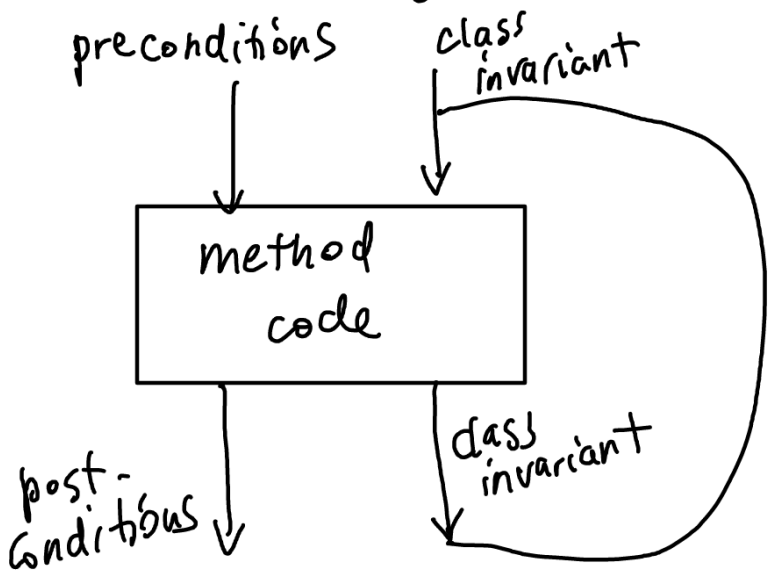
CS 2110: Loop Invariants

```
/** Returns: maximum value in a */
int max(int[] a) {
    int m=0;
    int i=0;
    while (i< a.length) {
        if (a[i] >= m) { m = a[i];}
    }
    return m;
}
```

A: Correct
B: Correct but inefficient
C: Incorrect
D: Incorrect and inefficient

testing

Can we reason rigorously about correctness of code?



challenge: loops

```
while (guard) {
    body
}
```

Loop Invariant

assumed true at beginning of every iteration
must hold after executing body

- every correct loop has an invariant.

Example: binary search

```
/** Returns: i where a[i] == k.
    Requires: k is in a, a is sorted (in ascending order) */
int search(int[] a, int k) {
    int l = 0, r = a.length - 1;          // why - 1
    while (l < r) {
        int m = (l + r) / 2;
        if (k <= a[m]) r = m;             // why not <
        else l = m + 1;                   // why + 1
    }
    return l;
}
```

Ex/ k = 7 a = { 1, 3, 7, 11 }

l, m, r

l2, r2

l3, r3, m3

```
/** Returns: i where a[i] == k.
    Requires: k is in a, a is sorted */
int search(int[] a, int k) {
  int l=0, r=a.length-1;
  while (l<r) {
    int m = (l+r)/2;
    if (k<=a[m]) r=m;
    else l=m+1;
  }
  return l;
}
```

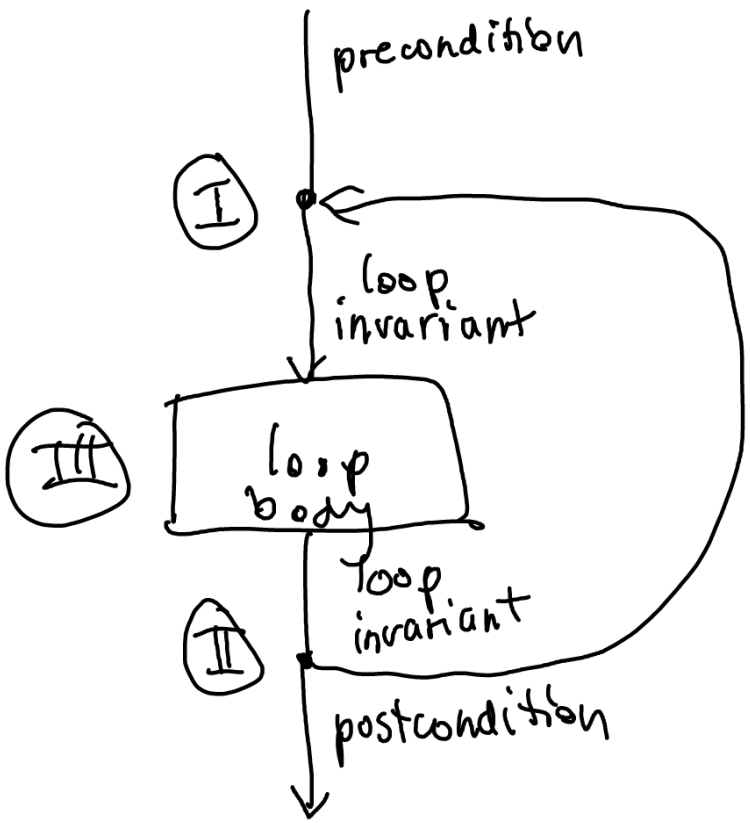
Loop precondition:
- a is sorted
- l = 0
- r = a.length - 1

Loop postcondition: k = a[l]

Binary search loop invariant
1. a sorted
2. l ≤ r
3. k ∈ a[l..r]

Notation: $i..j = i, i+1, \ldots, j$

$a[i..j] = a[i], \ldots, a[j]$



I Establishment

precondition implies loop invariant. If precondition holds, loop invariant does.

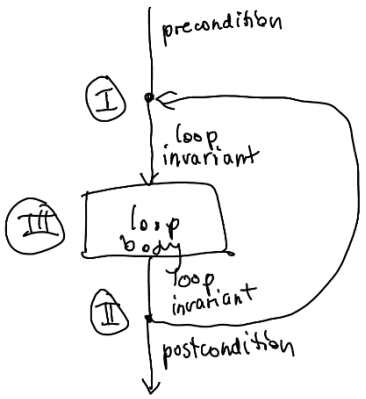
II Postcondition

invariant implies postcondition when the guard is false

III Preservation

If invariant holds and guard is true, executing body results in loop invariant being true.

```
/** Returns: i where a[i] == k.
    Requires: k is in a, a is sorted */
int search(int[] a, int k) {
    int l=0, r=a.length-1;
    while (l<r) {
        int m = (l+r)/2;
        if (k<=a[m]) r=m;
        else l=m+1;
    }
    return l;
}
```



loop precondition: $k \in a[l..a.length-1]$
- a is sorted
- $l = 0$
- $r = a.length - 1$

loop postcondition: $k = a[l]$

Binary search loop invariant
1. a sorted
2. $l \le r$
3. $k \in a[l..r]$

I Establishment

1. precondition $\Rightarrow$ a sorted ✓
2. $a.length \ge 1 \Rightarrow l \le r$ ✓
3. precondition ✓

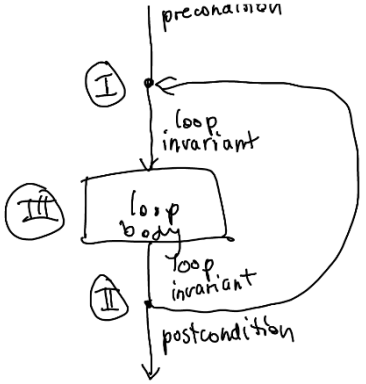
II Postcondition $k = a[l]$?

If loop guard is false, $l \ge r$

Inv. 2: $l \le r$, so $l = r$

Inv 3: $k \in a[l..r] = a[l]$

```
/** Returns: i where a[i]==k.
    Requires: k is in a, a is sorted */
int search(int[] a, int k) {
  int l=0, r=a.length-1;
  while (l<r) {
    int m = (l+r)/2;
    if (k<=a[m]) r=m;
    else l=m+1;
  }
  return l;
}
```



loop precondition: $k \in a[0..a.length-1]$
- a is sorted
- $l = 0$
- $r = a.length - 1$

loop postcondition: $k = a[l]$

Binary search loop invariant
1. a sorted $0 \le l$
2. $l \le r$
3. $k \in a[l..r]$

$m = \lfloor \frac{l+r}{2} \rfloor$ and $l < r$ so $l \le m < r$

III. Preservation

Assume loop invariant holds on a, l, r, k
Show it holds on values after body a', l', r', k'
$a' = a, k' = k$

1. a sorted? ✓

2, 3. After body, need $l' \le r'$ and $k \in a[l'..r']$

(L) $k \le a[m]$. $l' = l$ and $r' = \lfloor \frac{r+l}{2} \rfloor = m$
2. $l \le m$, so $l' \le r'$
3. a is sorted, so k must occur at or before m.
$k \in a[l..m] = a[l'..r']$

(R) $k > a[m]$. $l' = m+1$, $r' = r$,
2. $m < r$, so $m+1 \le r$, so $l' \le r'$
3. a is sorted $\Rightarrow k \notin a[l..m]$
$k \in a[m+1..r] = a[l'..r']$

What did we prove? (k=a[l])

If the loop exits, post condition (code) holds.

Partial Correctness: we assume ∧ terminates

Total Correctness: partial correctness + loop terminates

How to find loop invariants?

invariant too weak? ← too strong?

Usual mistake

Postcondition step fails (II)
or Preservation fails (III)

Establishment (I) fails.
or Preservation (III) fails.

Total correctness = Partial correctness + termination.

Key idea: prove some non negative quantity DF
= "decrementing function" (aka loop variant)

(IV) Show: DF decreases on every loop iteration,
assuming loop invariant & guard holds.
& can never decrease below 0

For binary search: DF = r − l