# Exercises

## 2.1-3 Consider the searching problem: Input: A sequence of n numbers A = {a1, a2,..., an} and a value v. Output: An index i such that v = A[i] or the special value NIL if v does not appear in A. Write pseudocode for linear search, which scans through the sequence, looking for v. Using loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.We have a list of botanic plants and we want to sort them in alphabetical order.

**Initialization**: The subarray is consisted of just one single element A[0] and he will be checked just befire the first test in the loop header.  
**Maintenance**: Showing that each iteration maintains the loop invariant is pretty simple knowing the fact that the loop body is consisted of only one if statement and if the current element A[index] is equal to V then we will return the index and the loop will stop. Otherwise, he will terminate through all values to A.Length – 1 and will check if they are equal to V.  
**Termination**: The loop will start from 0, in other words first first element of array A, and will continue untill current element is equal to V or untill reaches his end position A.Length.

## 2.1-4 Consider the problem of adding two n-bit binary integers, stored in two n-element arrays A and B. The sum of the two integers should be stored in binary forn in an (n+1)-element array C. State the problem formally and write pseudocode for adding the two integers.

**Initialization**: A[] and B[] are consisted of [0...Length – 1] elements. The initialization value is [Length – 1].

**Maintenance**: The for llop works by moving A and B[Length -1], [Length - 2] and so on each iteration untill it reaches [Length – Length == 0]. On each iteration to the left, we summarize the numbers from A and B array’s current position and save it in C[position + 1]. So at any point, the right-side from the current position, the digits are already summed – [position + 1], [position + 2] ... [Length -1].

**Termination**: At each iteration we write two numbers – one in the current position of the A/B[position + 1], and one in the position – 1. That’s because on the next iteration, the element at [current position – 1] is replaced except in the last iteration. That’s because our C[] is consisted of A/B[].Length + 1 elements.