An Axiomatic Basis for Computer Programming

C. A. R. Hoare

October, 1969

Computer Programming and Science

Computer Programming = Exact Science

- What is Programming

  Programming: The writing of a computer program

  Program: A set of coded instructions that enables a machine, especially a computer, to perform a desired sequence of operations

- What is Science

  Science: The observation, identification, description, experimental investigation, and theoretical explanation of phenomena

Reasoning on a Program

Input Data $\rightarrow$ [Computer Operations] $\rightarrow$ Result

- Reasoning on What?
  - Reasoning on the relations between the involved entities
  - The involved entities are the input data and the result

Computer Arithmetic

(Pure) Arithmetic $\neq$ Computer Arithmetic

- Computer Arithmetic
  - Typically supported by a specific computer hardware
  - Could only deal with some finite subsets of integers (or real numbers)
    $\rightarrow$ Overflow
- Overflow Handling Examples (for Integer Operations)
  - **Strict Interpretation**: an overflow operation never completes
  - **Firm Boundary**: take the maximum or the minimum
  - **Modulo Arithmetic**: modulo n, where n is the size of the set

Strict Interpretation

1. Strict Interpretation

+	0	1	2	3
0	0	1	2	3
1	1	2	3	*
2	2	3	*	*
3	3	*	*	*

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	*	*
3	0	3	*	*

* nonexistent

Firm Boundary

2. Firm Boundary

+	0	1	2	3
0	0	1	2	3
1	1	2	3	3
2	2	3	3	3
3	3	3	3	3

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	3
3	0	3	3	3

Modulo Arithmetic

3. Modulo Arithmetic

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

A Selection of Axioms for Integers

A1 $x + y = y + x$

A2 $x \times y = y \times x$

A3 $(x + y) + z = x + (y + z)$

A4 $(x \times y) \times z = x \times (y \times z)$

A5 $x \times (y + z) = x \times y + x \times z$

A6 $y \leqslant x \ \supset \ (x - y) + y = x$

A7 $x + 0 = x$

A8 $x \times 0 = 0$

A9 $x \times 1 = x$

An Example of Theorem

$$x = x + y \times 0$$

Proof.

$$x = x + 0 \qquad \text{(A7)}$$
$$= x + y \times 0 \qquad \text{(A8)}$$

□

Another Example of Theorem

$$y \leqslant r \;\supset\; r + y \times q = (r - y) + y \times (1 + q)$$

Proof.

$$\begin{aligned}
(r - y) + y \times (1 + q) &= (r - y) + (y \times 1 + y \times q) && \text{(A5)} \\
&= (r - y) + (y + y \times q) && \text{(A9)} \\
&= ((r - y) + y) + y \times q && \text{(A3)} \\
&= r + y \times q \quad \text{provided } y \leqslant r && \text{(A6)}
\end{aligned}$$

□

Some Remarks

- The premise $(y \leqslant r)$ is required because the addition is defined for non-negative integers
- In this respect, additional restrictions are needed for the previous theorems

$$0 \leqslant x \leqslant n \ \wedge \ 0 \leqslant y \leqslant n \supset x = x + y \times 0$$

Axioms for Finiteness

- The 10th Axiom for Infinite Arithmetic

  $\mathbf{A10}_\mathrm{I} \quad \neg\exists x\ \forall y\ \ (y \leqslant x)$

- The 10th Axiom for Finite Arithmetic

  $\mathbf{A10}_\mathrm{F} \quad \forall x\ \ (x \leqslant max)$

But, what about ∞?

Axioms for Overflow Handling

$\mathbf{A11}_S$ $\neg\exists x \ \ (x = max + 1)$

$\mathbf{A11}_B$ $max + 1 = max$

$\mathbf{A11}_M$ $max + 1 = 0$

Modelling of Program Execution

"If P is true before initiation of a program Q,
then R will be true on its completion."

$$P\{Q\}R$$

where

- P : precondition (predicate)
- Q : program (sequence of statements)
- R : postcondition (predicate)

cf. If no preconditions are imposed,

$$\mathbf{true}\{Q\}R$$

An Axiomatic System

- An axiomatic system for program verification will be developed
- The axiomatic system consists of:
  - **Axioms** which are true without any premises
  - **Rules** which are used to derive a theorem from existing theorems

Axiom of Assignment (D0)

$P[f/x]\ \{x := f\}\ P$

where

x is a variable identifier

f is an expression without side effects

$P[f/x]$ is obtained from P by substituting f for all occurrences of x

Rules of Consequences (D1)

- **Weakening the postcondition**
  If $P\{Q\}R$ and $R \supset S$ then $P\{Q\}S$
- **Strengthen the precondition**
  If $P\{Q\}R$ and $S \supset P$ then $S\{Q\}R$

Another notation:

$$\frac{P\{Q\}R,\ \ R \supset S}{P\{Q\}S} \quad \frac{S \supset P,\ \ P\{Q\}R}{S\{Q\}R}$$

Rule of Composition (D2)

If $P\{Q_1\}R_1$ and $R_1\{Q_2\}R$ then $P\{Q_1; Q_2\}R$

- **Sequencing the Statements**

$$\frac{P\{Q_1\}R_1,\ \ R_1\{Q_2\}R}{\{Q_1; Q_2\}R}$$

- **Zero Composition** (empty statement)

$$P\{skip\}P$$

Rule of Iteration

If $P \wedge B\{S\}P$ then $P\{\textbf{while } B \textbf{ do } S\}\neg B \wedge P$

Another notation:

$$\frac{P \wedge B\{S\}P}{P\{\textbf{while } B \textbf{ do } S\}\neg B \wedge P}$$

- P is called a *loop invariant*.
  - P is true on initiation of the loop (or of S)
  - P is true on completion of the loop
  - P is true on completion of S

An Example

Program

Compute the quotient and the remainder when we divide x by y.

$$Q: \quad \boxed{\begin{array}{l} ((r := x; q := 0); \\ \textbf{while } y \leqslant r \textbf{ do } (r := r - y; q := 1 + q)) \end{array}}$$

Program Property

$\textbf{true} \;\; \{Q\} \;\; \neg y \leqslant r \;\wedge\; x = r + y \times q$

Lemma 1.

$$\textbf{true} \;\supset\; x = x + y \times 0$$

Lemma 2.

$$x = r + y \times q \;\wedge\; y \leqslant r \;\supset\; x = (r - y) + y \times (1 + q)$$

Proving Steps (1/3)

1	$\textbf{true} \supset x = x + y \times 0$	Lemma 1
2	$x = x + y \times 0 \;\{r := x\}\; x = r + y \times 0$	D0
3	$x = r + y \times 0 \;\{q := 0\}\; x = r + y \times q$	D0
4	$\textbf{true} \;\{r := x\}\; x = r + y \times 0$	D1 (1,2)
5	$\textbf{true} \;\{r := x; q := 0\}\; x = r + y \times q$	D2 (4,3)

Proving Steps (2/3)

6	$x = r + y \times q \ \wedge \ y \leqslant r$	
	$\supset \ x = (r - y) + y \times (1 + q)$	Lemma2
7	$x = (r - y) + y \times (1 + q)$	
	$\{r := r - y\} \ \ x = r + y \times (1 + q)$	D0
8	$x = r + y \times (1 + q)$	
	$\{q := 1 + q\} \ \ x = r + y \times q$	D0
9	$x = (r - y) + y \times (1 + q)$	
	$\{r := r - y; q := 1 + q\} \ \ x = r + y \times q$	D2 (7,8)
10	$x = r + y \times q \ \wedge \ y \leqslant r$	
	$\{r := r - y; q := 1 + q\} \ \ x = r + y \times q$	D1 (6,9)

Proving Steps (3/3)

11 $x = r + y \times q$

$\{\textbf{while}\ y \leqslant r\ \textbf{do}\ (r := r - y; q := 1 + q)\}$

$\neg y \leqslant r\ \wedge\ x = r + y \times q$ D3 (10)

12 $\textbf{true}\ \{((r := x; q := 0);$

$\textbf{while}\ y \leqslant r\ \textbf{do}\ (r := r - y; q := 1 + q))\}$

$\neg y \leqslant r\ \wedge\ x = r + y \times q$ D2 (5,11)

Additional Rules

- Conditional 1

$$\frac{\mathrm{P} \wedge \mathrm{B}\ \{S\}\ \mathrm{Q}}{\mathrm{P}\ \{\textbf{if}\ \mathrm{B}\ \textbf{then}\ S\}\ \mathrm{Q}}$$

- Conditional 2

$$\frac{\mathrm{P} \wedge \mathrm{B}\ \{S_1\}\ \mathrm{Q},\ \ \mathrm{P} \wedge \neg\mathrm{B}\ \{S_2\}\ \mathrm{Q}}{\mathrm{P}\ \{\textbf{if}\ \mathrm{B}\ \textbf{then}\ S_1\ \textbf{else} S_2\}\ \mathrm{Q}}$$

Proving During Coding

input variables → PROGRAM → output variables

- Think of Assertions
  - The assertions (including preconditions and postconditions) are described in terms of variables
  - The PROGRAM may defines additional intermediate variables
- Kinds of Assertions
  - The input variables should satisfy some *preconditions*.
  - The output variables should satisfy some *postconditions*.
  - The intermediate variables should satisfy some *invariants*.

Coding and Proving Steps

Coding	**Proving**
determining input/output variables	determining preconditions/postconditions (problem specification)
determining intermediate variables	formulating assertions on the intermediate variables (the purpose of the variables)
determining the initial values for the intermediate variables	checking the assertions
refinement	

The Program "Find"

- Find an element of an array $A[1..N]$ whose value is f-th in order of magnitude, i.e.:

$$A[1], A[2], \ldots, A[f-1] \leqslant A[f] \leqslant A[f+1], \ldots, A[N]$$

- An Algorithm for Find

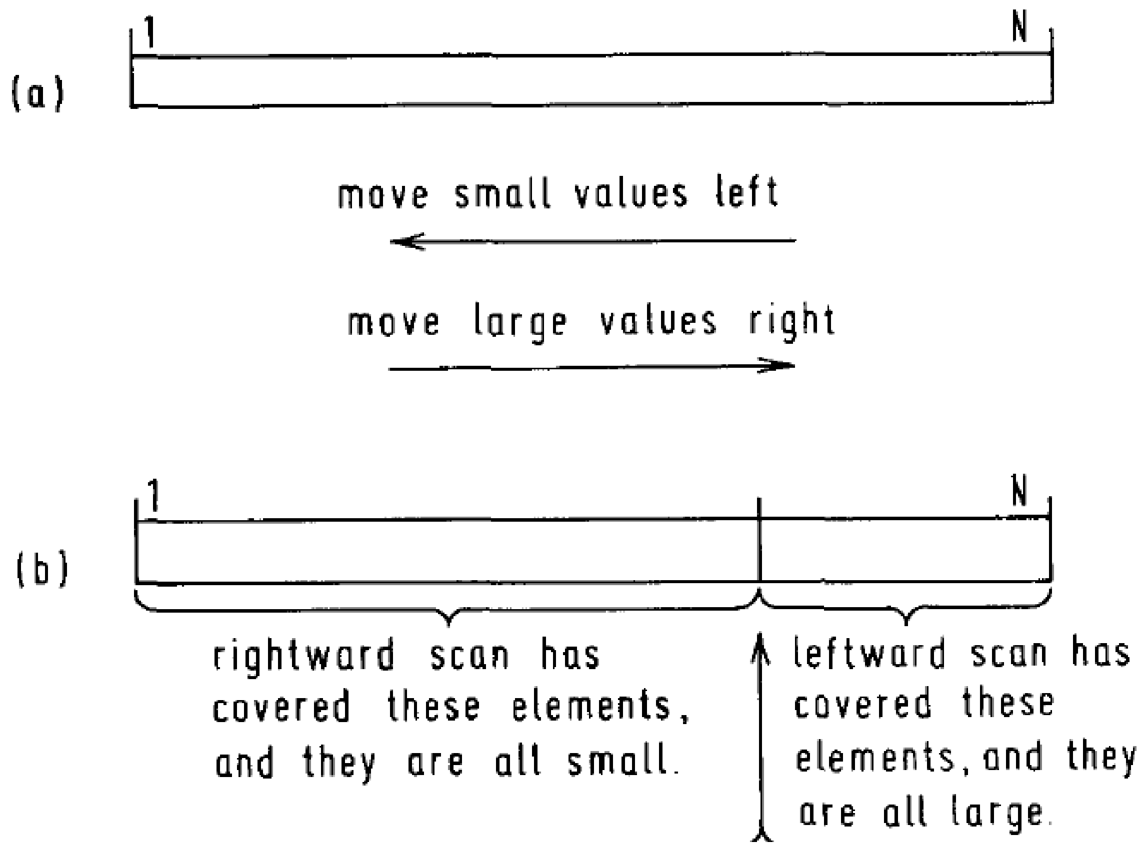
1. For a specific element r (say, $A[f]$), split $A[m..n]$ into two parts:

$$A[m], \ldots, A[k], \quad A[k+1], \ldots A[n]$$

   where $A[m], \ldots, A[k] \leqslant r$ and $A[k+1], \ldots A[n] \geqslant r$

2. If $f \in [m, k]$, $n := k$ and continue.
3. If $f \in [k+1, n]$, $m := k+1$ and continue.
4. If $m = n = k$, terminates.

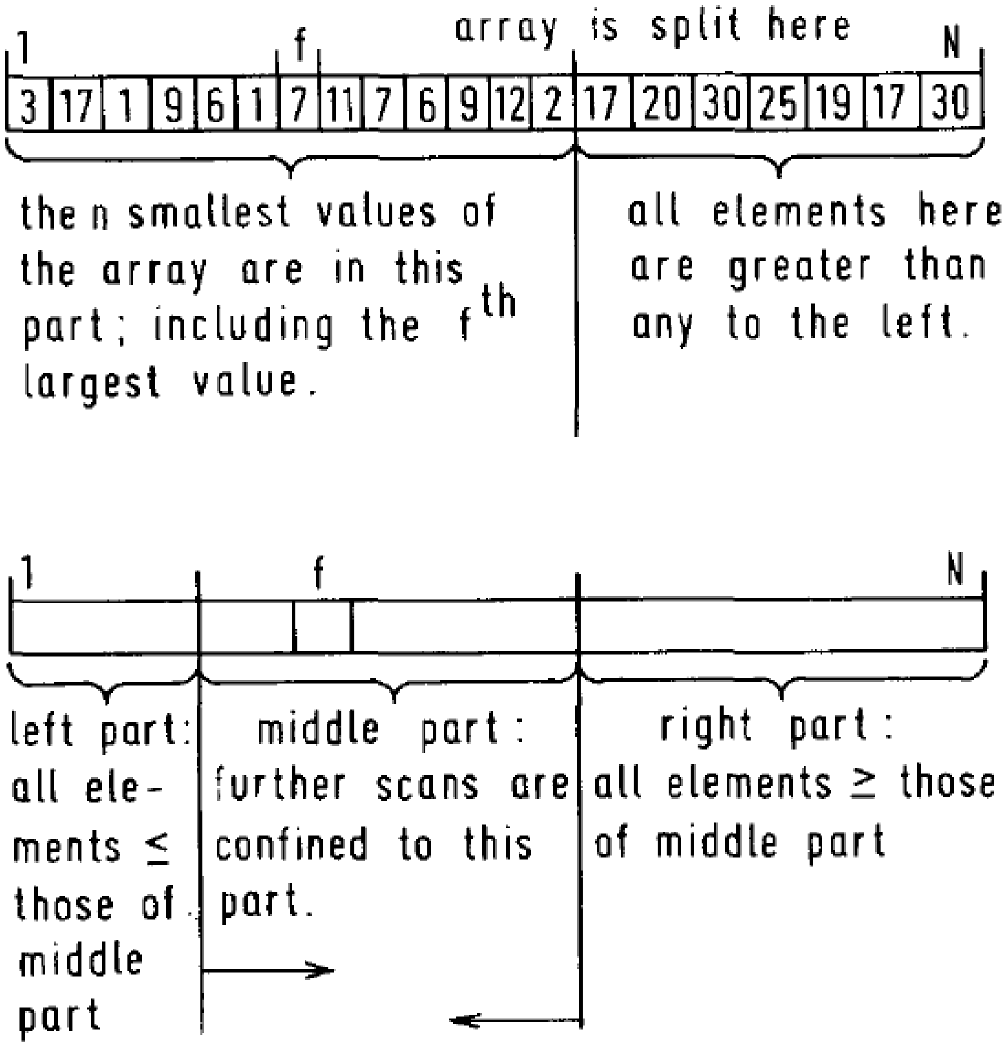
The Algorithm (1/2)



The Algorithm (2/2)

(c)

(d)



Stage 1: Problem Definition

- (Precondition) Given $A[1..N]$ and $1 \leqslant f \leqslant N$
- (Postcondition) Make A into

$$\forall p, q(1 \leqslant p \leqslant f \leqslant q \leqslant N \supset A[p] \leqslant A[f] \leqslant A[q]) \qquad \text{(FOUND)}$$

Stage 2: Finding the Middle Part (1/4)

- Identifying intermediate variables m and n
  where $A[m]$ is for the first element of the middle part
  and $A[n]$ is the last element of the middle part
- The purpose of m and n

$$m \leqslant f \quad \wedge \quad \forall p, q(1 \leqslant p < m \leqslant q \leqslant N \ \supset \ A[p] \leqslant A[q]) \qquad (m\text{-inv.})$$

$$f \leqslant n \quad \wedge \quad \forall p, q(1 \leqslant p \leqslant n < q \leqslant N \ \supset \ A[p] \leqslant A[q]) \qquad (n\text{-inv.})$$

- Determining the initial values for m and n:

$$m := 1; n := N$$

Stage 2: Finding the Middle Part (2/4)

- Check the invariants for the initial values

$$1 \leqslant f \quad \wedge \quad \forall p, q(1 \leqslant p < 1 \leqslant q \leqslant N \;\supset\; A[p] \leqslant A[q])$$

(Lemma 1 = m-inv.[1/m])

$$f \leqslant N \quad \wedge \quad \forall p, q(1 \leqslant p \leqslant N < q \leqslant N \;\supset\; A[p] \leqslant A[q])$$

(Lemma 2 = n-inv.[N/n])

Lemma 1 and Lemma 2 are trivially true because $1 \leqslant f \leqslant N$

Stage 2: Finding the Middle Part (3/4)

- Refine further (identifying a loop)

$$\textbf{while } m < n \textbf{ do } \text{“}\textit{reduce the middle part}\text{”}$$

- Does the loop accomplishes the objective of the program?

$$m\text{-inv.} \quad \wedge \quad n\text{-inv.} \quad \wedge \quad \neg(m < n)$$

$$\supset \quad m = n = f \quad \wedge \quad \forall p, q(1 \leqslant p \leqslant f \leqslant q \leqslant N \ \supset \ A[p] \leqslant A[f] \leqslant A[q])$$

(Lemma 3)

Stage 2: Finding the Middle Part (4/4)

- The current program structure:

$m := 1; n := N$

while $m < n$ **do**

    "*reduce the middle part*"

Stage 3: Reduce the Middle Part (1/6)

- Variables

$$i,\ j : \text{the pointers for the scanning}$$
$$r : \text{an discriminator}$$

- Invariants

$$m \leqslant i \quad \wedge \quad \forall p(1 \leqslant p < i \ \supset \ A[p] \leqslant r) \qquad \text{(i-inv.)}$$
$$j \leqslant n \quad \wedge \quad \forall q(j < q \leqslant N \ \supset \ r \leqslant A[q]) \qquad \text{(j-inv.)}$$

- Initial values

$$i := m; j := n$$

Stage 3: Reduce the Middle Part (2/6)

- Check the Invariants

$$m\text{-inv.} \supset i\text{-inv.}[m/i]$$

$$n\text{-inv.} \supset j\text{-inv.}[n/i]$$

Specifically,

$$1 \leqslant f \quad \wedge \quad \forall p, q(1 \leqslant p < 1 \leqslant q \leqslant N \ \supset \ A[p] \leqslant A[q])$$

$$\supset \ m \leqslant m \quad \wedge \quad \forall p(1 \leqslant p < m \ \supset \ A[p] \leqslant r) \qquad \text{(Lemma 4)}$$

$$f \leqslant N \quad \wedge \quad \forall p, q(1 \leqslant p \leqslant N < q \leqslant N \ \supset \ A[p] \leqslant A[q])$$

$$\supset \ n \leqslant n \quad \wedge \quad \forall q(n < q \leqslant N \ \supset \ r \leqslant A[q]) \qquad \text{(Lemma 5)}$$

Stage 3: Reduce the Middle Part (3/6)

- Changing i and j (Scanning)

  while $i \leqslant j$ **do**

  "increase i and decrease j"

- Updating m and n

  if $f \leqslant j$ **then** $n := j$

  else if $i \leqslant f$ **then** $m := i$

  else go to L

Stage 3: Reduce the Middle Part (4/6)

- Checking the Invariants

$$\begin{array}{rl} & j < i \quad \wedge \quad i\text{-inv.} \quad \wedge \quad j\text{-inv.} \\ \supset & (f \leqslant j \quad \wedge \quad n\text{-inv.}[j/n]) \quad \vee \quad (i \leqslant f \quad \wedge \quad m\text{-inv.}[i/m]) \end{array}$$

Specifically,

$$\begin{array}{l} j < i \;\wedge\; \forall p(1 \leqslant p < i \;\supset\; A[p] \leqslant r) \\ \qquad \wedge \;\; \forall q(j < q \leqslant N \;\supset\; r \leqslant A[q]) \\ \supset \;(f \leqslant j \quad \wedge \quad \forall p, q(1 \leqslant p \leqslant j < q \leqslant N \;\supset\; A[p] \leqslant A[q])) \quad \vee \\ \qquad (i \leqslant f \quad \wedge \quad \forall p, q(1 \leqslant p < i \leqslant q \leqslant N \;\supset\; A[p] \leqslant A[q])) \end{array}$$

(Lemma 6)

Stage 3: Reduce the Middle Part (5/6)

The Destination of **go to**

- When the loops terminates, $j < f < i$

- This means that 'FOUND' is satisfied:

$$1 \leqslant f \leqslant N \quad \land \quad j < f < i \quad \land \quad i\text{-inv.} \quad \land \quad j\text{-inv.} \;\supset\; \text{FOUND}$$

Specifically,

$$\begin{aligned} 1 \leqslant f \leqslant N \quad \land \quad j < f < i \;\land\; & \forall p(1 \leqslant p < i \;\supset\; A[p] \leqslant r) \\ \land\; & \forall q(j < q \leqslant N \;\supset\; r \leqslant A[q]) \end{aligned}$$

$$\forall p, q(1 \leqslant p \leqslant f \leqslant q \leqslant N \;\supset\; A[p] \leqslant A[f] \leqslant A[q]) \qquad \text{(FOUND)}$$

Stage 3: Reduce the Middle Part (6/6)

- The Resulting Program:

  $r := A[f]; i := m; j := n$

  while $i \leqslant j$ **do**

  "*increase* i *and decrease* j"

  if $f \leqslant j$ **then** $n := j$

  else if $i \leqslant f$ **then** $m := i$

  else go to L

Stage 4: Increase i and Decrease j (1/4)

- Increase i

  while $A[i] < r$ **do** $i := i + 1$

- Check the i-inv.

$$A[i] < r \quad \wedge \quad i\text{-inv.} \;\supset\; i\text{-inv.}[i+1/i]$$

Specifically,

$$A[i] < r \quad \wedge \quad m \leqslant i \quad \wedge \quad \forall p(1 \leqslant p < i \;\supset\; A[p] \leqslant r)$$
$$\supset\; m \leqslant i+1 \quad \wedge \quad \forall p(1 \leqslant p < i+1 \;\supset\; A[p] \leqslant r) \qquad \text{(Lemma 8)}$$

Stage 4: Increase i and Decrease j (2/4)

- Decrease j

  while $r < A[j]$ **do** $j := j - 1$

- Check the j-inv.

$$r < A[j] \quad \wedge \quad j\text{-inv.} \ \supset \ j\text{-inv.}[j-1/j]$$

Specifically,

$$r < A[j] \quad \wedge \quad j \leqslant n \quad \wedge \quad \forall q(j < q \leqslant N \ \supset \ r \leqslant A[q])$$
$$\supset \ j - 1 \leqslant n \quad \wedge \quad \forall q(j-1 < q \leqslant N \ \supset \ r \leqslant A[q]) \qquad \text{(Lemma 9)}$$

Stage 4: Increase i and Decrease j (3/4)

- On termination of the loops,

$$A[j] \leqslant r \leqslant A[i]$$

- If i and j have not crossed over ($i \leqslant j$), $A[i]$ and $A[j]$ should be exchanged
- That means:

if $i \leqslant j$ **then**
"*exchange* $A[i]$ *and* $A[j]$"

Stage 4: Increase i and Decrease j (4/4)

- The Resulting Program:

  while $A[\mathrm{i}] < \mathrm{r}$ **do** $\mathrm{i} := \mathrm{i} + 1$

  while $\mathrm{r} < A[\mathrm{j}]$ **do** $\mathrm{j} := \mathrm{j} - 1$

  if $\mathrm{i} \leqslant \mathrm{j}$ **then**

  "*exchange* A[i] *and* A[j]"

Stage 5: Exchange $A[i]$ and $A[j]$ (1/3)

- The code for the exchange:

  $w := A[i]; A[i] := A[j]; A[j] := w$

- Let A' stands for the array A after exchange, then

$$A'[i] = A[j] \quad \wedge \quad A'[j] = A[i] \quad \wedge$$
$$\forall k(1 \leqslant k \leqslant N \quad \wedge \quad k \neq i \quad \wedge \quad k \neq j \quad \wedge \quad A'[k] = A[k])$$

Stage 5: Exchange $A[i]$ and $A[j]$ (2/3)

- Checking the i-inv.: $i \leqslant j \quad \wedge \quad$ i-inv. $\supset$ i-inv.$[A'/A]$ i.e:

$$m \leqslant i \leqslant j \quad \wedge \quad \forall p(1 \leqslant p < i \ \supset \ A[p] \leqslant r)$$
$$\supset \ \forall p(1 \leqslant p < i \ \supset \ A'[p] \leqslant r) \qquad \text{(Lemma 10)}$$

- Checking the j-inv.: $i \leqslant j \quad \wedge \quad$ j-inv. $\supset$ j-inv.$[A'/A]$ i.e:

$$m \leqslant j \leqslant n \quad \wedge \quad \forall q(j < q \leqslant N \ \supset \ r \leqslant A[q])$$
$$\supset \ \forall q(j < q \leqslant N \ \supset \ r \leqslant A'[q]) \qquad \text{(Lemma 11)}$$

Stage 5: Exchange $A[i]$ and $A[j]$ (3/3)

- Checking the m-inv.: $i \leqslant j \quad \wedge \quad m\text{-inv.} \supset m\text{-inv.}[A'/A]$ i.e:

$$m \leqslant i \leqslant j \quad \wedge \quad \forall p, q(1 \leqslant p < 1 \leqslant q \leqslant N \supset A[p] \leqslant A[q])$$
$$\supset \forall p, q(1 \leqslant p < 1 \leqslant q \leqslant N \supset A'[p] \leqslant A'[q]) \qquad \text{(Lemma 12)}$$

- Checking the n-inv.: $i \leqslant j \quad \wedge \quad n\text{-inv.} \supset n\text{-inv.}[A'/A]$ i.e:

$$i \leqslant j \leqslant n \quad \wedge \quad \forall p, q(1 \leqslant p \leqslant N < q \leqslant N \supset A[p] \leqslant A[q])$$
$$\supset \forall p, q(1 \leqslant p \leqslant N < q \leqslant N \supset A'[p] \leqslant A'[q]) \qquad \text{(Lemma 13)}$$

The Whole Program

$m := 1; n := N$

while $m < n$ **do**

  $r := A[f]; i := m; j := n$

  while $i \leqslant j$ **do**

    while $A[i] < r$ **do** $i := i + 1$

    while $r < A[j]$ **do** $j := j - 1$

    if $i \leqslant j$ **then**

      $w := A[i]; A[i] := A[j]; A[j] := w$

  if $f \leqslant j$ **then** $n := j$

  else if $i \leqslant f$ **then** $m := i$

  else go to L

$L :$

Summary

- Axiomatic system is constructed
  - The relation between the precondition the postcondition of a program fragments can be exactly constructed
  - The program proof can be constructed using the axioms and rules which prescribes these relations
- Proving during Coding
  - Observe the nature of data
  - Formulate invariants for the data (or variables)
  - Coding (altering variables)
  - Proving that the invariants are preserved
  - Reconsidering the earlier decisions if the assertions cannot be proved

References and ...

- References
  - C. A. R. Hoare, "An Axiomatic Basis for Computer Programming,", *CACM*, 12(10), 1969.
  - C. A. R. Hoare, "Proof of a Program: FIND,", *CACM*, 14(1), 1971.
- Further References
  - Axiomatic Semantics Section of Various Programming Language Textbook
  - H. R. Nielson and F. Nielson, *Semantics with Applications: A Formal Introduction*, John Wiley & Sons, 1992.
  - D. Gries, *The Science of Programming*, Springer, 1981.