Testing of Programs

Black Box Test

- Design **black box tests** (as many as you feel are needed to be sure that testing is reasonably comprehensive) for a method whose signature, precondition and postconditions are all as follows.
  - **Precondition:**
    - **Signature:** public static boolean distinctEntries (int[] A)
    - A is an integer array
  - **Postcondition:**
    - The array A has not been changed
    - Value returned is true if the entries of A are distinct, and is false otherwise

Black Box Test

- Invalid Input :
  - when the input is not an integer array
  - **[“x”, “y”]**
  - **X**
  - **Null**
  - **…**
- Typical Cases:
  - values of array are distinct or not,…
  - [x, y, z]
  - [x, x, y, z]
  - [y, , z, x, x]
  - [y, z, x, x, w]
  - …
- Boundary Conditions:
  - when the array is empty, when two same values are at beginning or at the end, …
  - []
  - [null]
  - [x]
  - [x, x]
  - [y, x, x]
  - …

White Box

- Design any additional **white box tests** that you think are appropriate for the following method.

- Tests typically try to ensure that:
  - every statement in code is executed in one or more tests
  - each “if” and “else” branch of every conditional statement is tested
  - each loop is iterated zero, one, several, and as many times as possible (if these situations are feasible)
  - each exit condition causing a loop or function to terminate is executed all exception handling is tested

A = []
A = [x]
A = [x, y, …]
…

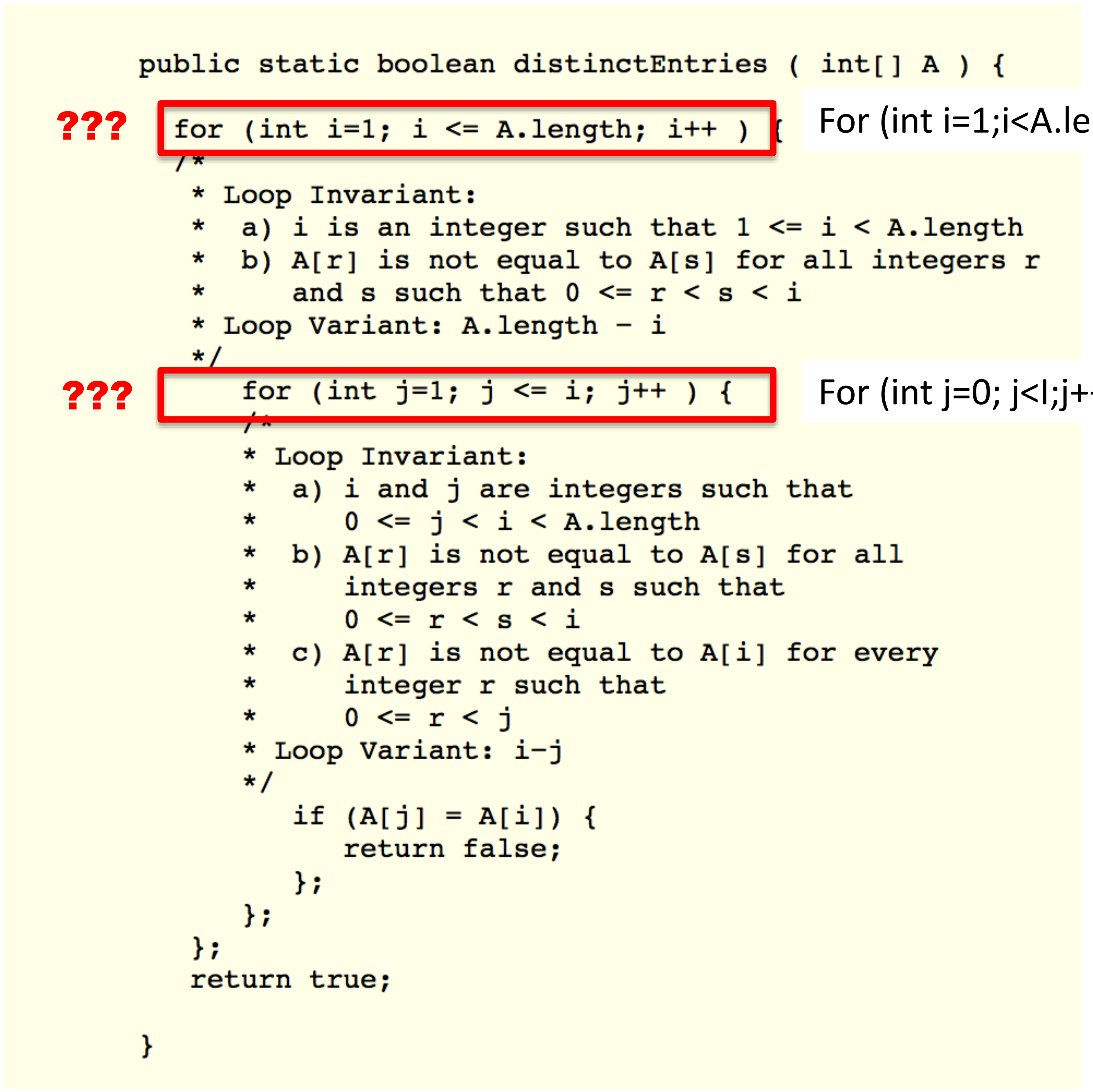
A = [x]
A = [x, y, …]
…

A = [x, y]
A = [x, x]
A= [y, x, x]
…

```
public static boolean distinctEntries ( int[] A ) {

  for (int i=1; i <= A.length; i++ ) {
  /*
   * Loop Invariant:
   *  a) i is an integer such that 1 <= i < A.length
   *  b) A[r] is not equal to A[s] for all integers r
   *     and s such that 0 <= r < s < i
   * Loop Variant: A.length - i
   */
      for (int j=1; j <= i; j++ ) {
      /*
      * Loop Invariant:
      *  a) i and j are integers such that
      *     0 <= j < i < A.length
      *  b) A[r] is not equal to A[s] for all
      *     integers r and s such that
      *     0 <= r < s < i
      *  c) A[r] is not equal to A[i] for every
      *     integer r such that
      *     0 <= r < j
      * Loop Variant: i-j
      */
         if (A[j] = A[i]) {
            return false;
         };
      };
   };
   return true;

}
```

Static Testing

- Perform **“static testing”** to try to find as many errors in the above code as you can. Correct any errors that you find.
  - **Two types:**
    Desk checking: read through code, look for errors
  - **Hand Executions: trace code execution on small inputs with known outputs by hand**



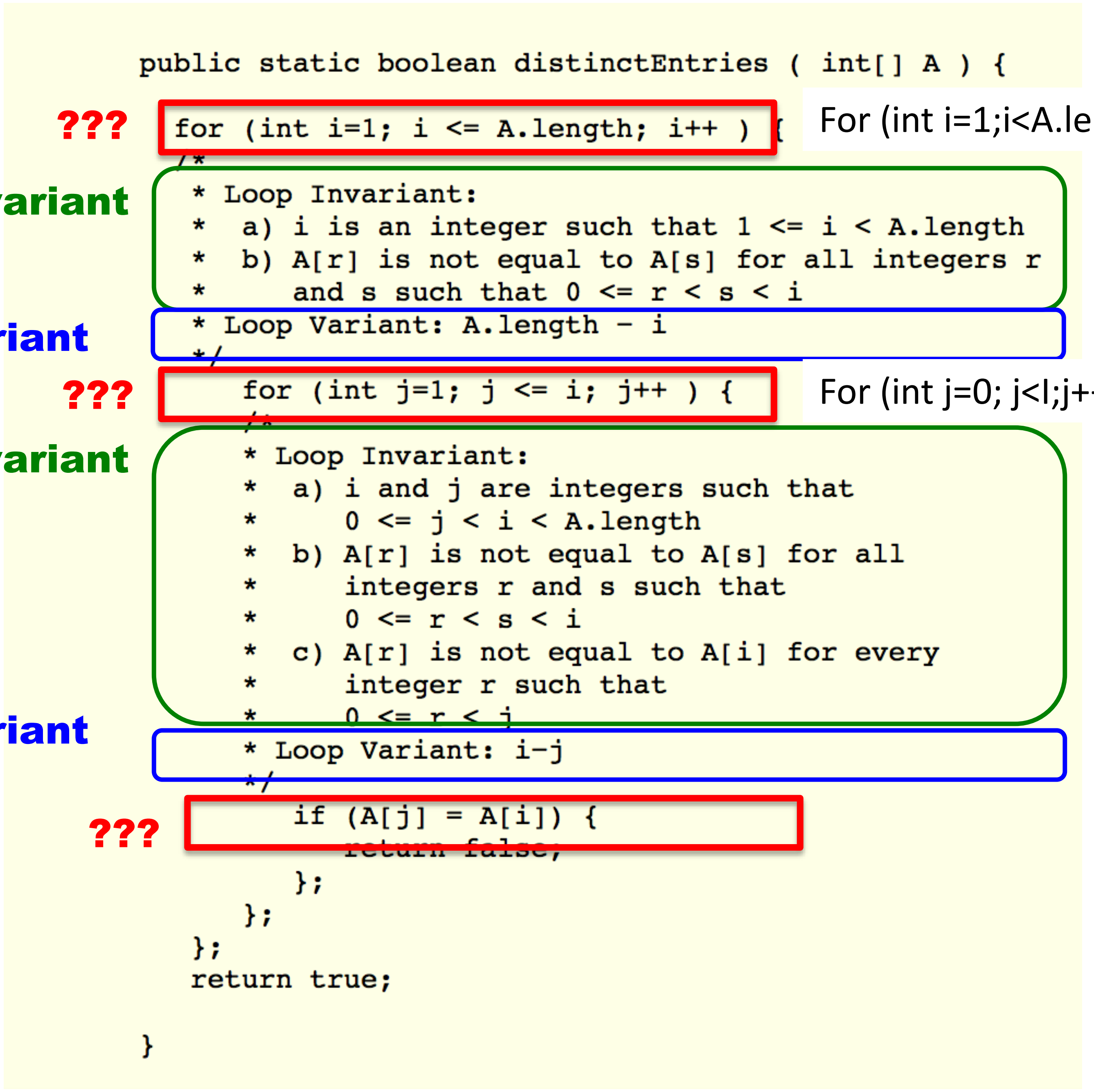
```
public static boolean distinctEntries ( int[] A ) {
  for (int i=1; i <= A.length; i++ ) {
   /*
    * Loop Invariant:
    *  a) i is an integer such that 1 <= i < A.length
    *  b) A[r] is not equal to A[s] for all integers r
    *     and s such that 0 <= r < s < i
    * Loop Variant: A.length - i
    */
      for (int j=1; j <= i; j++ ) {
      /*
       * Loop Invariant:
       *  a) i and j are integers such that
       *     0 <= j < i < A.length
       *  b) A[r] is not equal to A[s] for all
       *     integers r and s such that
       *     0 <= r < s < i
       *  c) A[r] is not equal to A[i] for every
       *     integer r such that
       *     0 <= r < j
       * Loop Variant: i-j
       */
         if (A[j] = A[i]) {
            return false;
         };
      };
   };
   return true;

}
```

???

For (int i=1;i<A.lenght;i++)

???

For (int j=0; j<I;j++)



```
public static boolean distinctEntries ( int[] A ) {
  for (int i=1; i <= A.length; i++ ) {
  /*
   * Loop Invariant:
   *  a) i is an integer such that 1 <= i < A.length
   *  b) A[r] is not equal to A[s] for all integers r
   *     and s such that 0 <= r < s < i
   * Loop Variant: A.length - i
   */
      for (int j=1; j <= i; j++ ) {
      /*
       * Loop Invariant:
       *  a) i and j are integers such that
       *     0 <= j < i < A.length
       *  b) A[r] is not equal to A[s] for all
       *     integers r and s such that
       *     0 <= r < s < i
       *  c) A[r] is not equal to A[i] for every
       *     integer r such that
       *     0 <= r < j
       * Loop Variant: i-j
       */
          if (A[j] = A[i]) {
            return false;
          };
      };
  };
  return true;

}
```

???

For (int i=1;i<A.lenght;i++)

Loop Invariant

Loop variant

???

For (int j=0; j<I;j++)

Loop Invariant

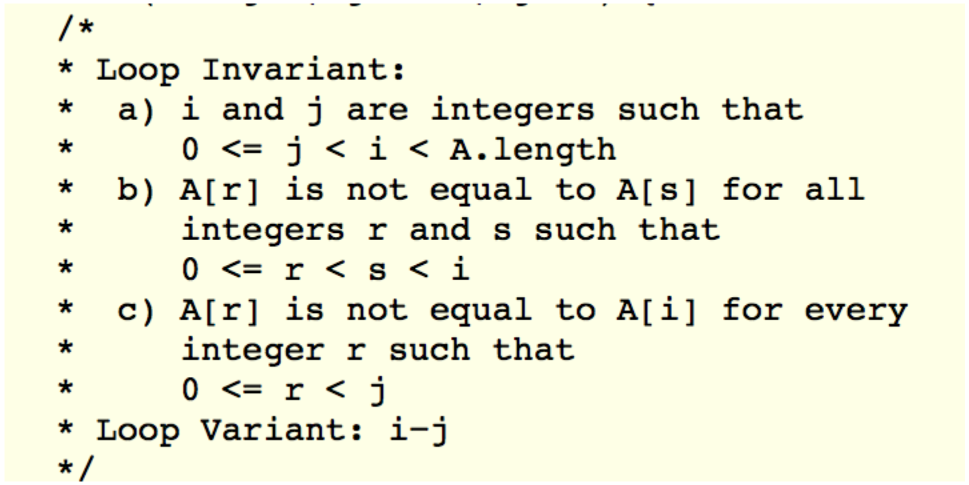
Loop variant

???

- Sketch a **proof of the correctness** of the ***corrected*** program you obtained from the above, using its documentation and the techniques for this that have now been introduced. Depending on how successful you were in answering the above question, you might be able to do this with few problems — or you might discover an error or two that your testing missed!

- For each loop
  - **First prove that the loop invariant is correct:**
    - Prove the base case **i=1**
    - Assume loop invariant holds for **i**, prove it holds for **i+1**
  - **Second prove that the loop is partially correct:**
    - we apply the loop invariant to prove that the algorithm's postconditions hold.

- Now, suppose you try to simplify the loop invariant for the inner loop by leaving out conditions (a) and (b). What goes wrong?



```
/*
 * Loop Invariant:
 *  a) i and j are integers such that
 *      0 <= j < i < A.length
 *  b) A[r] is not equal to A[s] for all
 *      integers r and s such that
 *      0 <= r < s < i
 *  c) A[r] is not equal to A[i] for every
 *      integer r such that
 *      0 <= r < j
 * Loop Variant: i-j
 */
```