FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

Basic Algorithm Analysis (3)

Correctness of Algorithms: Loop Invariant

DAA Term 2 2023/2024

Program Correctness

- What is a **correct program**?
- How do we know that our program is correct?
  - Use program verification tools
  - Formal/mathematical proof

- A program is said to be **correct** if it produces the correct output for every possible input.

- Proof of the correctness of a program:
  - Show that the correct answer is obtained if the program terminates (**partial correctness**)

  - Show that the program always terminates

Program Correctness

- For iterative algorithm: **Loop Invariant (partial correctness)**
- For recursive algorithm
  - Will be discussed later in “Recurrence” section

Both are based on the concept of mathematical induction.

Loop Invariant

- **Hoare's Triple: $\boldsymbol{p\{S\}q}$**
  - p is initial assertion/precondition, q is final assertion/postcondition, S is program segment
  - S is partially correct if
    - whenever $\boldsymbol{p}$ **is true** for the **input value of $\boldsymbol{S}$** and $\boldsymbol{S}$ **terminates**,
    - then q is true for the **output values of *S*.**

Loop Invariant

- An assertion/predicate that remains true each time the loop body S is executed.

- It provides **a link** between the **initial** and **final states**, connected through all the intermediate states.

Loop Invariant

- We must show three things about a Loop Invariant:
  - **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 —*usually along with the reason that the loop terminated*— gives us a <u>useful property</u> that helps show that the algorithm is correct.

Example 1

```
// precondition: n>0
int i = 0;
while(i<n){
    i++;
}
// postcondition: i=n
```

Which statement is TRUE **before entering the loop**, **during the iteration**, and **after the loop terminates**?

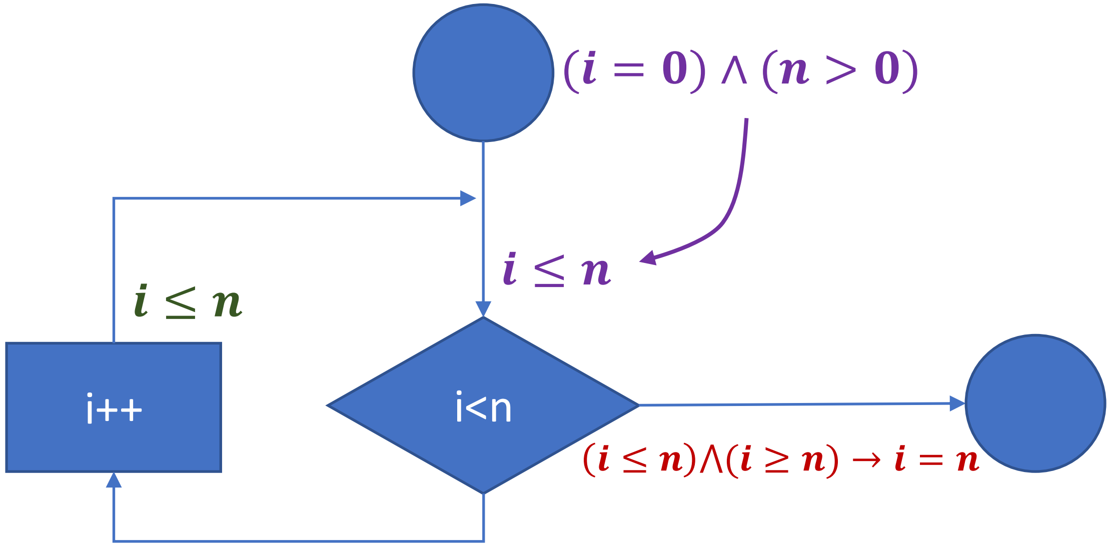
If objective of this procedure is to increments i from 0 to n, where n is positive integer

Which **invariant** is correct? Why and why not?

$$i = 0$$
$$i < n$$
$$i \leq n$$
$$n > 0$$

Example 1 (Cont'd)

```
// pre: n>0
int i = 0;
while(i<n){
    i++;
}
// post: i=n
```



Initialization : $i = 0$ and $n > 0$ (**$i \leq n$ holds**)

Maintenance : the value of i is incremented during each iteration if $i < n$, so **before each iteration, it is true that $i \leq n$**

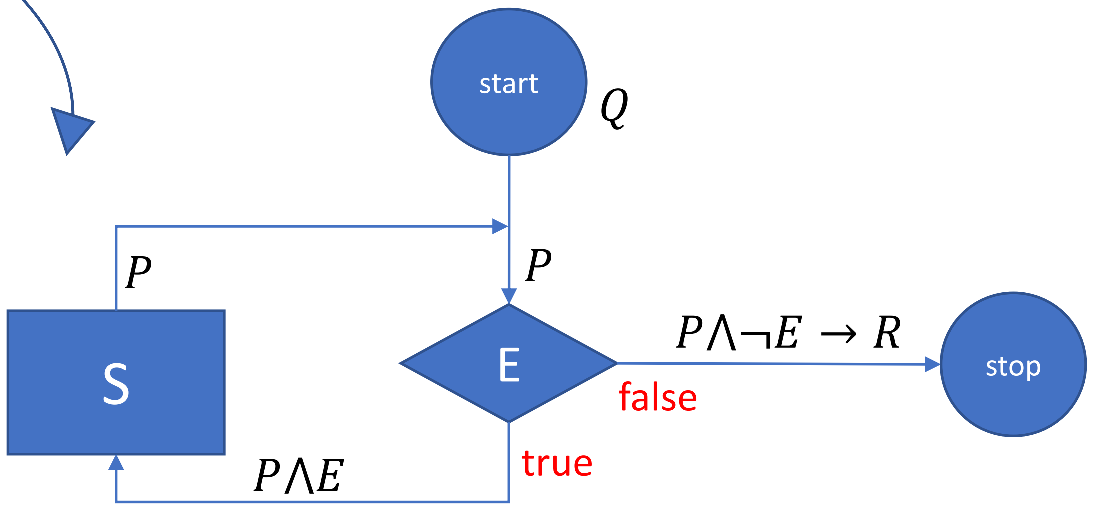
Termination : iteration stops when the guard is false ($i \geq n$), together with the invariant ($i \leq n$), **it implies $i = n$ (the postcondition)**

The correct invariant is **$i \leq n$ (before each iteration in while loop).**

Loop Invariant Formula

- While Loop:

```
1. while E:
2.      do S
3.      end
```



Q: precondition/condition in initialization
P: Invariant
R: post condition
S: Statements to be executed/program segment
E: Guard/Loop condition

Example 2

- Determine a suitable invariant to prove that the following program segment is correct. Show that the invariant holds in initialization, maintenance, and termination.

```
power = 1;
i = 1;
while i <= n
    power = power * x;
    i = i + 1;
```

Precondition?
Postcondition?

- Objective of this procedure is to compute the nth power of a positive real number x (where n is a positive integer)

Example 2 (Cont'd)

```
1. power = 1;
2. i = 1;
3. while i <= n
4.     power = power * x;
5.     i = i + 1;
```

Initialization : $i = 1, power = x^{1-1} = x^0 = 1$, thus loop invariant holds.

Maintenance : line 4-5 update $power$ and increment i. We need to show that if loop invariant holds before an iteration, it remain true before the next iteration.

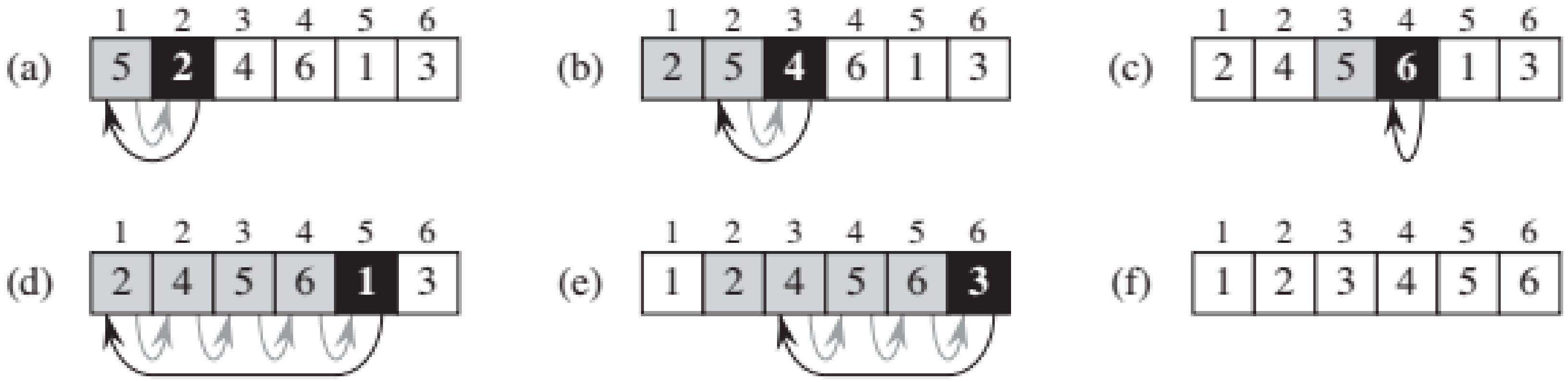
Termination : when the loop terminates, show that $loop\ invariant \land \neg loop\ guard \rightarrow postcondition$

Loop Invariant for Insertion Sort

INSERTION-SORT(A)

```
1  for j = 2 to A.length
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
6          A[i + 1] = A[i]
7          i = i - 1
8      A[i + 1] = key
```

At the start of each iteration of the **outer for loop** —*the loop indexed by j (line 1 to 8)*— the subarray $A[1 \ldots j-1]$ consists of the elements originally in $A[1 \ldots j-1]$ but in sorted order



Loop Invariant for Insertion Sort: Proof (1)

At the start of each iteration of the **outer for loop** —*the loop indexed by j (line 1 to 8)*— the subarray $A[1 \dots j-1]$ consists of the elements originally in $A[1 \dots j-1]$ but in sorted order

- **Initialization**
  - Before the first loop, we have $j = 2$, the current subarray refers to $A[1]$.
  - It is true that the subarray $A[1]$ is sorted, hence loop invariant is true during initialization
- **Maintenance**
  - What does the body loop do? See the pseudocode line 2 to 8.
  - Show that after each iteration, the process works such that the subarray $A[1 \dots j-1]$ consists of ordered version of elements originally in $A[1 \dots j-1]$

Loop Invariant for Insertion Sort: Proof (2)

At the start of each iteration of the **outer for loop** —*the loop indexed by j (line 1 to 8)*— the subarray $A[1 \ldots j-1]$ consists of the elements originally in $A[1 \ldots j-1]$ but in sorted order

- **Termination**
  - The loop terminates when $j > A.length$
  - With the current value of $j = A.length + 1$, what can we conclude about the subarray $A[1 \ldots j-1]$?

Finding Invariant

- Constructing I by weakening P (I: loop invariant, P: post condition)
  - By weakening P, I will holds more states than P does. The loop should then terminate when the particular instance of I corresponds to the situation where P also holds: this will influence the choice of guard.
  - Weakening P can be done by the following options.
    - Replacing a constant with a variable
      - Revisit previous examples: increment, power, insertion sort
    - Deleting a conjunct

Finding Invariant: Deleting a conjunct

- If a postcondition consists of a number of conjuncts, then it can be weakened by deleting one (or several) of its conjuncts.
- The resulting predicate will be true in more states than the postcondition, and might be suitable as a loop invariant.
- The loop guard in this case will be the negation of the deleted conjunct. So the negation of the guard and the remaining conjuncts together imply the postcondition.

- *From lecturer slide by Bpk. L. Yohanes Stefanus*

Finding Invariant: Deleting a conjunct

- The integer square root r of a natural number n is the greatest integer whose square is no more than n. So we have

  $$P = r^2 <= n \ \& \ n < (r + 1)^2$$

- Deleting the second conjunct leaves $r^2 <= n$. This will do as an invariant of a loop to achieve the postcondition P. It is true when $r = 0$, so an initial state for the loop can easily be established. The loop guard will be the negation of the deleted conjunct: $E = (r + 1)^2 <= n$.
- The loop body simply increments r .

- *From lecturer slide by Bpk. L. Yohanes Stefanus*

Finding Invariant: Deleting a conjunct

- Thus the complete loop to compute integer square root is:

```
r := 0;
WHILE (r + 1)² <= n
DO r := r + 1
END
```

- *From lecturer slide by Bpk. L. Yohanes Stefanus*

Exercise

- Find the suitable loop invariant for the following code and show that the invariant holds in initialization, maintenance, and termination.

```
max ← a_1
for i = 2 to n {
    if (a_i > max)
        max ← a_i
}
```

```
i = 1, j = 1
while (both lists are nonempty) {
    if (a_i ≤ b_j) append a_i to output list and increment i
    else           append b_j to output list and increment j
}
append remainder of nonempty list to output list
```

References

- Lecturer Slides by Bapak L. Yohanes Stefanus
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.
- https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/02AlgorithmAnalysis.pdf