

CLRS Chapter 2 Edition 4 Solutions

Cameron Beck

Updated June 15, 2023

Section 2.1

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       $A[i+1] = key$ 
```

- A **loop invariant** is a condition that is necessarily true immediately before and immediately after each iteration of a loop. (Note that this says nothing about its truth or falsity part way through an iteration.)
- Given an appropriate invariant, we can help prove the correctness of an algorithm.
- In the example above, we might say our loop invariant is that the sub-array $A[1..j-1]$ is **always** sorted.

To use a loop invariant, you must show three things

- **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:** The loop terminates, and when it terminates, the invariant, along with the reason that the loop terminated, gives us a useful property that helps show that the algorithm is correct.

Let's apply these principles to INSERTION-SORT

- **Initialization:** We start by considering $j = 2$. The sub-array $A[1 \dots 1]$ consists of only one element, $A[1]$, and therefore must be sorted.
- **Maintenance:** The body of the **for** loop works by moving the values in $A[j - 1]$, $A[j - 2]$, $A[j - 3]$, and so on by one position to the right until it finds the proper position for $A[j]$ (line 8). Thus, the sub-array $A[1 \dots j - 1]$ remains sorted. Incrementing j for the next iteration of the **for** loop preserves the loop invariant.
- **Termination:** Once the value of j exceeds $A.length$, the loop terminates. Substituting $A.length + 1$ for j in the wording of the loop variant yields that the sub-array $A[1 \dots n]$ consists of the elements originally in $A[1 \dots n]$, but in sorted order. Hence, the algorithm is correct.

A more formal treatment would require that we state and show a loop invariant for the **while** loop on line 5 as well.

Exercise 2.1-1

	1	2	3	4	5	6
$j = 2$	31	41	59	26	41	58
$j = 3$	31	41	59	26	41	58
$j = 4$	26	31	41	59	41	58
$j = 5$	26	31	41	41	59	58
$j = 6$	26	31	41	41	58	59

Table 1: Values of the array at each index after each iteration of the **for** loop in INSERTION-SORT given the sequence $\langle 31, 41, 59, 26, 41, 58 \rangle$.

Exercise 2.1-2

```

SUM-ARRAY( $A, n$ )
1   $sum = 0$ 
2  for  $i = 1$  to  $n$ 
3       $sum = sum + A[i]$ 
4  return  $sum$ 

```

A loop invariant for the given procedure SUM-ARRAY is that before each iteration of the loop, the variable *sum* **always** contains the sum of the sub-array from $A[1..i-1]$, with the following proof:

- **Initialization:** Since the sub-array will consist of zero elements, and *sum* is initialized to zero, we can suppose that it is reasonable to say that an array of size zero has a sum of zero, and thus the loop invariant is true prior to the first iteration.
- **Maintenance:** During each iteration, we add the value at the current index to *sum* and increment *i*. In doing so, we ensure that *sum* continues to equal the sum of elements in the sub-array $A[1..i-1]$.
- **Termination:** The loop terminates when $i > n$, meaning that for an array of $n = 5$, the loop would terminate when $i = 6$. According to our loop invariant, this would mean that we have added up all elements of the sub-array $A[1..5]$, which is equal to the original input array.

Exercise 2.1-3

Changing INSERTION-SORT to be monotonically decreasing instead of increasing requires only a slight adjustment to line 5:

```
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       $A[i+1] = key$ 
```

Exercise 2.1-4

```
LINEAR-SEARCH( $A, x$ )
1  for  $i = 1$  to  $A.length$ 
2      if  $A[i] == x$ 
3          return  $i$ 
4  return NIL
```

Loop invariant: at the start of each iteration of the **for** loop, the element x has not been found in the sub-array $A[1..i-1]$

- **Initialization:** At $i = 1$ we have yet to perform any comparisons, so the element x has yet to be found.
- **Maintenance:** During the i th iteration, we compare the value at the current index to x and immediately terminate the loop if they are equal.

This means that we are guaranteed to have not found x in the sub-array $A[1 \dots i - 1]$ by the time we start a new iteration.

- **Termination:** The loop terminates when $i > A.length$ or $A[i] == x$. In either case, the loop invariant holds true. If NIL is returned, we know that we scanned the entire array without finding x . If we return an index, we know that although the sub-array $A[1 \dots i - 1]$ did not contain x , $A[i]$ did, which satisfies the requirements of our loop invariant.

Exercise 2.1-5

Problem: Add two n -bit binary integers a and b stored in two n -element arrays $A[0 \dots n - 1]$ and $B[0 \dots n - 1]$, where each element is either 0 or 1, $a = \sum_{i=0}^{n-1} A[i] \cdot 2^i$, and $b = \sum_{i=0}^{n-1} B[i] \cdot 2^i$. The sum $c = a + b$ of the two integers should be stored in binary form in an $(n + 1)$ -element array $C[0 \dots n]$ where $c = \sum_{i=0}^n C[i] \cdot 2^i$. Write a procedure ADD-BINARY-INTEGERS that takes as input arrays A and B , along with the length n , and returns array C holding the sum.

ADD-BINARY-INTEGERS(A, B, n)

```

1  Let  $C$  be an array of length  $n + 1$  initialized with 0's
2   $carry = 0$ 
3  for  $i = n$  to 1
4       $digitSum = A[i] + B[i] + carry$ 
5       $C[i + 1] = digitSum \pmod{2}$ 
6      if  $digitSum > 1$ 
7           $carry = 1$ 
8      else
9           $carry = 0$ 
10  $C[1] = carry$ 
11 return  $C$ 
```

Section 2.2

Exercise 2.2-1

Exercise 2.2-2

Exercise 2.2-3

Problems

Problem 2-1