C. L. R. S' Introduction to Algorithms - A partial solutions manual

George Lydakis

May 26, 2025

Contents

Part I

Foundations

Chapter 2

Getting Started

2.1 Insertion sort

Exercise 1
Using Figure 2.2 as a model, illustrate the operation of INSERTION-SORT on an array initially containing the sequence $\langle 31, 41, 59, 26, 41, 58\rangle$.

Solution.
We show the steps of the algorithm below, underlining the element being processed in each iteration. The element is shown as already positioned in its correct place, implying that the elements that follow it (up to and including position i for the i-th iteration) have been shifted rightwards:

$$\langle \underline{31}, 41, 59, 26, 41, 58\rangle \to \langle 31, \underline{41}, 59, 26, 41, 58\rangle \to \langle 31, 41, \underline{59}, 26, 41, 58\rangle \to$$
$$\langle \underline{26}, 31, 41, 59, 41, 58\rangle \to \langle 26, 31, 41, \underline{41}, 59, 58\rangle \to \langle 26, 31, 41, 41, \underline{58}, 59\rangle$$

Exercise 2
Consider the procedure SUM-ARRAY:

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

It computes the sum of the n numbers in aray $A[1 : n]$. State a loop invariant for this procedure, and use its initialization, maintenance, and termination properties to show that the SUM-ARRAY procedure returns the sum of the numbers in $A[1 : n]$.

Solution.
The loop invariant is: at the start of the i-th iteration, the variable sum contains precisely the sum of A's elements at indices $1, 2, \ldots, i-1$.

- **Initialization**: At the start of iteration 1, line 1 has caused sum to contain 0, which equals the sum of A's elements for the empty collection of indices.
- **Maintenance**: Suppose the property holds before iteration i, which means $sum = \sum_{k=1}^{i-1} A[k]$. Then, the execution of iteration i means that line 3 now causes sum to be incremented by $A[i]$, and thus $\sum_{k=1}^{i} A[k]$. But this means precisely that the invariant is true before iteration $i+1$ as well.
- **Termination**: The loop trivially terminates due to running for a constant number of iterations n. Furthermore, we can think of its end as the start of an "imaginary" iteration $n+1$. Then, due to the invariant, it will hold that $sum = \sum_{k=1}^{n} A[n]$, which is the sum of all elements in the array.

Exercise 3
Rewrite the INSERTION-SORT procedure to sort into monotonically decreasing instead of monotonically increasing order.

Solution.
All that is needed here is for the condition of line 5 to be changed to "$j > 0$ and $A[j] < key$". It is then very easy to see that the loop invariant only changes in terms of what "sorted order" means, and thus the remaining proof of correctness would be the same, except that now we show that the algorithm sorts the array in decreasing order.

Exercise 4
Consider the **searching problem:**
Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$ stored in array $A[1:n]$ and a value x.
Output: An index i such that x equals $A[i]$ or the special value NIL if x does not appear in A.
Write pseudocode for **linear search**, which scans through the array from beginning to end, looking for x. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

Solution.
The pseudocode is as follows:

```
LINEARSEARCH(A, n, x)
1  idx = NIL
2  for i = 1 to n
3      if A[i] == x
4          idx = i
5          break
6  return idx
```

We claim that the following loop invariant holds: At the start of the i-th iteration, none of $A[1], A[2], \ldots, A[i-1]$ are such that $A[i] = x$.

- **Initialization:** At the start of the first iteration, the stated collection of indices is the empty set, and so the statement holds trivially.
- **Maintenance:** If the invariant holds at the beginning of iteration i, then during this iteration one of two things can happen. If $A[i] = x$, then the "if" block of lines 3-5 executes, and the loop terminates early with $idx = i$. Otherwise, no variables are changed and iteration $i + 1$ begins, from which we conclude that none of $A[1], \ldots, A[i]$ equals x.
- **Termination:** Termination is always guaranteed since the loop runs for at most n iterations. Furthermore, by the stated invariant, at the end of the loop (start of an "imaginary" $n + 1$ iteration), x is not contained in $A[1:n]$. This means that idx has remained equal to NIL (see the point above regarding no changes in variables if lines 3-5 doesn't execute), and this is correctly returned. The only remaining case is for the loop to terminate early. As we've seen above, this only happens if x is found at *some* position i, in which case idx contains i, which is then immediately returned.

Therefore, whenever x is not in $A[1:n]$ the algorithm returns NIL. Whenever x is in $A[1:n]$, the algorithm returns *an* index i such that $A[i] = x$.

Exercise 5
Consider the problem of adding two n-bit binary integers a, b, stored in two n-element arrays $A[0:n-1]$ and $B[0:n-1]$, where each element is either 0 or 1, $a = \sum_{i=0}^{n-1} A[i]2^i$, and $b = \sum_{i=0}^{n-1} B[i]2^i$. The sum $c = a + b$ should be stored in binary form in an $(n+1)$-element array $C[0:n]$, where $c = \sum_{i=0}^{n} C[i]2^i$. 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.

Solution.
The algorithm is as follows:

ADD-BINARY-INTEGERS(A, B, n)
1 $carry = 0$
2 C = empty array of size $n + 1$
3 **for** $i = 0$ **to** $n - 1$
4 $\quad C[i] = (carry + A[i] + B[i]) \mod 2$
5 $\quad carry = \lceil \frac{carry + A[i] + B[i]}{2} \rceil$
6 $C[n] = carry$
7 **return** C

As an outline of a correctness proof, the invariant could be that at the start of the i-th iteration (0-indexed), the binary number that has as most significant digit $carry$, and then as least significant digits $C[i-1], C[i-2], \ldots, C[0]$ (ordered from most to least significant) equals the sum of the binary numbers formed by $A[0:i-1]$ and $B[0:i-1]$.

2.2 Analyzing algorithms

Exercise 1
Express the function $n^3/1000 + 100n^2 - 100n + 3$ in terms of Θ-notation.

Solution.
The function is clearly $\Theta(n^3)$, since it is a polynomial of highest degree 3.

Exercise 2

Exercise 4
How can you modify any sorting algorithm to have a good best-case running time?

Solution.
Checking if an array is already sorted can be done in linear time: simply iterate through the array and verify each element is not smaller (or not larger) than the previous one. Therefore, if one includes such a check at the beginning of any sorting algorithm, and returns the array without performing further processing if it is indeed already sorted, the best case becomes $\Theta(n)$.