# Poll Everywhere

PollEv.com/2110fa25 text 2110fa25 to 22333

On the left is a *buggy* `med3()` definition. Which JUnit assertion will detect the bug?

```
/** Returns median of `a`, `b`, `c`. */
static int med3(int a, int b, int c) {
    if (a >= b) {
        return Math.max(b,c);
    } else if (a >= c) { // a < b
        return a;
    } else { // a is smallest
        return Math.min(b,c);
    }
}
```

(A) `assertEquals(2,med3(1,2,3))`

(B) `assertEquals(2,med3(2,3,1))`

(C) `assertEquals(2,med3(2,1,3))`

(D) `assertEquals(2,med3(3,1,2))`


# Lecture 4: Loop Invariants

CS 2110

September 4, 2025

# Today's Learning Outcomes

20. Describe the loop invariant of an iterative method involving an array and visualize it using a diagram.

21. Use an array diagram to develop an iterative method.

22. Write precise specifications for methods involving arrays that use range notation.

# Announcements

**Assignment 1 due yesterday**

- Grading now, should be completed by Monday

**Assignment 2 released, due next Wednesday**

**Check out the new "Grades" tab on the course website**

- Let us know if you see any issues
- Remember to choose your grade calculation by Monday!

# Announcements

**Support Resources:** (full list on website)

- Academic Excellence Workshop (AEW) Sections
- Engineering Tutors-on-Call program
- Office Hours, Ed Discussion, etc.

Please reach out if there's anything we can do to help!

# Loop Anatomy

```
int sum = 0;
for (int i=0; i<a.length; i++) {
    sum += a[i];
}
```

Loop body

Increment: loop variable(s)

Initialization:

declare + initialize loop variable(s)

Loop guard:

boolean expression

true ⇒ run loop body

false ⇒ "fall through" loop

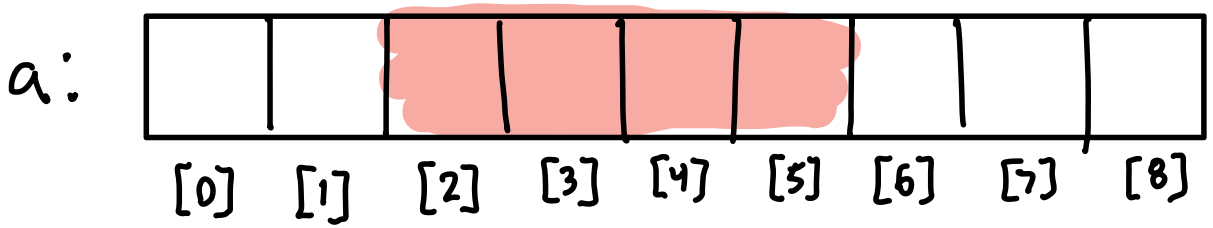
# while Loops

```
int sum = 0;
for (int i=0; i<a.length; i++) {
    sum += a[i];
}
```

```
int sum = 0;
int i=0;
while (i<a.length) {
    sum += a[i];
    i++;
}
```

# Range Notation

A range of an array is a contiguous subset of entries.




$a[2..5] = a(1..6) = a[2..6)$

include endpoints

exclude endpoints

Special cases:

from start: $a[0..i] = a[0..i]$, $a(0..i] = a(0..i]$

from end: $a[i..] = a[i..a.length-1]$

⋮

$a[i..j]$ empty when $i > j$

# Poll Everywhere

PollEv.com/2110fa25 text `2110fa25` to `22333`

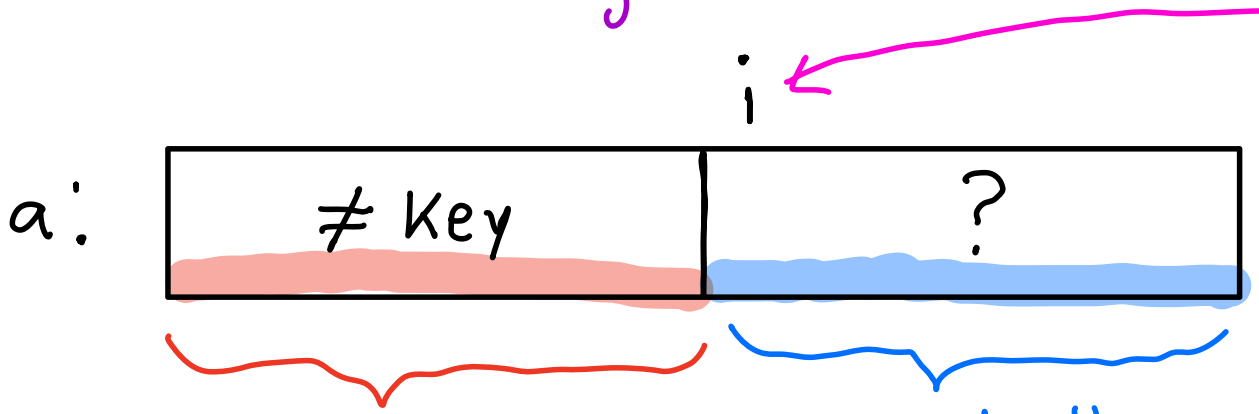
How many elements belong to the range `a(1..6]` ?

2 , 3, 4, 5, 6

| | |
|---|---|
| 4 | (A) |
| 5 | (B) |
| 6 | (C) |
| 7 | (D) |

# Range Properties and Array Diagrams

Array diagram: visualizes ranges properties of array




Super important:

i written to one side of boundary, not above boundary

every entry in a[..i) is ≠ key

we don't have any info about a[i..]

# Writing "Loopy" Code

```
/** Returns # of occurrences of `key` among the elements in array `a`. */
static int frequencyOf(int key, int[] a) { ... }
```

Pre
(before loop)

a:


Post
(after loop)

a:


At the start
of iteration i

a:

# (Loop) Invariants

An invariant is an assertion about code that will be true at multiple, pre-determined points of its execution.

A loop invariant describes relationship between loop + other local vars that is true every time loop guard is evaluated


Loop invariant: count = # of occurrences of key in a[..i)

# Developing a Loop: Initialization

We must initialize the local variables to make the loop invariant true the first time we check the loop guard.

```
/** Returns the # of occurrences of `key` among the elements in array `a`.*/
static int frequencyOf(int key, int[] a) {
    int i = 0;    // i = next index of 'a' to check   (start by checking a[0])
    int count = 0;
    /* Loop invariant: `count` = # of occurrences of `key` in `a[..i)` */
    while ( ... ) { ... }
}
```

a[..0) is empty (contains no 'key's)

# Poll Everywhere

PollEv.com/2110fa25 text `2110fa25` to `22333`

Which of these is true *immediately after* we fall through the loop body?

| | | |
|---|---|---|
| **loop invariant** is `true` | **loop guard** is `true` | **(A)** |
| **loop invariant** is `true` | **loop guard** is `false` | **(B)** |
| **loop invariant** is `false` | **loop guard** is `true` | **(C)** |
| **loop invariant** is `false` | **loop guard** is `false` | **(D)** |

# Developing a Loop: Guard

Choose condition that becomes false once we're done with the loop's work.

Loop invariant true when we exit loop, use this to determine return value

```
/** Returns the # of occurrences of `key` among the elements in array `a`.*/
static int frequencyOf(int key, int[] a) {
    int i = 0; // next index of `a` to check
    int count = 0;
    /* Loop invariant: `count` = # of occurrences of `key` in `a[..i)` */
    while ( i < a.length ) { ... }
    return count;
}
```

done when i == a.length

continue looping while i < a.length

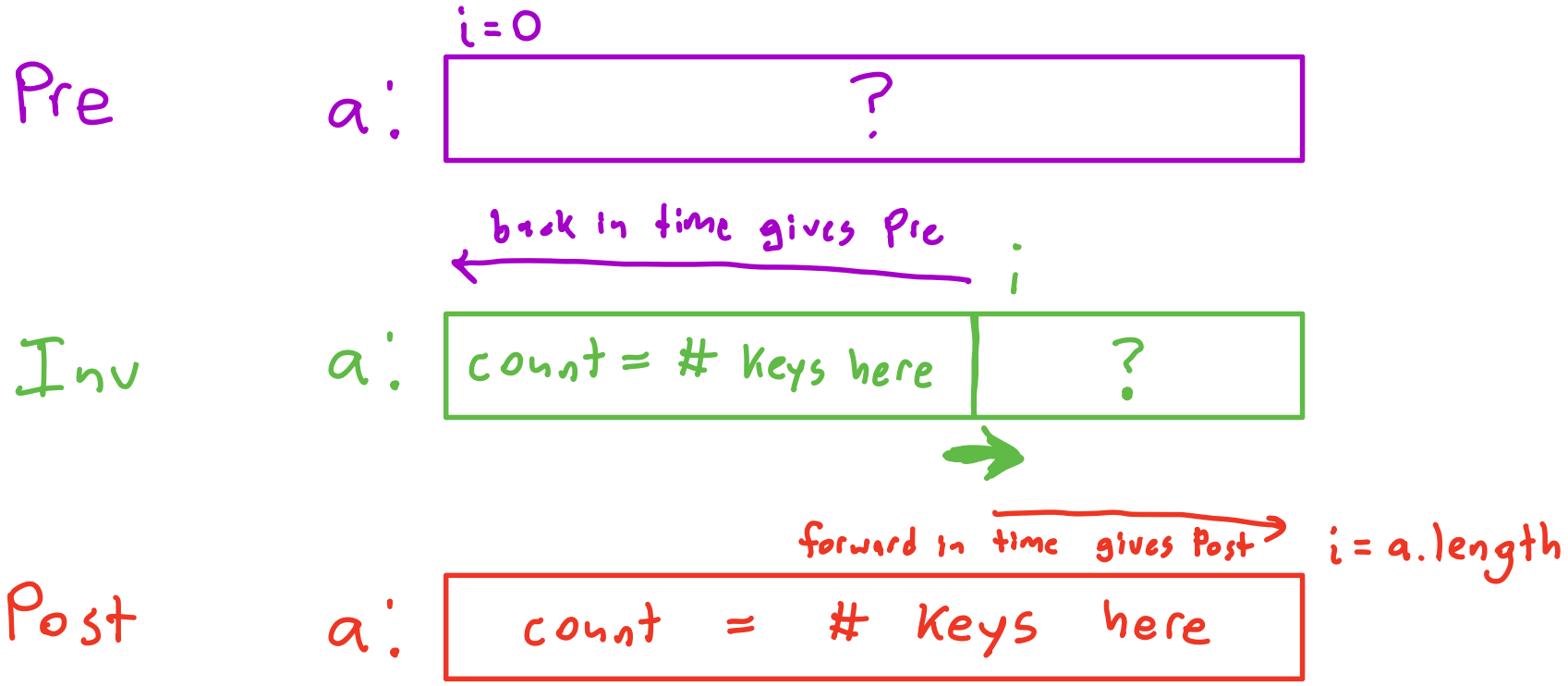
a[.. a.length) is all of a

# Developing a Loop: Body

In each iteration
- make progress toward loop's goal
- restore loop invariant

```
int i = 0;  int count = 0;
/* Loop invariant: `count` = # of occurrences of `key` in `a[..i)` */
while (i < a.length) {
    if (a[i] == key) {
        count++;
    }
    i++;    // progress
}
```

# Back to the Array Diagrams



# Example 2: `argmin()`

```
/** Returns an *index* of the minimum element in `a`.
   * Requires that `a.length > 0`. */
static int argmin(double[] a) { ... }
```

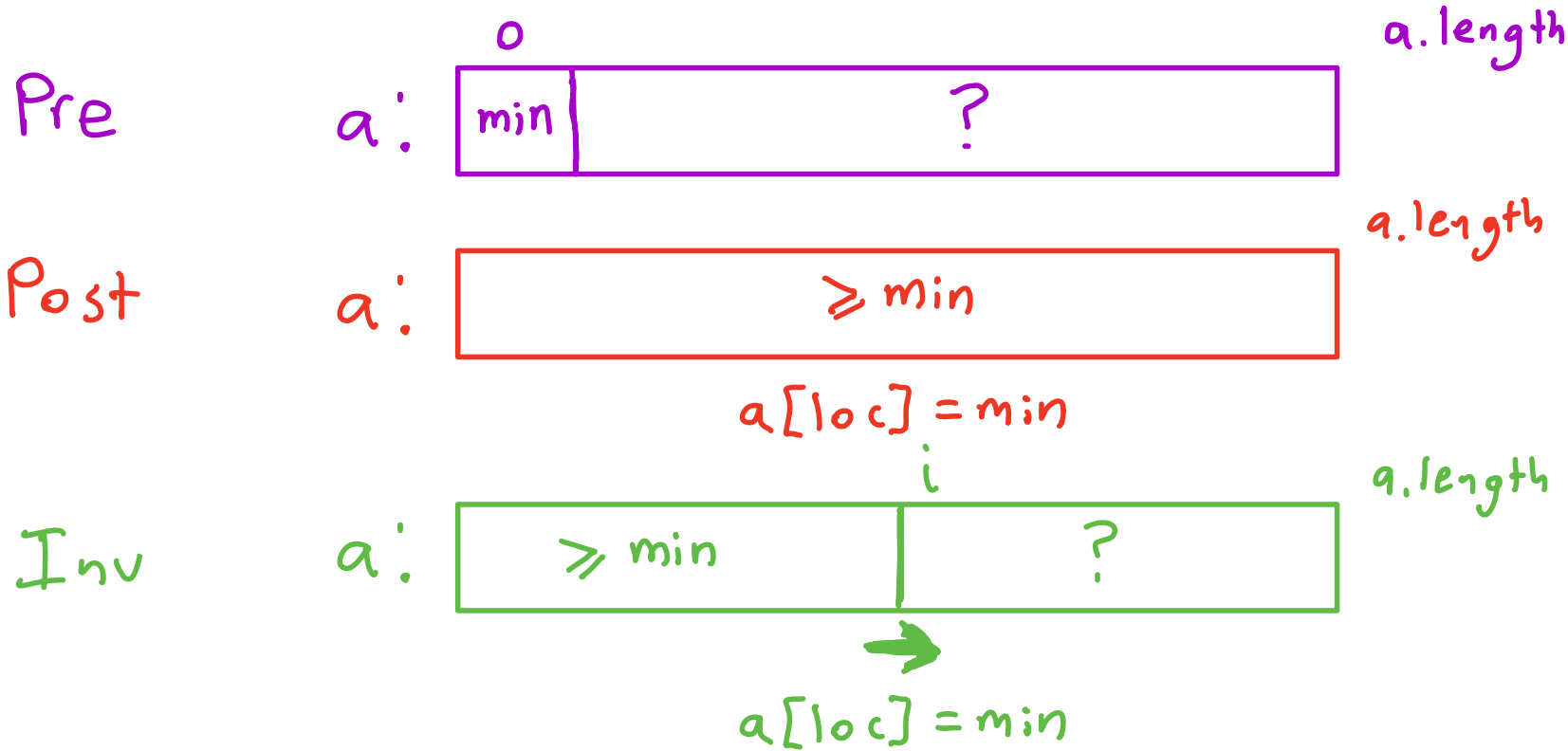
argmin(new double[]{1.0, 3.5, 4.2, 0.7, 6.3, 2.8}) = 3

0 1 2 3

What do we need to keep track of?

- which entries we've checked (i)
- the smallest element we've seen (min)
- where this smallest element is (loc)

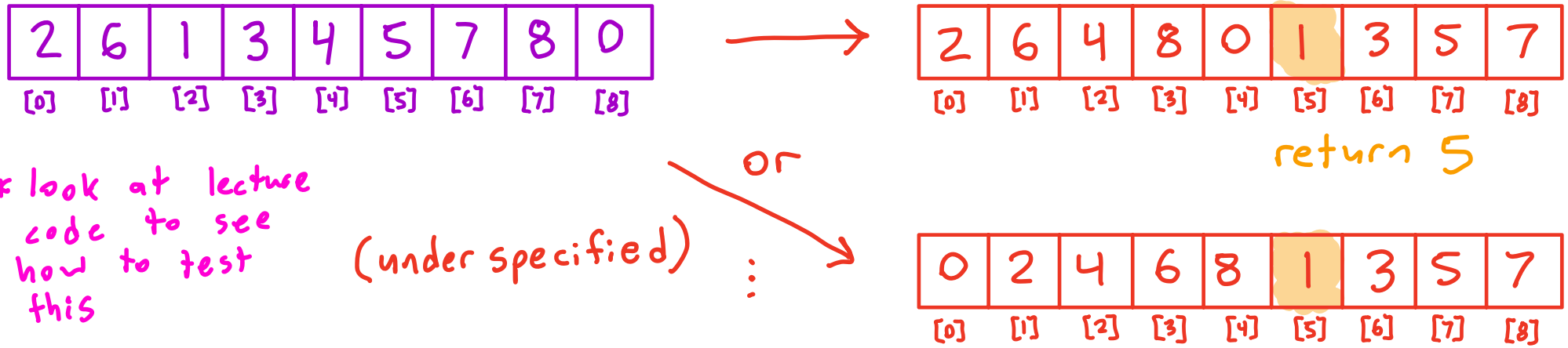
# `argmin()` Array Diagrams



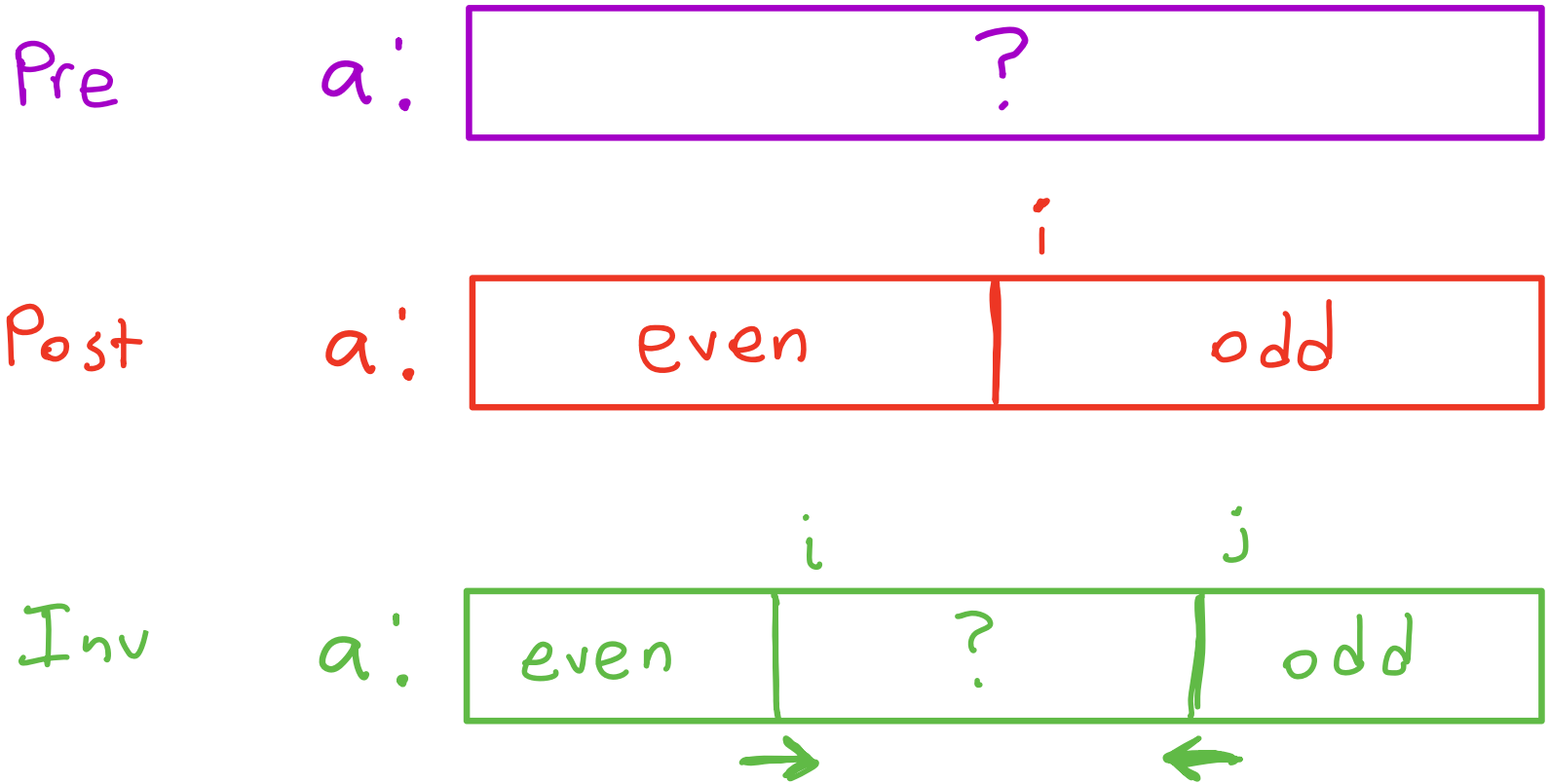
# Coding Demo: `argmin()`

# Example 3: `paritySplit()`

```
/** Rearranges `a` so that all even elements appear before all odd elements.
  * Returns the index of the first odd element, or `a.length` if all elements
  * are even. */
static int paritySplit(int[] a) { ... }
```



# paritySplit() Array Diagrams




* Note: we can also put i,j on other sides of boundaries.

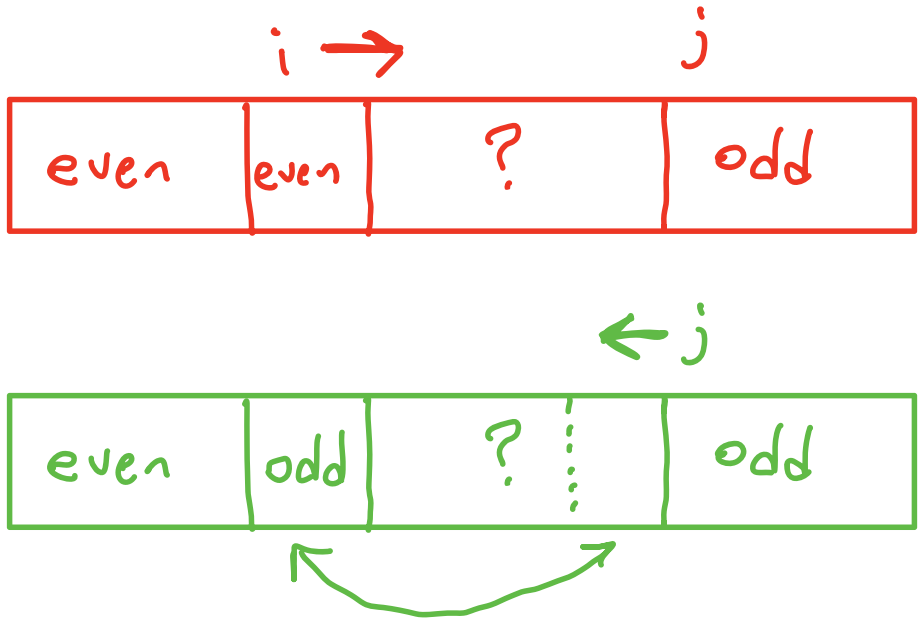
This will give another correct solution with different initialization, loop guard, body.

See lecture exercise 4.7.

# Coding Demo: `paritySplit()`

# paritySplit() Loop Body

```
while (i<j) {
    if (a[i] % 2 == 0) {
        i++;
    } else {
        // swap a[i], a[j-1]
        j--;
    }
}
```



# Review: Steps for Developing Loops

1. Identify the local variables.
2. Draw out the “Pre” and “Post” array diagrams.
3. Draw the “Inv” array diagram
    - Hybridizes "Pre" and "Post" diagrams
    - Incorporates all local variables
4. Write the loop invariant.
5. Slide the “Inv” → "Pre" to write initialization
6. Slide “Inv” → "Post" to write loop guard, post-loop code
7. Develop the loop body
    - Make progress toward the post-condition
    - Re-establishes the loop invariant