort.

# Dijkstra 3-way partitioning

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i] < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i] > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

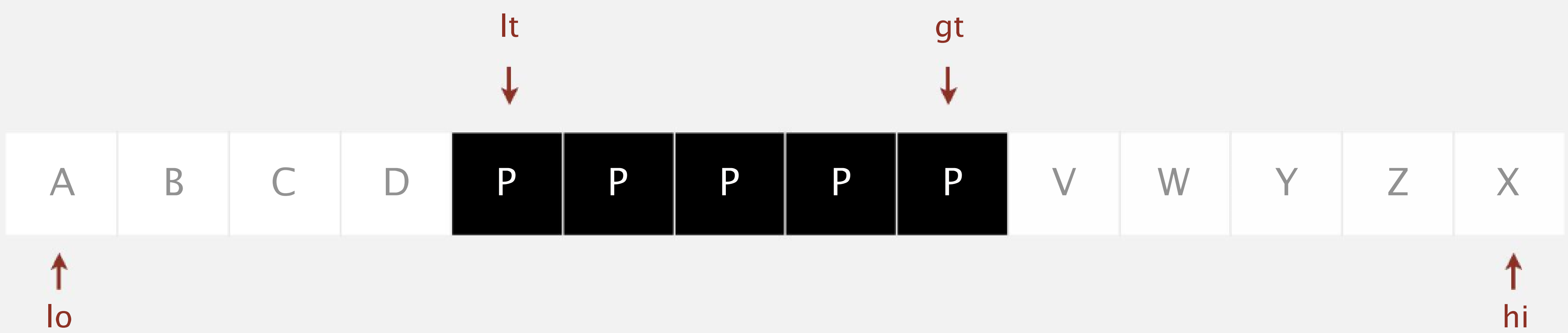



**invariant**

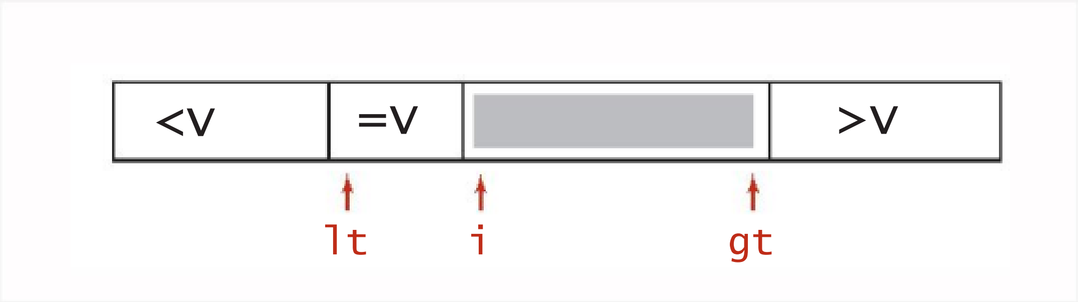


# Dijkstra 3-way partitioning

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

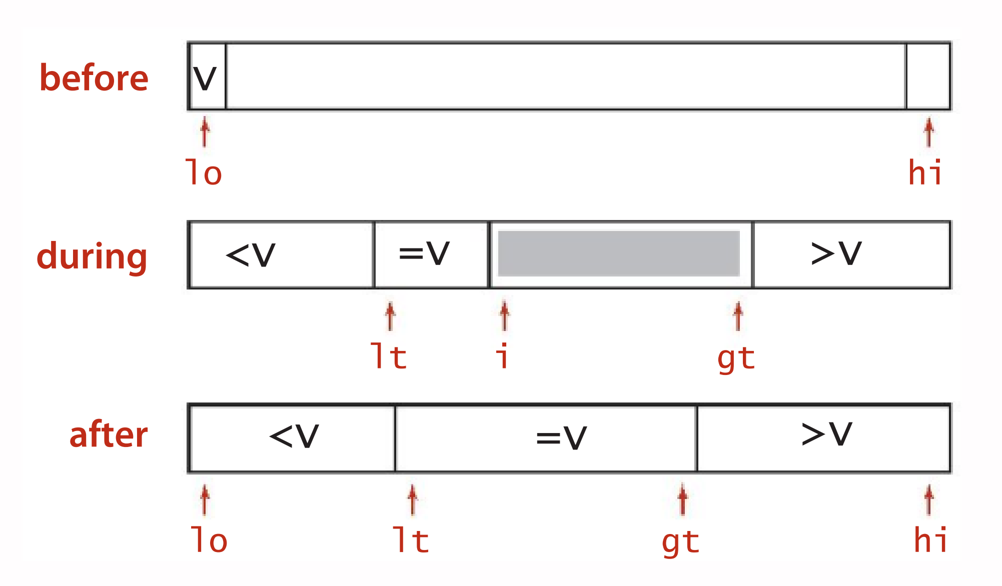



**invariant**



# 3-way quicksort: Java implementation

```
private static void sort (Comparable[] a, int lo, int hi) {
   if (hi <= lo) return;

   int lt = lo, gt = hi;
   Comparable v = a[lo];
   int i = lo;

   while (i <= gt) {
      int cmp = a[i].compareTo(v);
      if      (cmp < 0) exch( a, lt++, i++);
      else if (cmp > 0) exch( a, i, gt--);
      else              i++;
   }

   sort( a, lo, lt - 1);
   sort( a, gt + 1, hi);
}
```



# Proving Termination

Find an integer expression that

- is non-negative before the loop starts, at every iteration of the loop, and after the loop ends.
- decreases by at least 1 at every iteration of the loop.

This integer expression is called a variant (something that changes).

The loop must terminate because a non-negative integer can decrease by 1 a finite number of times.

# Example 2: Finding a variant

**Example 2:**
Prove that the following program terminates.

```
y = 1;
z = 0;
while (z < x) {
   z = z + 1;
   y = y * z;
}
```

How do we find a variant? The loop guard ($z < x$) helps.

# Example 2: Proof of Termination

Consider the variant $(x - z)$.

Before the loop starts, $(x - z) \geq 0$ because the precondition is $(x \geq 0)$ and the second assignment mutates $z$ to be 0.

During every iteration of the loop, $(x - z)$ decreases by 1 because $x$ does not change and $z$ increases by 1.

Thus, $x - z$ will eventually reach 0.

When $x - z = 0$, the loop guard $z < x$ will terminate the loop.

# Summary

Key points learnt:

### Partial correctness for while loops

- Determine whether a given formula is an invariant for a while loop.
- Find an invariant for a given while loop.
- Prove that a Hoare triple is satisfied under partial correctness for a program containing while loops.

### Total correctness for while loops

- Determine whether a given formula is a variant for a while loop.
- Find a variant for a given while loop.
- Prove that a Hoare triple is satisfied under total correctness for a program containing while loops.